MONASH UNIVERSITY

THESIS ACCEPTED IN SATISFACTION OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

ON................ 27 January 2004 ....

[redacted]

Sec. Research Graduate School Committee

A Study of Single Laser Interferometry-based Sensing and Measuring Technique in Robot Manipulator Control and Guidance

Volume 1

Submitted By

Pek Loo TEOH
Bachelor of Engineering (Monash University – Australia)

A thesis submitted in fulfillment of the
requirements for the degree of

Doctor of Philosophy in Engineering Science

Department of Mechanical Engineering
Monash University

September 2003

Statement

This thesis contains no material which had been accepted for the award of any other degree in any university, and, to the best of my knowledge and belief, it contains no material which has been previously published or written by any other person, except where due reference is made in the text of the thesis.

Melbourne, September 2003

[redacted]

....

Pek Loo TEOH

Acknowledgements

The research presented in this thesis was conducted in the Robotics & Mechatronics Research Laboratory (RMRL) at the Department of Mechanical Engineering, Monash University. The author wishes to express his sincere gratitude towards his supervisor Associate Professor Bijan SHIRINZADEH for providing the research facilities and his guidance and assistance during the course of this study. The author wishes to extend his appreciation to Dr. Gürsel ALICI. The author also wishes to express his appreciation to Mr. Hugh VENABLES, Mr. Long GOH, Mr. Eric WIRTH, Mr. Arthur TURNOCK, and Mr. Mark SYMONDS for the construction of mechanical and electronic components. This study was partly funded by the Australian Research Council (ARC) – file number: R001078/82, Monash University and the Harold Armstrong Research Fund.

The author would also like to thank Mr. Andrew McCONVILLE, Mr. Ryan STEVENSON, Mr. Alan GAN, and all his housemates for their friendship and encouragements during the course of the research work.

Last but not least, the author would like to express his gratitude to his parents and his girlfriend. He will always remember their encouragement and support.

Abstract

With the ever growing requirement for accuracy in many applications, specially the dynamic accuracy required by applications such as machining, laser cutting, laser welding, etc., it has become obvious that robot calibration techniques are crucial. Researchers have shown that in order to improve the dynamic response of robot manipulators, their performance must be measured accurately under motion. Laser interferometry-based tracking techniques for dynamic position and orientation (pose) measurements has been proposed recently. It is well known that such measurements will allow the identification of the dynamic model and its incorporation into robot control will dramatically improve the performance of the robots. Moreover, by externally sensing the pose of the robot end-effector on-line, a closed-loop control can be established.

This project is undertaken to investigate issues associated with the single beam laser interferometry-based dynamic measurement technique for the tracking and dynamic position measurements of moving targets. The research also aims to establish a methodology for the laser-based closed-loop control and guidance of robot manipulators. A Laser Interferometry-based Sensing and Measuring (LISM) assembly was developed for this purpose. The physical make-up, functionality, measurement and analysis techniques employed for each sub-system have been presented. The behaviour and characteristics of the sub-systems on the performance of the LISM apparatus have also been studied. Experiments on closed-loop control and guidance of robot manipulators have been conducted with a robot manipulator and the experimental results are presented. Conclusions on the performance of the proposed methodologies are drawn based on the results.

An orientation measurement strategy has also been established to examine the feasibility of dynamic robot orientation measurement using a specially developed Gimbal apparatus. In addition, an error model associated with uncertainties of the position and orientation measurements has been characterised. The knowledge of uncertainties allows the characterisation and specification of individual sub-system for the experimental apparatus to improve the performance.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Problem Description

During the past two decades, there has been an increase in the number of applications for robots in the manufacturing and service industries. Robots are used extensively in applications such as parts handling, machining, welding and assembly operations. Three main parameters, resolution, repeatability and accuracy, are used to describe the positioning accuracy and the path following capability of the robot manipulators. Traditional techniques for the programming of robots have utilised the teach-by-showing method where the robot is jogged to each of the desired Cartesian positions and the joint positions are recorded in the memory of the controller [1]. The program then read these joint positions during playback. In this approach, the critical factor is the repeatability of the robot. In fact, the repeatability of industrial robots is at least a magnitude better than their accuracy.

Robot manipulators today are required to perform complex assembly and high precision path following applications. These applications depend significantly on the absolute accuracy of the robotic devices. The utilisation of off-line programming techniques in developing robot application programs has further emphasised the need for greater absolute accuracy in robots. For off-line programming, robot paths are calculated for a particular task and a program containing a number of poses along these paths is generated and downloaded to the robot controller. These poses are specified in Cartesian co-ordinates and not from taught positions. The robot accuracy becomes significant in such an approach.

The application of robots acquiring measurements of manufactured parts also requires improved robot accuracy. In such an application, a measurement device is attached to the end-effector of the robot, which moves the device into position to take a measurement. All measurements are related back to the end-effector's pose. The measurement accuracy will only be as good as the robot's positioning accuracy. Furthermore, the need to develop performance measurement techniques for the assessment of the suitability of various robots to different tasks again focuses on the importance of absolute accuracy. According to the ISO 9283 standard produced by the International Standards Organisation, path and velocity accuracy are key characteristics in this assessment [2].

Various robot control strategies have been designed to improve robots' position accuracy and trajectory following capability. This is achieved by finding the difference between the desired and actual position quantities through feedback. These control strategies can be generally separated into joint-based control and Cartesian-based control. Joint-based control schemes utilise position quantities expressed in joint space whereas Cartesian-based control schemes utilised position quantities expressed in Cartesian space. Joint-based control is widely used in the industry. Cartesian-based control is still in the research stage due to the unavailability of Cartesian-based feedback devices.

The aim of this research is to investigate the issues associated with laser interferometry-based closed-loop control and guidance of robot manipulators. Control variables governing the effectiveness of the strategies are investigated and studied.

1.2 Robot Kinematic Model and Calibration

The first step in any control design is to accurately model the robotic devices to be controlled. The variables of robotic devices that need to be modelled include the actuator characteristics, the kinematic parameters and the inertial parameters. Accurate modelling of these parameters is important to achieve high position accuracy and high performance position control.

The kinematic model in robotics is the mathematical representation of the geometry of the robot mechanism and the required configurations of that mechanism to position the end-effector at a desired pose. Firstly, the robot mechanism must first be described mathematically as a kinematic chain of rigid links connected by means of revolute or prismatic joints using a kinematic model. The most common approach involves the formation of a transformation matrix that contains joint and link values known as the Denavit-Hartenberg (DH) parameters [1, 3]. The accuracy of the DH parameters determines how accurately the robot end-effector's position can be calculated. However, no robotic devices are ever manufactured perfectly. There exist slight variations in DH parameters because of the occurrence of the following errors:

- Geometric errors - Errors in robot geometry caused by the inaccuracies in the manufacture and assembly of the components within robot. These errors are not taken into account in the kinematics calculations and the ideal values of the parameters are used. Further, flexing and bending of the robot arm due to external loading causes errors in the calculations of the actual location of the end-effector. Moreover, the materials used for the manufacture of robots are sensitive to temperature change, which in turn vary the geometry of the robot. Heat sources include motors, friction and environmental changes. These errors are non-linear and they cannot be compensated for by a simple scaling factor.
- Non-geometric errors – Errors that occur during robot motion, where there will be inertial loading and dynamic resonance caused by the motion. These errors are difficult or expensive to model. They are usually ignored during a point to point operation when the path is not important. During a path following operation, these errors become significant. Improper calibration, sensor inaccuracies, drive train backlash, etc., are other non-geometric errors. These errors are difficult to identify and model.

These variations can result in end-effector inaccuracy of several millimetres for many industrial robots. Robot kinematic calibration has been recognised as a basic process to improve the position accuracy for any robotic device [4].

Robot kinematic calibration is the method used to improve the robot absolute accuracy without modifying the mechanical unit by compensating for the variations of kinematic parameters using an external measuring device. The most difficult aspect of kinematic calibration is that many measurements throughout the workplace are required to establish compensation values for the variations of kinematic parameters and to obtain accurate measurements of the end-effector position. Traditional methods of calibration have involved calibration fixtures with precision points and special position measurement apparatus. Examples of these methods include extensible ball bar, dial indicators, linear variable differential transducers (LVDT), Latin Square Ball Plate methods and co-ordinate measurement machines. [5, 6]. All of these methods are time-consuming or do not provide dynamic measurements. Further, constraints are being placed on the robot that will affect and limit the range of the robot's movement. Modern methods include RoboTrak with string-pull device [6, 7], photogrammetry method with CCD cameras [5, 6, 8] and theodolites triangulation methods [9]. Dynamic measurements can be obtained using the above methods. However, considerable amount of time is needed for setting up the above measurement equipment and these techniques will only provide high accuracy when performed by a skilled operator. Recently, laser position measuring methodologies have been proposed [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]. These measurement techniques can provide real-time, high accuracy, non-contact and automatic dynamic measurements in a volume typical of robot workspaces. This research aims at investigating the dominant variables in laser interferometry-based sensing and measuring (LISM) technique. Although commercial laser position measuring systems are available, some proprietary information regarding the systems could not be obtained, thus hinders the investigation process. Therefore, an experimental LISM apparatus was established for this study. The effect of the dominant variables on the accuracy, repeatability, speed and target following capability of the LISM apparatus were examined. Furthermore, orientation measurement capabilities are not available in the commercial system and there is a lack of experimental results on the performance. Orientation measurement methodology using an experimental Gimbal unit was being studied. The capabilities of the proposed methodology are reported through experimental verification. This experimental LISM apparatus and experimental Gimbal unit will improve the fundamental understanding and knowledge of the hardware integration and control techniques on the LISM technique for position and orientation (pose) measurement.

It has been shown that the rigid body dynamics of robotic systems could create substantial trajectory errors during fast motion [4]. Control techniques that use the dynamic model of the robot have been designed to predict actuator commands corresponding to a motion. The dynamic model consists of the kinematic parameters as well as the motor parameters and the inertial parameters of payload and robot links. A robot's trajectory errors can be reduced by using accurate parameters in the model. However, obtaining an exact dynamic model is difficult as there will be some modelling errors as well as errors caused by unexpected disturbances not included in the model. This will also involves considerable amount of trial and error to obtain an accurate estimation. Significant amount of computation power and a high sampling rate are also required to improve the stability of the techniques [1, 4, 23]. Furthermore, a robot's trajectory is usually specified in Cartesian space whereas the robot control is implemented in joint space. The incompatibility between Cartesian and joint space variables introduces errors in the techniques [23]. This nullifies the extra computational power used in the control.

In this study, the experimental LISM apparatus is employed to investigate the issues associated with the closed-loop control of robotic devices. Laser interferometry-based position feedback control and laser interferometry-based guidance control strategies are proposed to provide an online control of robotic devices.

The specific aims of this thesis include:

- to identify the key variables in the LISM technique and characterise the effects of these variables on the performance of the LISM apparatus;
- to establish an orientation measurement technique;
- to characterise and develop an error model associated with uncertainties of the pose measurements obtained from the experimental LISM apparatus;
- to investigate issues, requirements, and factors associated with ground truth feedback sensing, and establish a methodology for closed-loop control of robot manipulators;
- to establish a methodology for laser interferometry-based guidance of robot manipulators.

1.3 Principal Contributions

To assist in assessing the thesis, the principal contributions presented in this thesis are summarised as follows:

1. Experimental LISM apparatus for the tracking and dynamic measurement of a moving target with an unknown trajectory has been developed. It is shown that the PC-based LISM apparatus has slow update rate due to limited real-time capability of Windows operating system. The key variables affecting the tracking performance of the LISM apparatus is the noise in the PSD sub-system and the low update rate. Higher laser beam power and motors with higher resolution and accuracy are also required to improve the tracking performance.
2. Dual PSD-based orientation measurement methodology using the specially developed Gimbal unit for the measurement of the relative orientation of the retroreflector with respect to the laser beam is presented. It is demonstrated experimentally that the dual PSD-based orientation measurement methodology is feasible in measuring the pitch and yaw angles of the retroreflector accurately. The

accuracy of roll angle measurement is affected by the initial fixed beam offset applied in the set-up. The proposed approach can also be used to rotate the target retroreflector, such that the line of sight for the incident laser beam is maintained. This approach effectively removed the limitation of the small incident acceptance angle for the retroreflector.

3. The establishment of the kinematic model for the LISM apparatus is presented. It is demonstrated experimentally that accuracy of the dynamic position measurements obtained from the calibrated kinematic model improved significantly.
4. The sources of errors associated with the LISM apparatus and the Gimbal unit have been identified. Dominant variables associated with the uncertainties of the measurements obtained from the experimental LISM apparatus and the Gimbal unit has been established. The knowledge of these dominant variables allows the correct selection of hardware components to improve the measurements accuracies.
5. Closed-loop control and laser interferometry-based guidance of robot manipulators have been established. The behaviour and effects of the proposed methodologies on the robot end-effector have been investigated through experimental results. It has been shown that the path following and positioning accuracy of the robot end-effector can be improved with the implementation of the proposed methodologies.
6. The major limitation of the proposed closed-loop control and laser interferometry-based guidance of robot manipulators has been recognised to be the communication delay between the LISM control unit and the robot controller.

These contributions have been published in 5 conference papers, and 2 journal papers. Full contents of these papers are provided in Appendix G.

1.4 Organisation of this Thesis

Chapter 2 presents a review of the previously published work related to the background of the problem addressed in this thesis and various techniques, the control algorithms, and components related to this study.

Chapter 3 presents the establishment of the LISM apparatus, and the experimental verification conducted to determine the dominant variables of the apparatus and the effect of these variables on the performance.

Chapter 4 provides the study of orientation measurement technique using an experimental Gimbal unit. The capabilities of the proposed methodology are reported through experimental verification.

Chapter 5 presents the kinematic model of the LISM apparatus, which is essential for the measurement of the positions of the end-effector. Analysis of the uncertainties of the measurement acquired by the LISM apparatus and Gimbal unit are evaluated and conclusions are drawn from the analysis providing an insight into the accuracy of the LISM apparatus and the Gimbal unit.

Chapter 6 presents the design and development, together with the evaluation of the proposed closed-loop control strategy in the control of the position of robotic devices. The experimental implementations on a 6-axis robot manipulator are presented.

Chapter 7 presents the design of the proposed laser interferometry-based guidance control strategy for robot control. Experimental results providing a fundamental understanding of the effects of the methodology or robot performance are also discussed.

Chapter 8 presents conclusions and recommendations for future work.

Chapter 2

Background and Literature Review

2.1 Introduction

This chapter presents the background of the sub-systems required to establish a laser interferometry-based dynamic measurement technique. This is followed by a review of the laser-based dynamic measurement techniques with single, dual or multiple beams established over the past decade. The underlying concept and capabilities (e.g. speed, accuracy, and range), as well as any special features of the various techniques are reported.

2.2 Laser Interferometer

An American physicist, Albert A. Michelson, first invented the laser interferometer in 1880 [24]. The underlying concept of laser interferometry is based on the wave theory of light, which states that light travels as a sinusoidal wave with wavelength, λ as shown in Fig. 2.1. Fig. 2.2 shows a schematic diagram of a laser interferometer. As shown, a laser beam is emitted from laser source L. Beamsplitter B reflects about half of the power of the laser to the fixed reference retroreflector R. The remainder of the laser passes through the beamsplitter B and strikes the target retroreflector T. The beams from R and T are combined at the receiving optics point O. As T moves, there is a progression of light and dark bands (fringe shift) produced by the constructive and destructive interference of the peaks and the valleys of the light waves (Fig. 2.3). Photo detectors and computer-controlled counters are used to pick up the beam and determine the number of shifts from light to dark band. The relative distance moved by the target retroreflector can be measured with a resolution equal to half of the wavelength of the laser, λ/2.

However, significant inaccuracies could be introduced in such a technique since the computerised counter would count fringe shift when T is moving away from (or towards) the interferometer and at the same time moves infinitesimally towards (or away from) the interferometer due to vibration. This design does not take into account the directional changes of the target motion, and hence appropriate adjustments need to be made to the technique as described in the following sections.

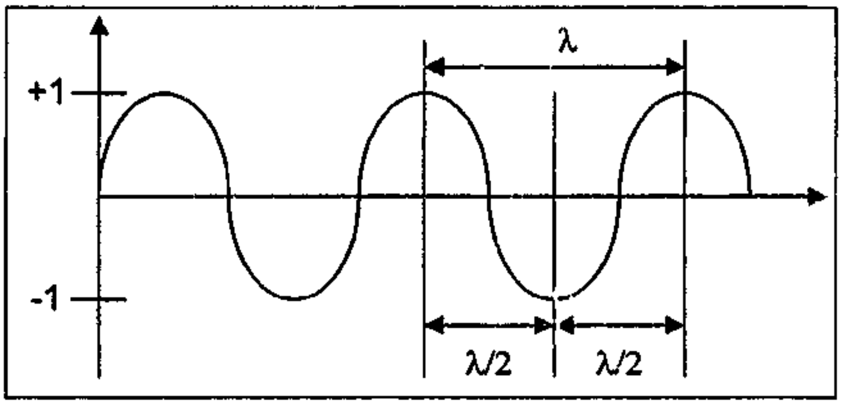


Figure 2.1: Sinusoidal representation of light wave

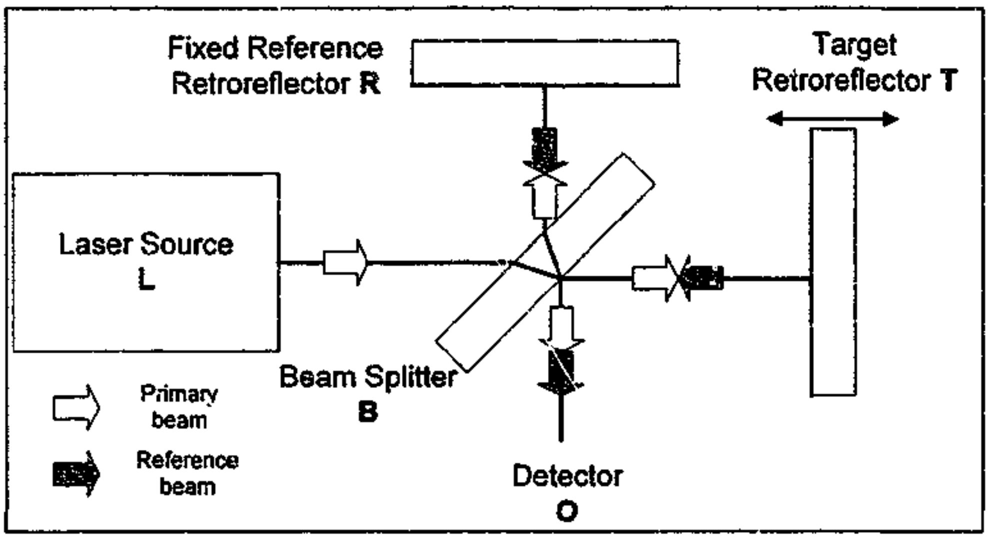


Figure 2.2: Schematic diagram of a laser interferometer

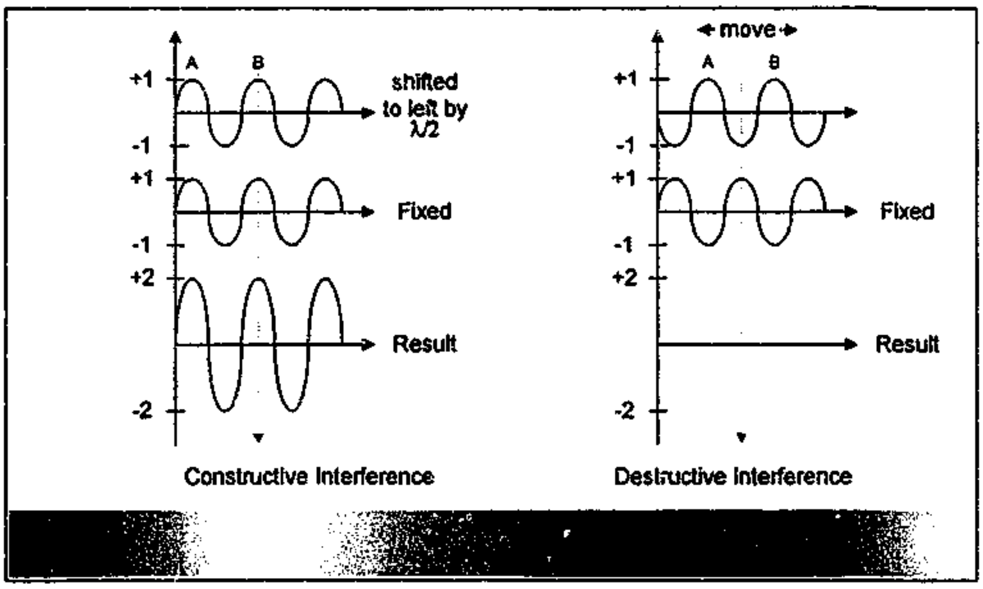


Figure 2.3: Constructive and destructive interference pattern

2.2.1 Single Frequency

For a single frequency laser interferometer, the laser beam is polarised by the beam splitter into its horizontal and vertical components. Two detectors are used to detect the reflected beams at the receiver – one through a 0° polariser (sine detector) and the other a 45° polariser (cosine detector). By comparing the phase relationship of the two waveforms as shown in Fig. 2.4, the counting electronics can determine whether a fringe shift was caused by target T moving towards or away from the interferometer [24].

2.2.2 Heterodyne (Dual Frequency)

A special variant of single frequency laser employing the Zeeman split is used in the dual frequency, or heterodyne interferometers. Two frequency components, F1 and F2, with a difference of approximately 1.8 MHz are generated by a magnetic field. They are polarised at right angles to each other to allow them to be separated by optical polarisers [24, 25]. Fig. 2.5 shows a heterodyne interferometer.

As the beam passes through the interferometer, F_2 is directed to the fixed reference retroreflector at R where it is reflected and directed to the detector D. F_1 passes through polarising beam splitter to target retroreflector T. As T moves, frequency F_1 just appear to shift based on the Doppler Principle, yielding a frequency change of ΔF_1. Movement of T towards the interferometer will increase F_1, causing a positive change in F_1. Alternatively, any movement of T away from the interferometer will decrease F_1, causing a negative change in F_1. At the detector D, F_1 and F_2 are recombined to give the interference signal with a frequency equal to:

$$\text{Interference signal} = [F_2 - (F_1 \pm \Delta F_1)] \qquad (2.1)$$

Electronics in the laser head generates a reference signal with frequency equal to $F_2 - F_1$. The interference and reference signals are combined to give the frequency change ΔF_1, which represents the rate of change of the target T displacement. Integration is carried out to yield the relative displacement of the target T.

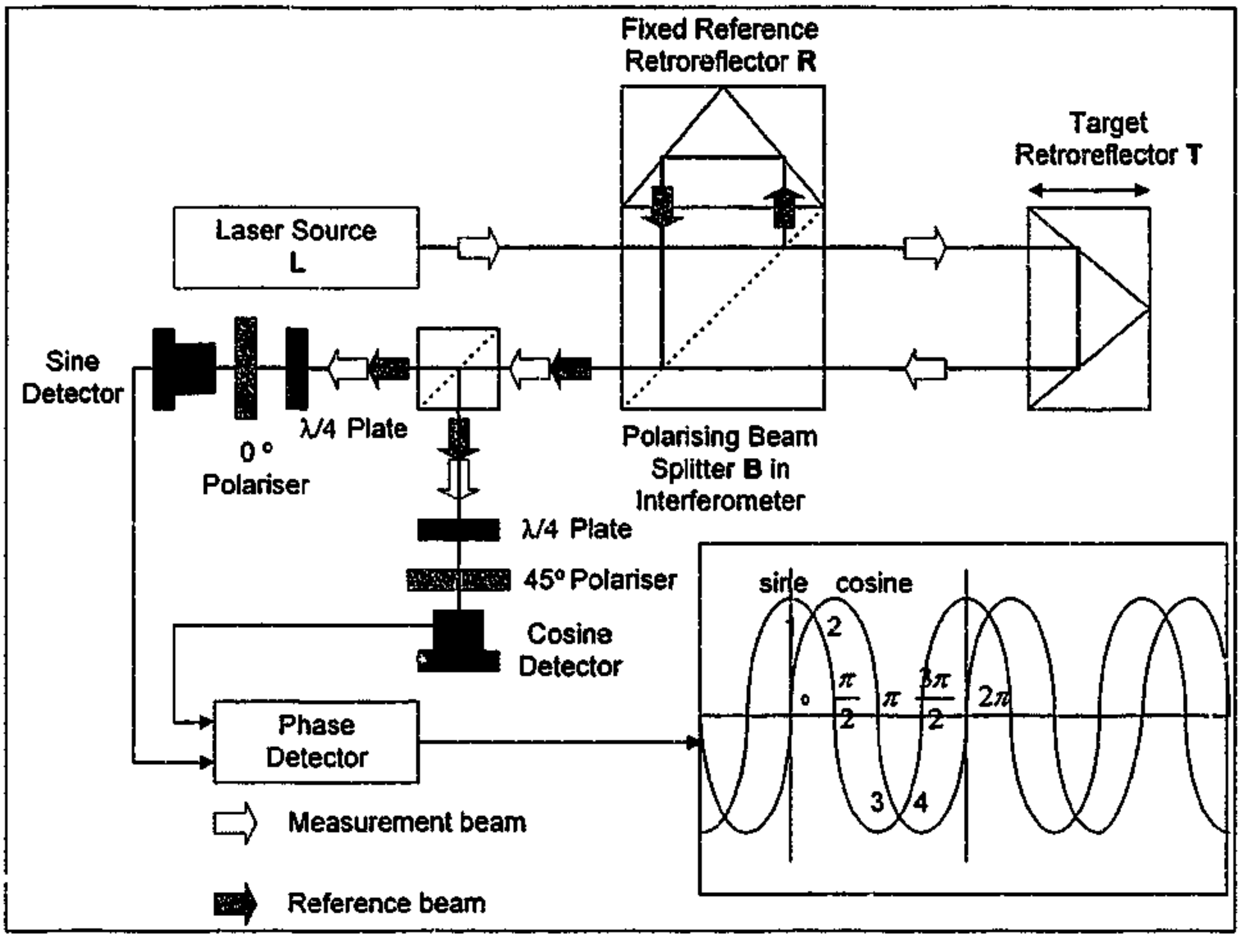


Figure 2.4: Single Frequency Laser Interferometer

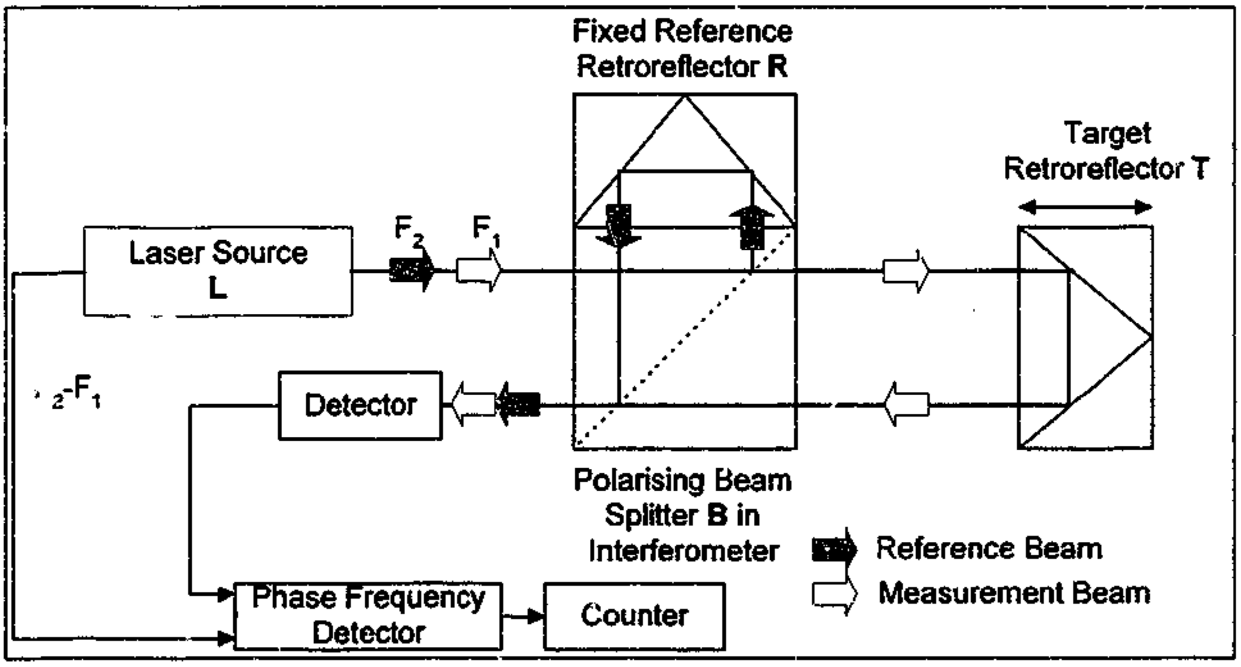


Figure 2.5: Heterodyne laser interferometer

2.2.3 Advantages and Disadvantages of Laser Interferometer

The most significant advantage of the laser interferometer is the higher measuring resolution obtained. A single frequency laser interferometer has an inherent resolution of λ/8 or 0.08 μm and a heterodyne laser interferometer has an inherent resolution of λ/4 or 0.16 μm.

One drawback of laser interferometer is the variation of the laser's wavelength due to changes in the operating environment (e.g. temperature, pressure and humidity). Accuracy of the measurement will degrade with the variation in wavelength. Additional sensors such as temperature, pressure and humidity sensors are necessary to provide on-line adjustments to the wavelength. Further, the interferometer measures only the relative change in target displacement, not the absolute distance of the target.

2.3 Position Sensing Detectors

Silicon Position Sensing Detectors (PSDs) are photodiodes used to detect and record the position of a laser beam incident on the detector surface. These detectors operate by the absorption and conversion of the photon energy of the laser into an electrical current. Electrodes are attached to the detectors and the currents measured from each electrode can be employed to give the positions of the centroid of the laser beam relative to the centre of the detector [26]. There are two types of detectors commonly used for beam position measurement, the segmented detector and the lateral effect detector [27, 28].

2.3.1 Segmented Detector

The segmented detector is a uniform disc of silicon with small gaps across the surface to divide the sensing surface into two or four equal sectors. Each sector produces a current proportional to the amount of light incident on it. If the beam is centred, each sector will provide an equal amount of photo-current. Fig. 2.6 shows a schematic diagram of the surface of the four segments detector for two-dimensional measurements.

The equations used to calculate X and Y displacements of the beam are:

$$X = \frac{(q_2 + q_4) - (q_1 + q_3)}{q_1 + q_2 + q_3 + q_4} \qquad (2.2)$$

$$Y = \frac{(q_1 + q_2) - (q_3 + q_4)}{q_1 + q_2 + q_3 + q_4} \qquad (2.3)$$

where q1, q2, q3, and q4 are the current outputs of the detector's quadrants.

One limitation of the segmented detector is that the position signals are meaningful only when the beam falls on all segments. The photocurrents also become non-linear with displacements of more than 10 % of the beam radius. Only small beam displacement can be measured accurately. Segmented detectors are excellent devices for applications such as beam centering [28].

2.3.2 Lateral Effect Detector

The lateral effect detector is a continuous photodiode with no gaps. The electrodes are connected in such a way that the opposite pair yield photo-currents that are processed to give the displacement of the beam [28]. Fig. 2.7 shows the diagram of this detector type for two-dimensional measurements.

The equations used to calculate X and Y displacements of the beam are:

$$X = \frac{q_1 - q_3}{q_1 + q_3} \tag{2.4}$$

$$Y = \frac{q_4 - q_2}{q_4 + q_2} \tag{2.5}$$

where q_i (i = 1, 2, 3, 4) represents photo-current recorded by electrode i.

However, photo-currents become non-linear with beam displacement away from the centre. Linearisation of detector response can be performed through calibration to obtain a precise measurement across the entire sensing area. This linearisation enables the lateral effect detector to measure the position of a beam over its entire surface. One limitation to this type of detector is that each lateral effect detector has its own unique set of calibration correction factors for linearisation. Therefore, calibration has to be performed for each lateral effect detector or change in detector set-up.

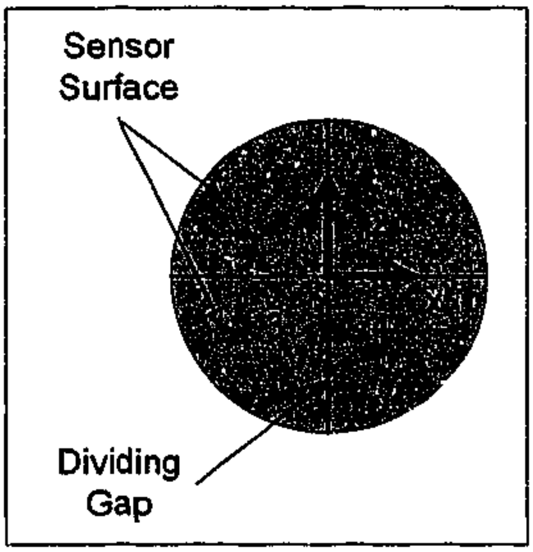


Figure 2.6: Four segments detector

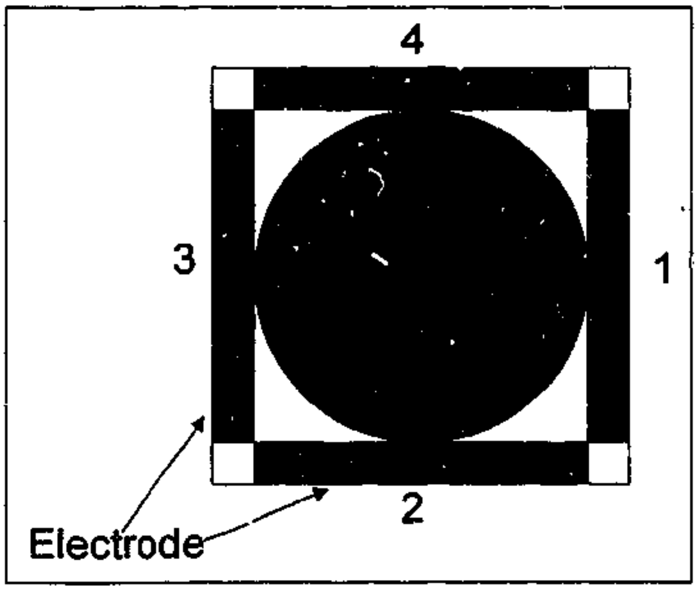


Figure 2.7: Dual axis lateral effect detector

2.4 Reflecting Target

The most common reflecting target used is a specially arranged mirror assembly called the air-path type retroreflector. Fig. 2.8 shows a diagram of a standard air-path type retroreflector. It is made up of three mirrors with high reflective index assembled in such a way that they are orthogonal to each other. The intersection point of the three mirrors will be the centre of the retroreflector. An ideal retroreflector will have the properties that a light beam incident on any point on the retroreflector, regardless of its orientation, will be reflected through 180°. The reflected beam will travel back parallel to the incident beam.

The centre of the retroreflector is ideally the midpoint between the incident and the reflected beams. The retroreflector is attached to the end-effector of a robot manipulator, and it is usually lightweight to minimise loading effects on the robot. It has a light incident angle range of ±30° from the centre axis.

The second type of reflecting target is known as the cat's eye retroreflector [19, 29]. Fig. 2.9 shows a diagram of the cat's eye retroreflector. Commercial cat's eye retroreflectors consist of two glass hemispheres with similar refractive indices joined together with a thin layer of adhesive material. The incident aperture is reflection free and the other side is coated with a reflecting material. A beam entering at any point will be refracted and incident on the reflecting material. The reflected beam will once again be refracted and exit the retroreflector. Ideally, the reflected beam is parallel to the entered beam. This retroreflector has a light incident angle range of about ±75° from the centre axis.

Other types of reflecting targets include solid glass retroreflector and right-angle glass prism. Fig. 2.10 shows a diagram of a right-angle glass prism. Similar to the cat-eye retroreflector, the incident aperture is reflection free and a beam will be reflected parallel to the incident beam based on the principle of total internal reflection [27]. However, due to the refraction at the incident aperture surface, a lateral distance error occurs. Thus, these types of retroreflectors have a very limited incident angle range to ensure accurate measurements.

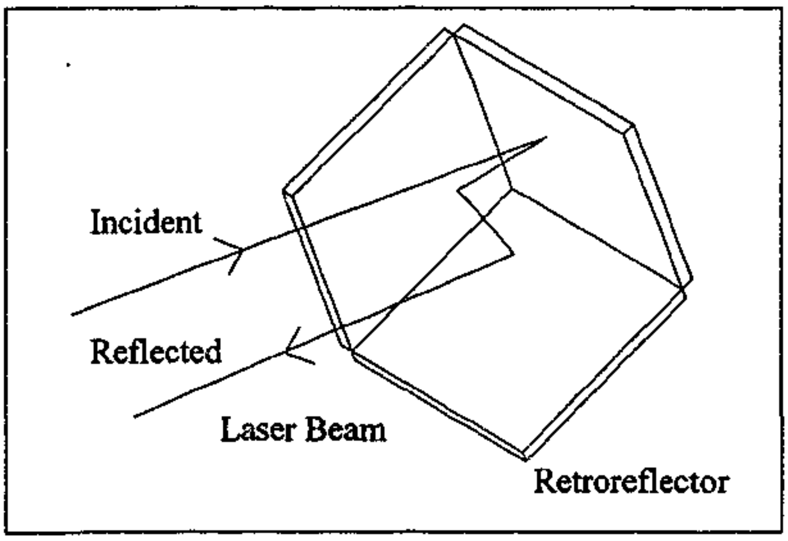


Figure 2.8: Air-path type retroreflector

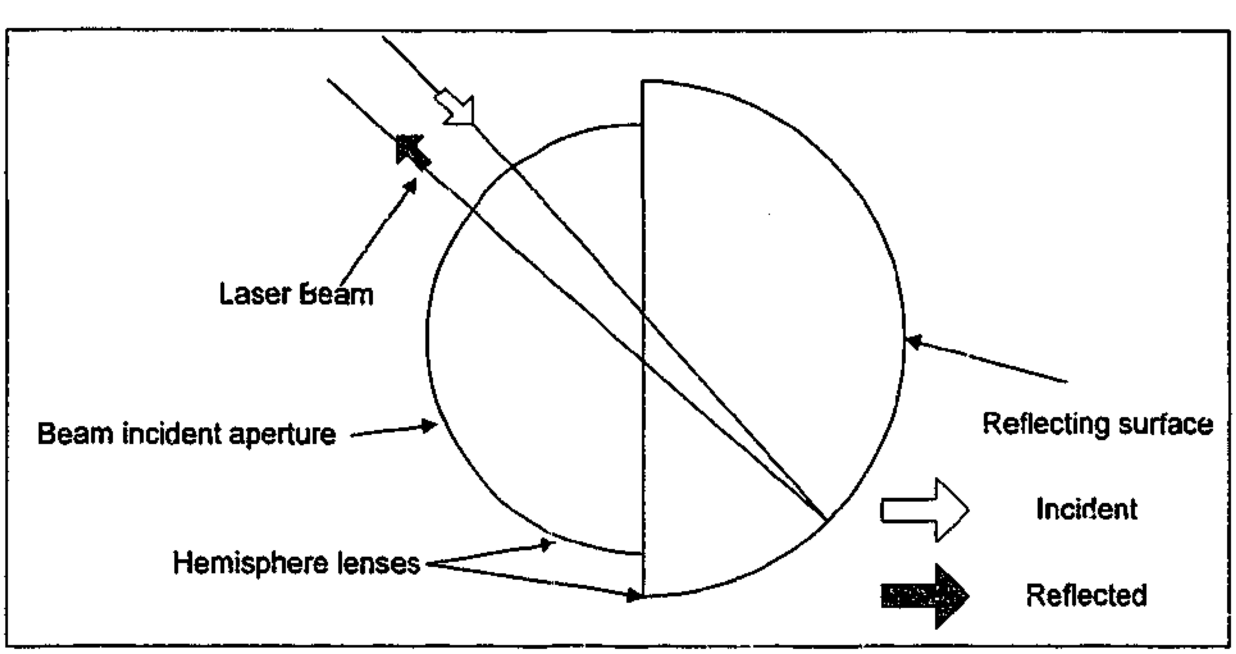


Figure 2.9: Cat's eye retroreflector

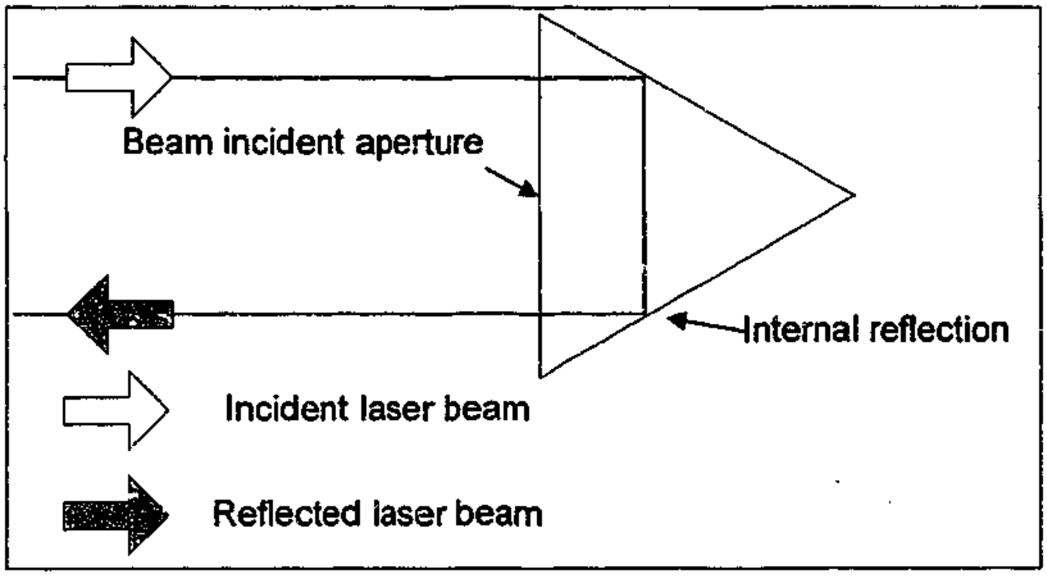


Figure 2.10: Right angle glass prism

2.5 Beam Steering Mechanism

Beam steering mechanisms (BSM) are used to keep the laser beam directed on the target. In this technique, a single mirror attached to an actuation mechanism (e.g. galvanometer, stepper, servo motors) is the most common type of BSM used to direct the laser beam towards the target. Two axes of rotations are needed to direct the laser beam to any spatial location. Position sensors (e.g. potentiometer, optical shaft encoder, or capacitive position transducer) are attached to the actuators to read the angular displacement. Fig. 2.11 shows the figure of a typical BSM.

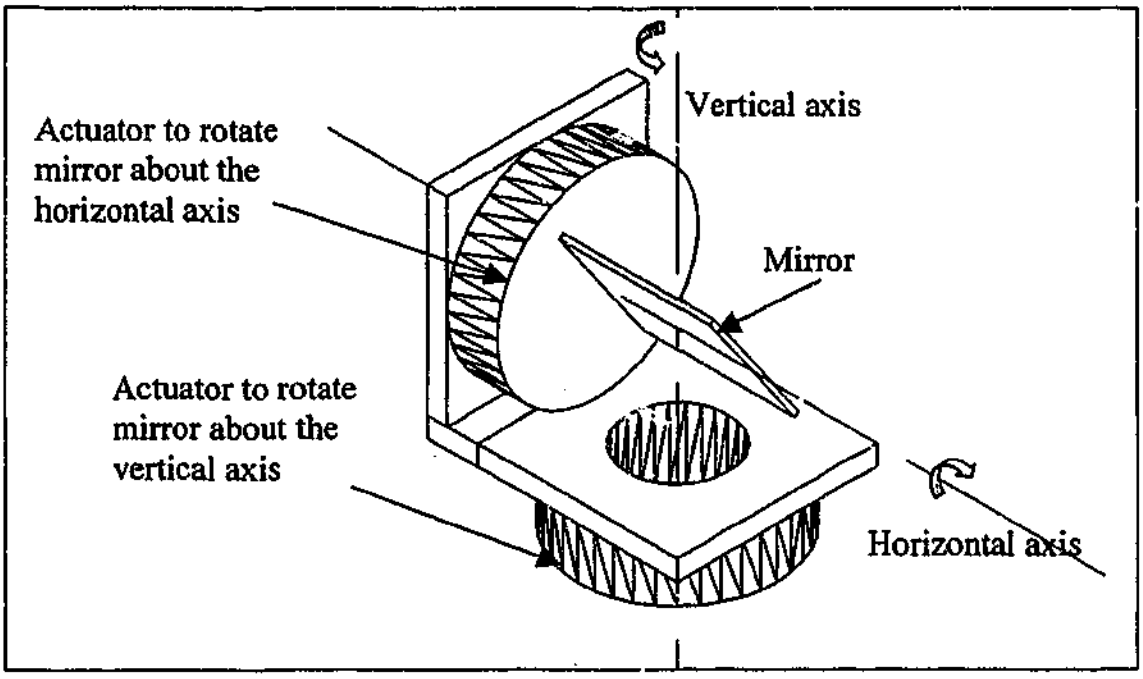

Actuator to rotate mirror about the horizontal axis
Vertical axis
Mirror
Actuator to rotate mirror about the vertical axis
Horizontal axis

Figure 2.11: A typical beam steering mechanism

2.6 Review of Laser-based Dynamic Measurement Techniques

Laser-based dynamic measurement techniques involve the tracking of the reflecting target attached to the end-effector of the robot or other positioning devices. The laser beam is directed to the reflecting target by the BSM. Ideally, the laser beam hits the centre of the target, causing no offset between the incident and reflected beam. When the target starts moving, the laser beam does not hit the centre of the target, which results in a displacement between the incident and the reflected beam. This displacement is measured by the PSD and constitutes the tracking offset. The laser-based dynamic measurement controller minimises the tracking offset by turning the BSM, thus enabling the laser beam to follow arbitrary movements of the target. Position and orientation information of the target can be obtained by analysing data sampled from various sensors. The control architecture of the laser-based dynamic measurement technique can be divided into 5 functional groups:

1) Sensing of target displacement in the direction of the beam;
2) Sensing of tracking offset along the plane perpendicular to the direction of the beam;
3) Sensing of the target orientation;
4) Directing the beam towards the target to maintain tracking;
5) Calculation of the target position and/or orientation.

This section presents a review of published works on laser-based dynamic measurement techniques. These techniques can be broadly classified into two categories: interferometric and non-interferometric. Non-interferometric based techniques are included due to the similarity in principle and configurations. Central issues include the physical make-up and functionality of the hardware, the measurement and analysis techniques employed, and where appropriate, control strategies for the technique to provide for the 5 functional groups as stated above.

2.6.1 Dual Laser Interferometry-Based Tracking and Measurement Technique

The first laser interferometry-based dynamic measurement technique utilising two laser beams, known as the Optical Scanner Measurement Technique, was proposed by Gilby and Parker [15]. Fig. 2.12 shows a general layout of this measurement technique. This technique involves tracking an air-path type retroreflector using two similar static measurement sub-systems (Fig. 2.13), reported as tracking heads. A quadrant detector (4-segment PSD) is used to detect the tracking offset of the retro-reflected beam. The laser beam from each unit is directed towards the retroreflector by two optical scanners mounted in orthogonal rotation axes. The optical scanner is made up of a plane mirror driven by a moving iron galvanometer. A capacitive position transducer is used to provide angular displacement. The knowledge of the scanner angles and tracking offsets allows the determination of line of sight. The combination of two lines of sights from each tracking head with the relative position between the heads provides a means to calculate the position of the target by triangulation. This method is reported to be capable of providing dynamic position measurements of target moving at a maximum velocity of 5 m/s with accuracy better than ±0.01 mm in a 1 m^3 working volume [15].

Further developments of the Optical Scanner Measurement Technique produced the Optotrac Measurement Technique [16]. This is based on the same principle as the Optical Scanner Measurement Technique utilising triangulation to determine the position of the object being tracked. It is claimed to be capable of obtaining the static and dynamic performance of robots to ISO standards as well as being fully portable and self-calibrating. The overall design of the system is similar to that of the Optical Scanner Measurement Technique as outlined in Fig. 2.13. However, this technique uses a cat's eye reflector as the target optic instead of the air-path type retroreflector. A tracking head geometric modelling is included to account for the geometrically imperfect tracking head. Calibration of this model is essential in order to attain the highest possible accuracy. The prototype is claimed to have repeatability better than 10 μm and accuracy of ±0.1 mm. The measuring volume is $1m^3$. It is also claimed that this set-up also allows automatic recovery when the tracking signal is lost [16].

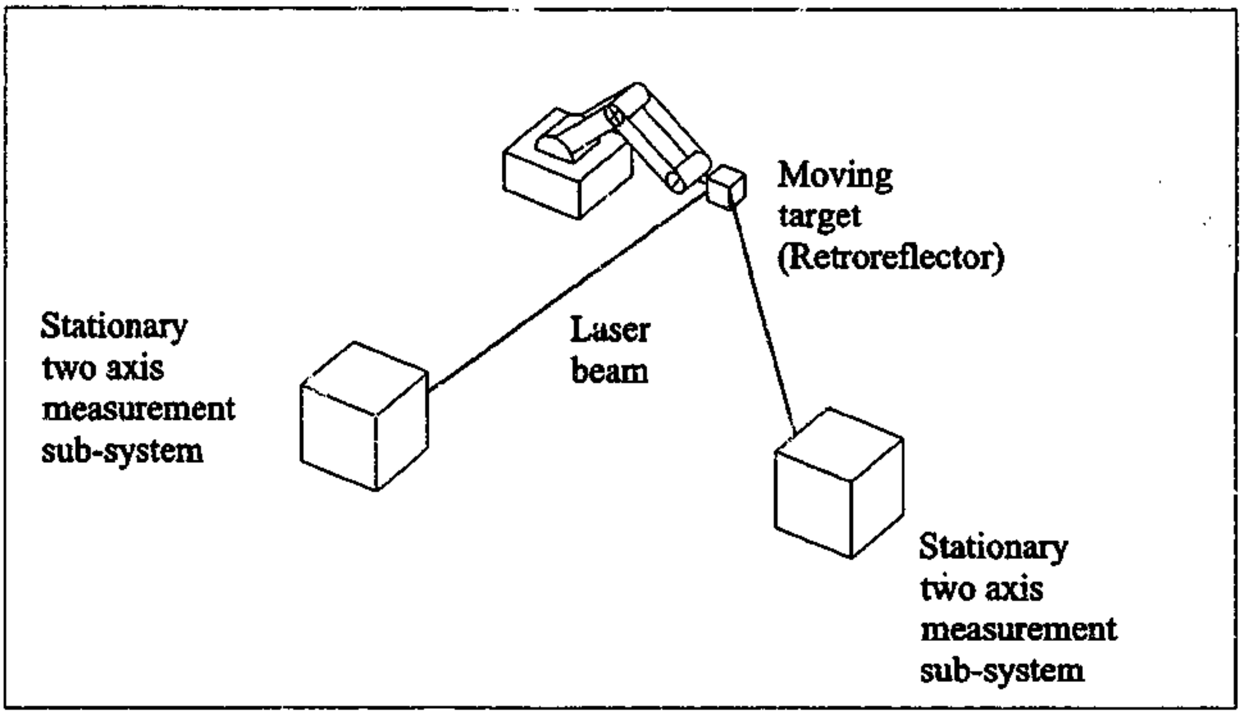


Figure 2.12: Schematic diagram of Optical Scanner Measurement Technique

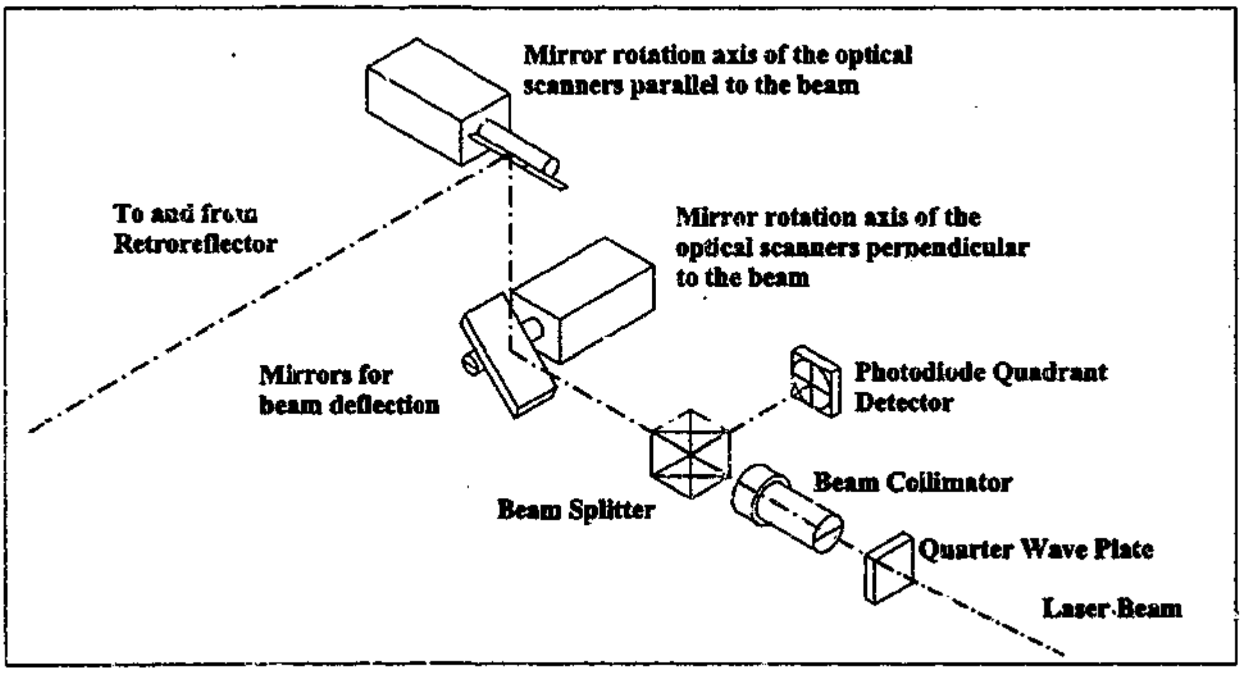

Mirror rotation axis of the optical scanners parallel to the beam
To and from Retroreflector
Mirror rotation axis of the optical scanners perpendicular to the beam
Mirrors for beam deflection
Photodiode Quadrant Detector
Beam Collimator
Beam Splitter
Quarter Wave Plate
Laser Beam

Figure 2.13: Measurement sub-system layout for Optical Scanner Measurement Technique

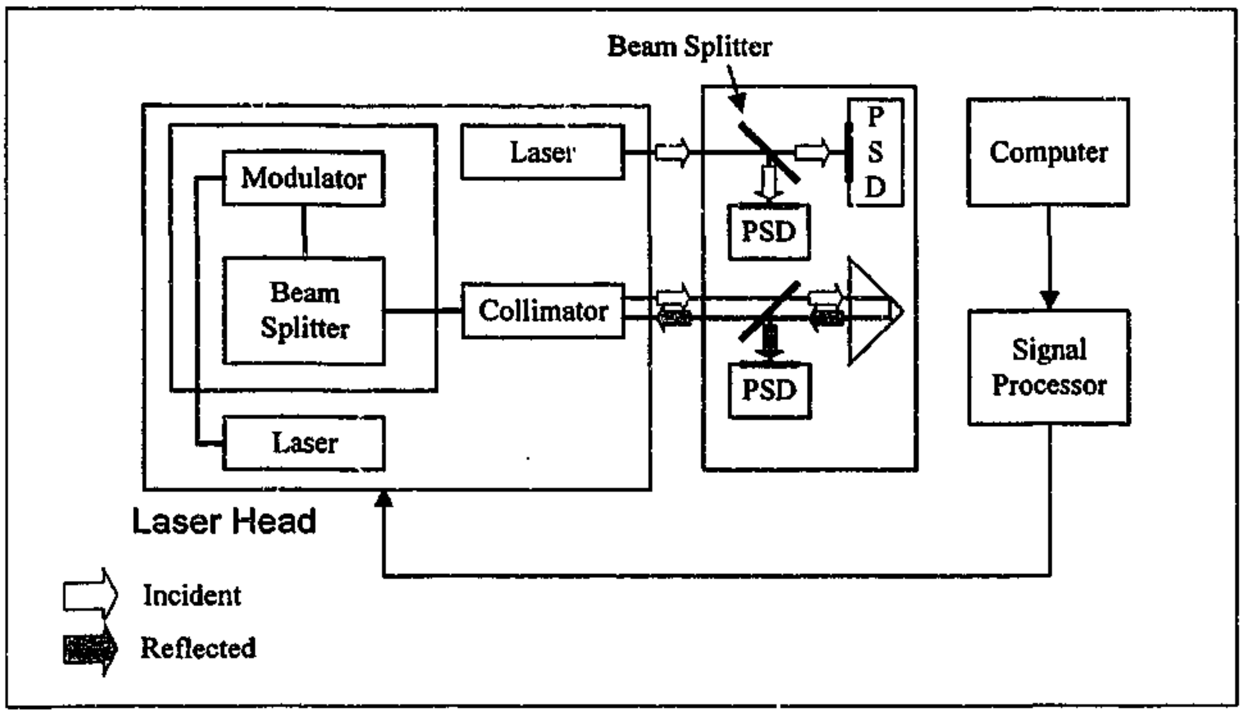


Figure 2.14 - Robotest measurement technique

The Robotest Measurement technique was developed through close collaboration between Polytec GmbH and the Institute of Tooling Machinery and Industrial Technology of the University of Karlsruhe in Germany [17]. This technique can measure motion along a linear path, recording the deviations of the robot and its displacement, velocity and acceleration. Fig. 2.14 shows the layout of the apparatus. This technique comprises a stationary laser head and a measurement head mounted on the robot's end effector. The laser head contains two lasers and an interferometer while the measurement head contains a retroreflector and three position sensitive diodes of the lateral effect detector type. This technique is used to measure the robot's linear displacement of up to ±3 μm accuracy for velocities up to 10 m/s within a cylindrical measuring volume of 10m in length and 20 mm in radius. The apparatus can sample data at up to 5 kHz [17].

A two-dimensional version of the Optical Scanner Measurement Technique was reported by Heeren and Veldpaus [18]. Fig. 2.15 provides a schematic representation of the technique. The laser beam from each measurement unit is directed towards the retroreflector pair with a flat mirror controlled by an actuator. The actuator was designed to reduce Coulomb friction to achieve high resolution and repeatability. The optical encoder consists of a code wheel with 3 encoder modules built around it to achieve resolution of 10^{-5} radian. The tracking offset of the retro-reflected beam is detected by a 2-segment PSD, which measures the beam horizontal deviation from the centre. Plano-cylindrical lenses are

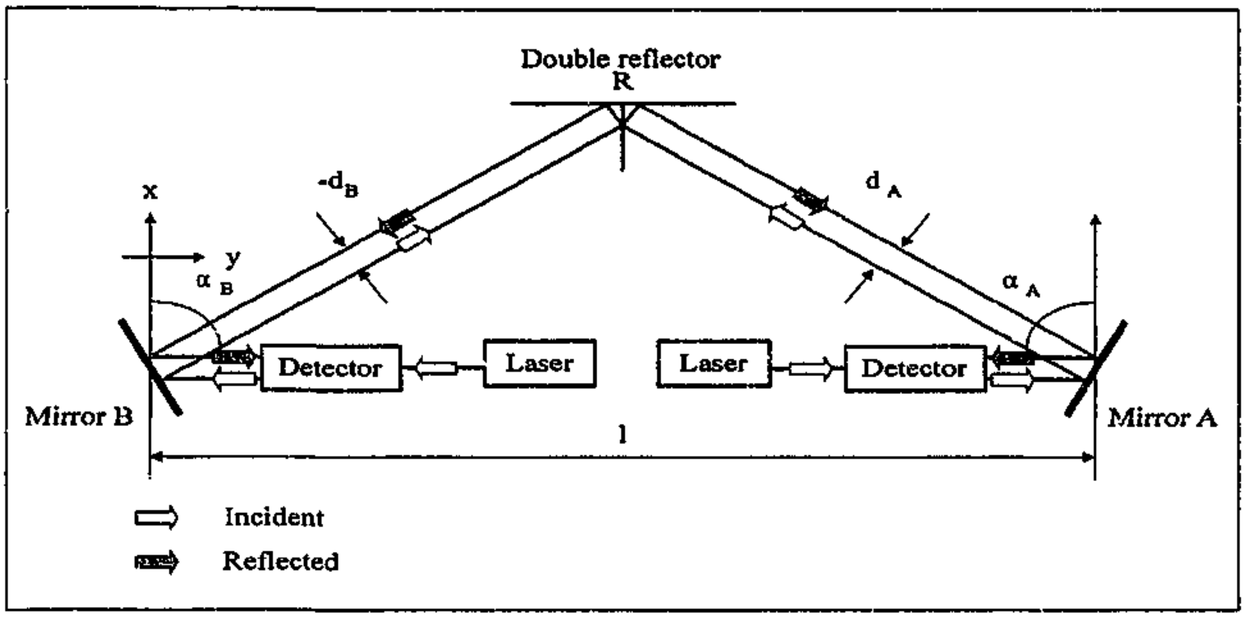


Figure 2.15 - The two dimensional optical measurement technique with double reflector

mounted in front of the PSD to ensure that the beam spot detection is insensitive to small translation of the retroreflector in the vertical (z) direction. The position of the double reflector mounted on the target can be found by triangulation. The measurement of the robot's end effector position is restricted to a horizontal plane of approximately 1 m^2. The resolution and repeatability of the system is reported as 0.05 mm with a maximum tracking speed of 5 m/s [18].

2.6.2 Single Laser Interferometry-Based Tracking and Measurement Technique

Recent studies have focused on single beam laser interferometry-based measurement techniques. It was first proposed by Lau, et al. at the National Bureau of Standards [14]. It was reported to be able to measure the position of a moving target as well as the pitch and roll angle of the target. Fig. 2.16 shows the set-up of this technique. This technique comprises of a tracking unit and a target unit. The tracking unit includes a laser interferometer system and a dual-axes servo-controlled tracking mirror (cardan joint) to direct the laser beam towards the air-path type retroreflector. The target unit consists of a partial mirror (15% transmission) and 2 two-dimensional lateral effect detectors and this target is rotated by a dual-axes servo. Additional optics such as the retardation plates and polarising beamsplitters are included to provide for 2 beam paths. The first beam path directs the beam to PSD A, used for the measurement of tracking offset. Appropriate commands are generated for the servo- controlled tracking mirror to correct the offset. The second beam path directs the beam to PSD B, which produces a measurement of angular misalignment between the measuring beam and the surface of the mirror. Appropriate commands are generated for the dual-axes servo to rotate the target unit so that the target mirror stays perpendicular to the measuring beam. The accuracy of this technique is reported to be 0.002% of the measuring range for position and 1 arc second for pitch and yaw angles in a measuring volume of 2x2x2 m^3 [14].

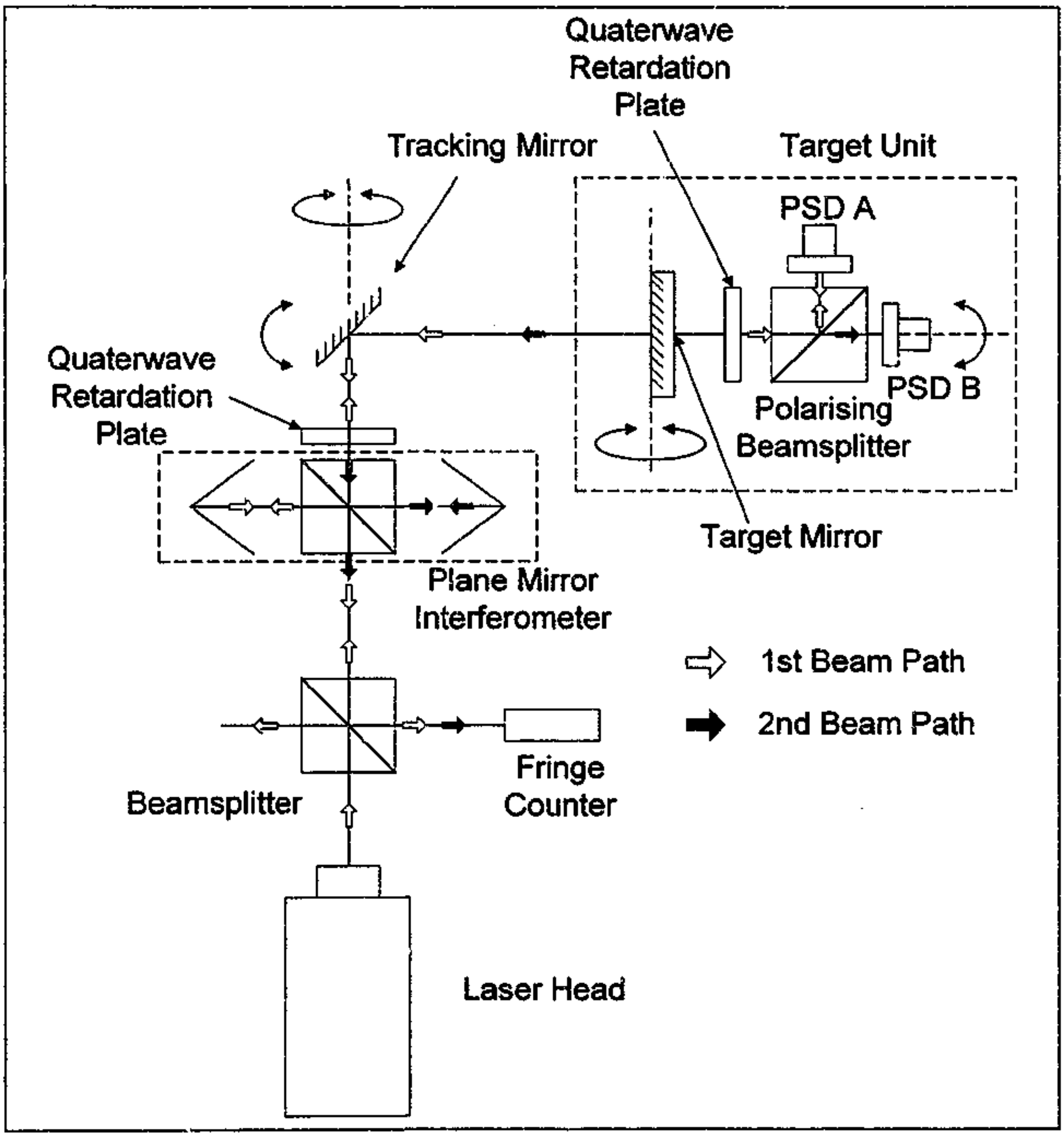


Figure 2.16 – Schematics of the first 5-axis single laser interferometry-based measurement technique

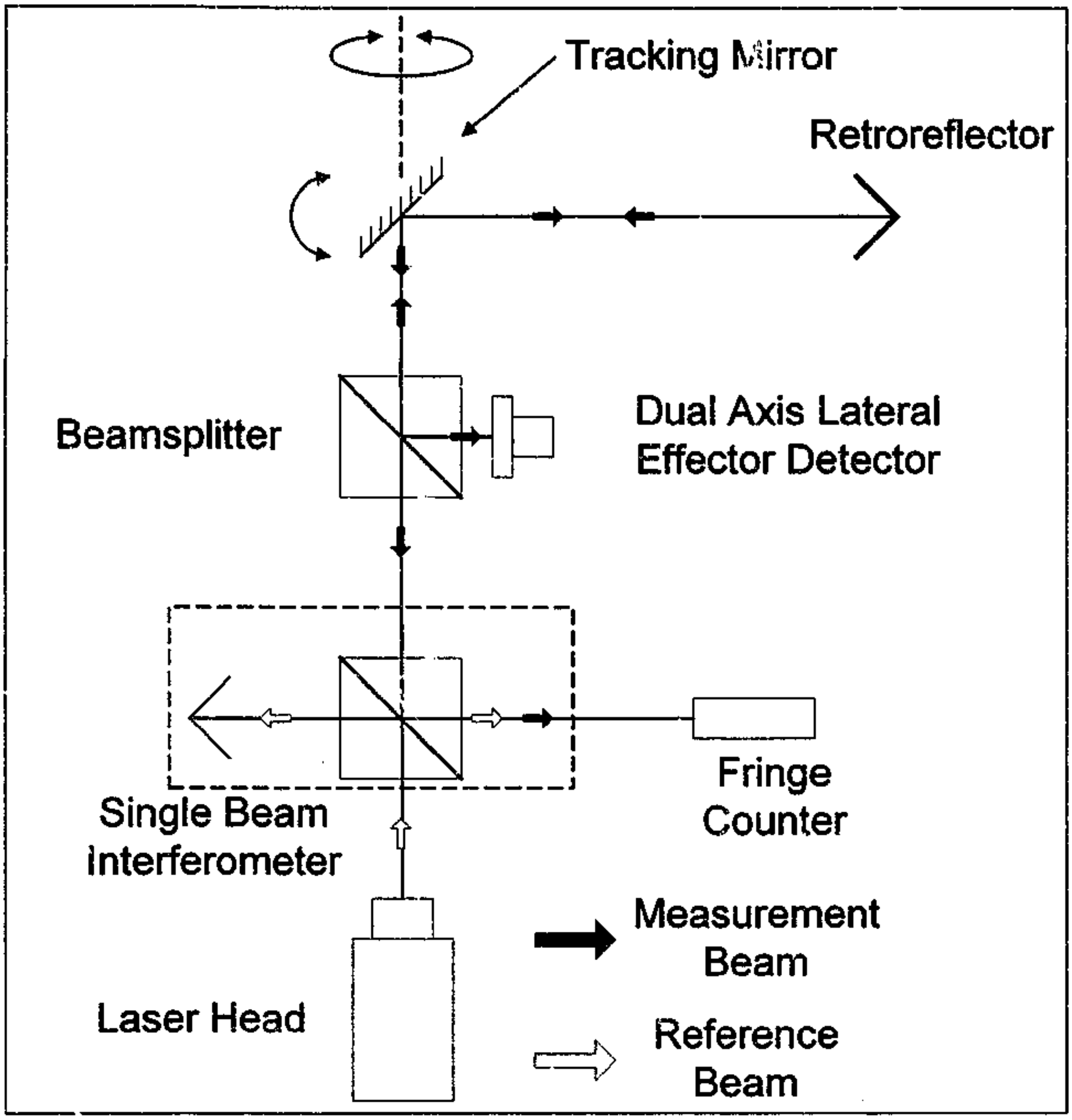


Figure 2.17 –3D single laser interferometry-based measurement technique

Lau et. al. also reported a technique for position measurements only. In this case, a cat's eye or an air-path type retroreflector is used instead of the servoed target. At the beamsplitter, part of the retro-reflected beam is directed onto a dual-axes lateral effect detector to determine the tracking offset. The remaining beam returned to the interferometer is picked up by the detector in the laser interferometer. This is used to determine beam displacement using the Doppler Shift principle. Tracking of the target retroreflector is achieved by rotating the tracking mirror by the corrective angles calculated using the tracking offset. Fig. 2.17 shows the set-up of this technique.

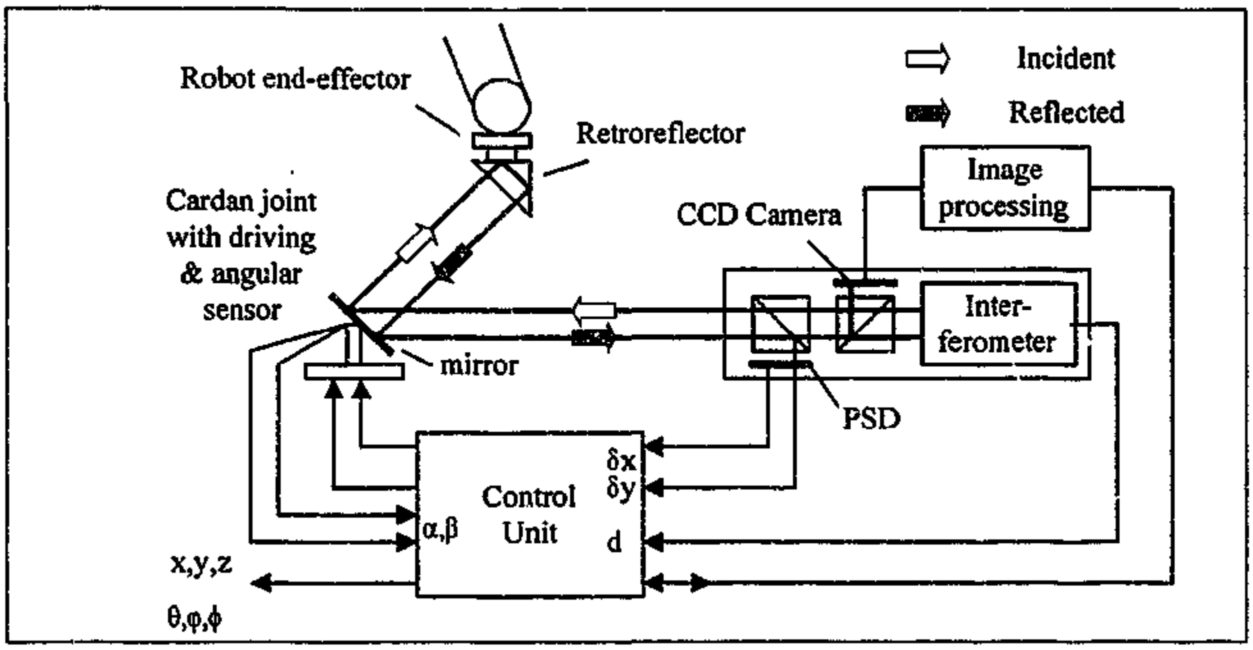


Figure 2.18: A functional diagram of the 6-axis sensor measurement technique reported by Vincze et.al.

Vincze et. al. [10, 11, 30, 31, 32] reported a technique which allows measurement of both robot's position and orientation of its end effector in real time. A functional diagram of the technique is provided in Fig. 2.18. A similar set-up to the technique proposed by Lau et. al. is used for the position measurement. To account for geometric errors of the cardan joint and the misalignment of the optical components, a description of the kinematic model of the apparatus is presented. With this model, the position of the target is calculated using the cardan joint angles, the displacement measured using the laser interferometer and the PSD signal. It is reported that the developed prototype system based on this approach has a position measurement accuracy of 0.2 mm and has a sampling rate of 10 kHz. It is capable of tracking robot velocities of up to 6 m/s and accelerations of up to 25 m/s^2. It is claimed that a predictive control algorithm and a floating point digital signal processor used in this approach improve the position measurement accuracy to 50 μm and robot acceleration of up to 100 m/s^2 [32].

Orientation measurement is achieved by capturing and analysing the image of the reflected laser beam [12]. This can be achieved by directing part of the retro-reflected beam to a Charged Coupled Device (CCD) camera using another beamsplitter. The image captured by the camera consists of darkened lines whose directions and angles are directly related to the orientation of the retroreflector. Analysis of this image enables the determination of the roll ρ, pitch ϕ, and yaw φ of the end effector. The accuracy of the orientation calculation depends upon the accuracy of the CCD camera and subsequent image processing [10]. For the technique described, it was reported that the overall orientation measurement accuracy was better than 10 arc seconds (0.0027°) for a working range of ±30° of pitch or yaw with an unrestricted roll angle. However, the sampling rate of normal CCD arrays is restricted to 50 Hz, which is not sufficient for real-time orientation measurement. A linear CCD array may be used to provide real-time measurement as these allow sampling rates of more than 1 kHz. The accuracy of the technique is limited to an extent by the effects of atmospheric conditions on the laser beam. However, the greatest loss of accuracy is caused by errors resulting from the determination of the shadow line projections on the CCD arrays. Less information about the retroreflector edges is collected, which reduces the accuracy of the resulting measurement. In addressing this, additional geometric features can be added to the retroreflector such as wire cross hairs across the front of the retroreflector [10, 11, 12, 31].

A similar method developed is the Luxwess measurement technique [22]. This measurement technique is based on the same conceptual design as the previous technique described. A number of changes and improvements have been made to enhance its performance. This approach employs three mirrors and a single prism instead of the previous two-axis cardan joint with a single plain mirror. This allows twice the accuracy of the single mirror approach [22]. Backlash is eliminated through the use of direct drive motors that are mounted directly to the axes without any coupling and air bearings are used to eliminate friction. Tumbling effects are measured on-line using miniaturised capacitive sensors, and miniaturised temperature sensors measure the temperature of the beam steering unit. Luxwess has a position measurement accuracy of 50 μm per meter distance. The system has a sampling rate of 10kHz that enables a maximum velocity and acceleration of object being measured of 6m/s and 10m/s^2 respectively [22].

Another method of laser-based orientation measurement technique for a mobile robot has been proposed by Bao et. al. [33]. This technique does not require image processing. This orientation measurement technique is achieved by incorporating two PSDs and a beamsplitter located on an optical set-up along with the retroreflector. This optical set-up is then positioned on a two-axis cardan joint, which is then mounted on the mobile robot to be tracked. This approach is reported to offer real time orientation measurement with high precision. It is claimed that the precision of this orientation measurement is in the order of 0.0002 radians (0.011°) provided that there are no construction errors in building the system [33]. However, the claimed accuracy was never substantiated by experiments.

2.6.3 Multiple-laser Tracking Robot-Calibration Technique

A technique that utilises four laser-interferometer sub-systems, each of which is mounted on its own two-axis tracking mechanism, has also been proposed [19]. This technique makes use of a specially designed cat's eye retroreflector target and spherical seat bearings to provide a tracking accuracy below 1 μm in a cubic space with sides of a few metres when air turbulence is neglected. Fig. 2.19 provides a schematic diagram of the proposed technique.

This technique is based on the principle of multiple-laser trilateration (i.e. a co-ordinate measuring method using only length data by laser interferometry). It is unique in that unlike other techniques that employ mirror rotation for target tracking, this technique employs interferometer rotation. A small single-beam Michelson interferometer is attached on each tracking mechanism. The interferometers are attached to the laser head via polarisation-maintaining optical fibres. Each interferometer can provide a measure of the distance from the tracking mechanism to the cat's eye target and can accept movement of the target up to 0.6 m/s. The cat's eye retroreflector used was specially developed and was reported to give an optical path error of less than 1 μm along two different circumferences between ± 80° (the best commercially available cat's eyes generally have an error of about 3 μm). Thus, the use of this particular cat's eye ensures high accuracy for a maximum range of incident angles [19]. The tracking error for each mechanism is determined in the conventional way using a beamsplitter to divert the return beam from the target to a quadrant detector. Fig. 2.20 shows a diagram of the tracking mechanism.

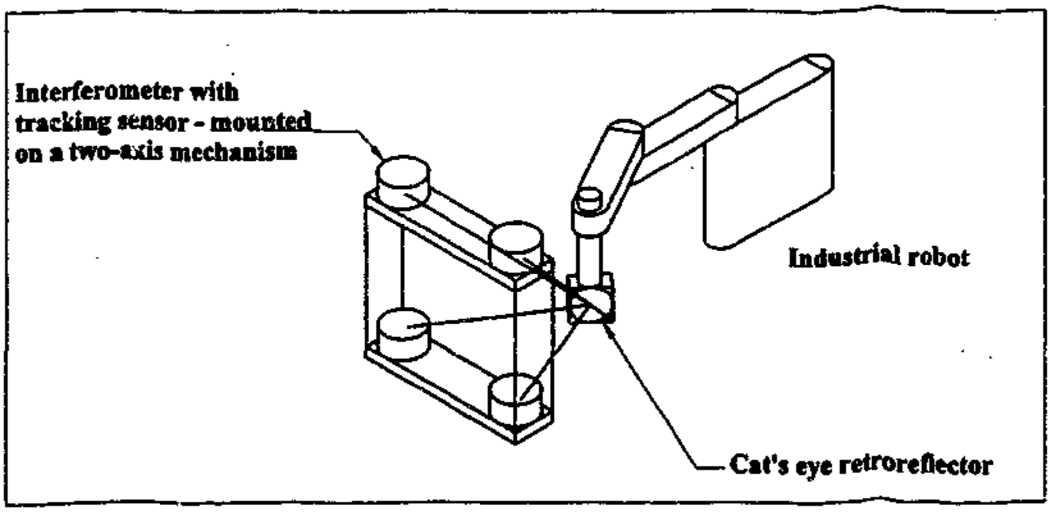

Interferometer with tracking sensor - mounted on a two-axis mechanism
Industrial robot
Cat's eye retroreflector

Figure 2.19: The multiple laser tracking robot-calibration technique

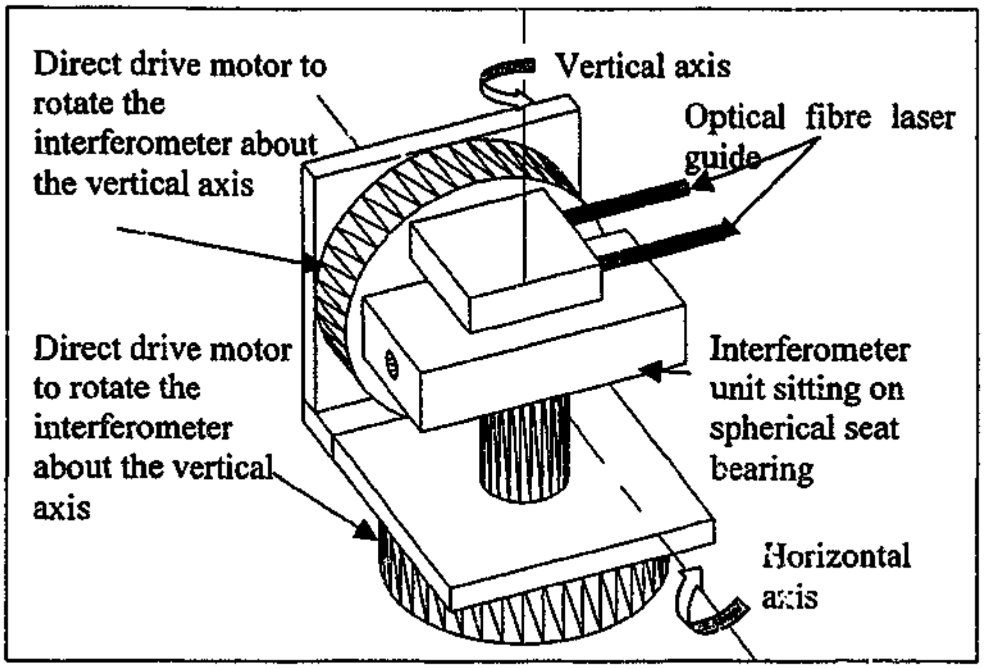

Direct drive motor to rotate the interferometer about the vertical axis
Vertical axis
Optical fibre laser guide
Direct drive motor to rotate the interferometer about the vertical axis
Interferometer unit sitting on spherical seat bearing
Horizontal axis

Figure 2.20: Side view of 3D tracking mechanism with spherical seat bearing

2.7 Summary

The background of components to establish a laser interferometry-based sensing and measuring (LISM) apparatus has been presented. The previously published work related to this study, together with the various techniques and control algorithms implemented have also been reviewed. The next chapter will focus on the establishment of the experimental LISM apparatus and the experimental verifications of the performance of the apparatus.

Chapter 3

Experimental LISM Apparatus: Development and Characterisation of Sub-systems

3.1 Introduction

This chapter provides a description of the principle of the Laser Interferometry-based Sensing and Measuring (LISM) technique and the sub-systems constituting the experimental LISM apparatus. This apparatus will be based on a single laser interferometry-based technique utilising a PSD to detect tracking offset and a laser interferometer to determine the target displacement. The sub-systems can be divided into the following 4 main functional groups:

1) Laser Interferometer sub-system;
2) PSD sub-system;
3) Beam steering sub-system;
4) Orientation measurement sub-system;

The physical make-up, specification, functionality and the measurement and analysis techniques employed for each sub-system are the central of focus of this chapter. Orientation measurement sub-system is presented in the next chapter.

3.2 Principle of Laser Interferometry-based Sensing and Measuring for Position Measurement

Laser interferometry-based sensing and measuring of robot motion generally involves the dynamic acquisition of the positions of a robot end-effector in its workspace. It can also be used to provide orientation measurements of the robot's end-effector. The LISM technique uses the linear and angular displacement data, obtained from the interferometer and beam steering mechanism, respectively, to provide the position of the target retroreflector mounted on the end-effector of a robot manipulator. It maintains tracking of the target retroreflector by sensing the offset of the incident and reflected laser beam from the target. It subsequently performs offset corrections by adjusting the angles of the beam steering mechanism. The functional layout of the LISM technique developed in this study is shown in Fig. 3.1. A photo of the apparatus is shown in Fig. 3.2.

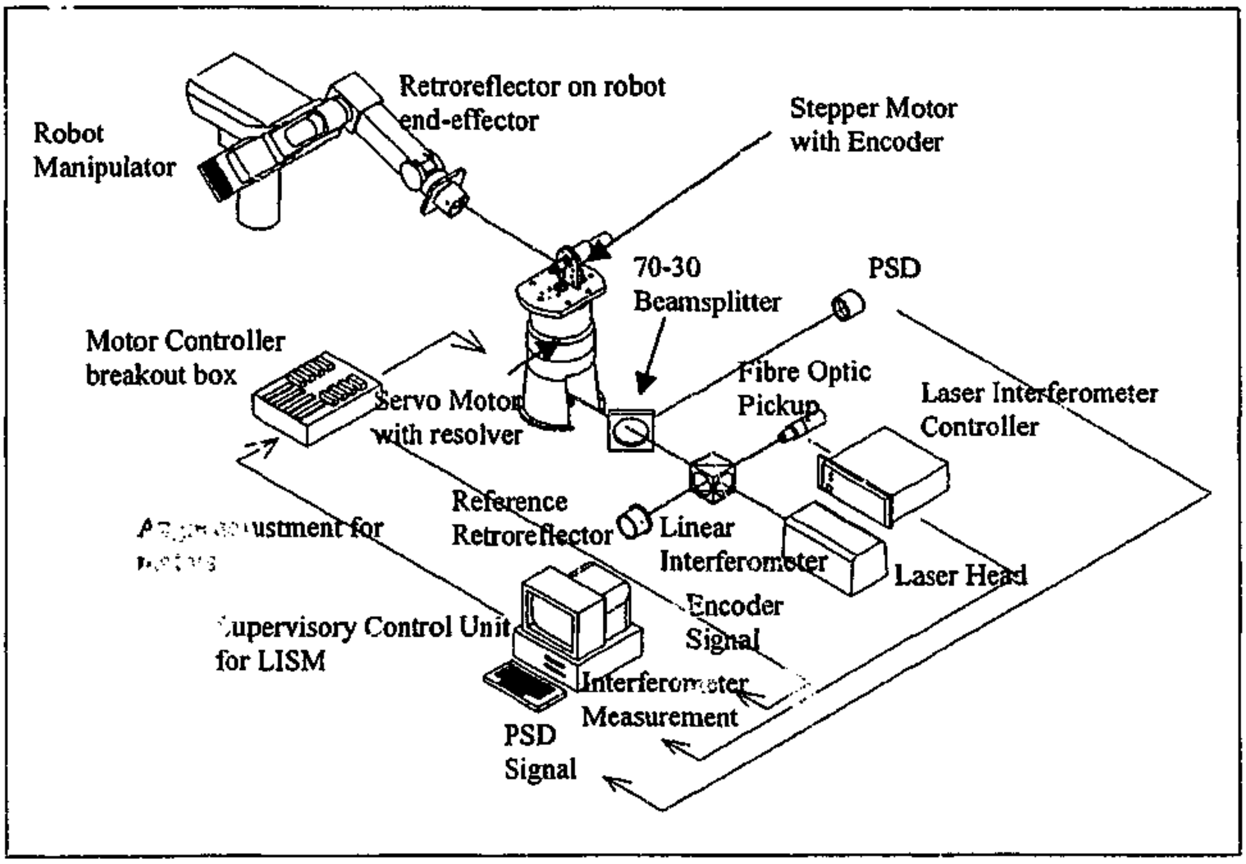

Robot Manipulator
Retroreflector on robot end-effector
Stepper Motor with Encoder
70-30 Beamsplitter
PSD
Motor Controller breakout box
Servo Motor with resolver
Fibre Optic Pickup
Laser Interferometer Controller
Reference Retroreflector
Angle adjustment for motors
Linear Interferometer
Laser Head
Encoder Signal
Supervisory Control Unit for LISM
Interferometer Measurement
PSD Signal

Figure 3.1: Functional layout of LISM technique

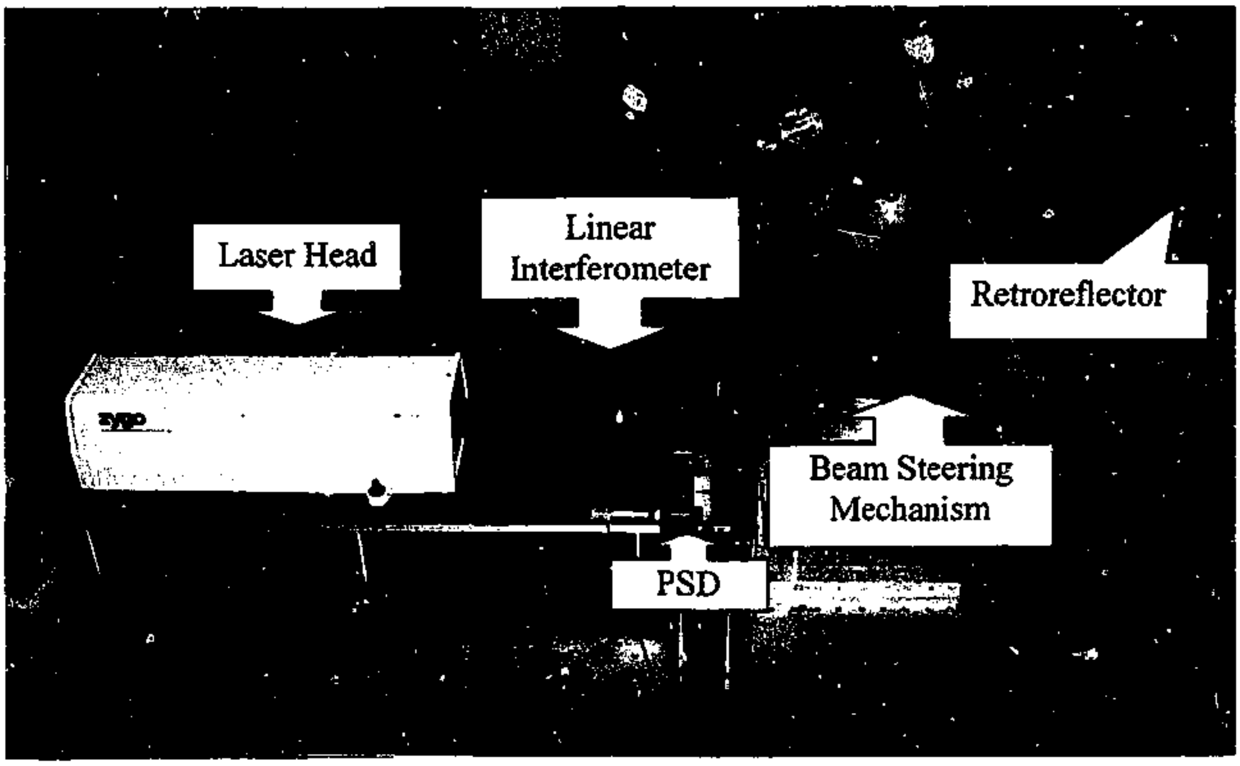


Figure 3.2: Photo of LISM apparatus

As shown in the layout, a heterodyne laser beam is emitted from the laser head. The laser beam passes through a polarisation beamsplitter in the interferometer, where it is split into a reference beam and a measurement beam. The reference beam is directed to the reference retroreflector where it is reflected and directed to the fibre optic pick-up. It is then transmitted to the measurement board on the laser interferometer controller. The measurement beam passes through a 70-30 beamsplitter before being directed to a retroreflector by the beam steering mechanism. The retroreflector is generally mounted on the robot's end-effector. At the retroreflector, the incident measurement beam is reflected through 180° and the reflected measurement beam travels back parallel to the incident beam. If the incident measurement beam does not hit the centre of the retroreflector, there will be an offset between the incident and the reflected measurement beams.

This reflected measurement beam follows the same path back to the 70-30 beamsplitter. 70% of this beam power is directed to the interferometer before being deflected by the polarisation beamsplitter in the interferometer and picked up by the fibre optic cable. As the end-effector carrying the retroreflector is moved, there is a shift in frequency for the measurement beam based on the Doppler Principle [24]. The measurement and reference beams are combined in the laser interferometer controller to form the interference signal, which provides the rate of change of target displacement. This rate is then integrated to provide the relative displacement of the target. The remaining 30% of the reflected beam power is directed by the 70-30 beamsplitter to a position sensing detector (PSD). This PSD determines the offset between the incident and the reflected laser beams. This offset is referred to as the tracking offset.

The LISM control system minimises the tracking offset obtained from the PSD acquisition system by signalling the motor controller which in turn rotates the axes of the beam steering mechanism, thus following the arbitrary movements of the target. Measurement of the position of the target in space is obtained from the displacement of the beam, tracking offsets, and angular displacements of the axes of the beam steering mechanism. The above information is utilised within the kinematics equations of the LISM apparatus describing the Cartesian position of the target. A predictive control algorithm that allows estimation of the future position of the target from the previous position, velocity and acceleration values, [36], can be employed to increase the speed of tracking.

3.3 Laser Interferometer sub-system

3.3.1 Specifications

The laser interferometer measurement sub-system is the Zygo ZMI-1000 interferometer [25, 37]. The diameter of the beam is 6 mm with a minimum and maximum power of 425 μW and 1000 μW, respectively. The ZMI-1000 is capable of measuring the relative displacement of a retroreflector moving up to 1.1 m/s in the direction of the beam with a linear resolution of 2.47 nm. Fig. 3.3 shows the photo of the ZMI-1000.

It must be noted that the laser interferometer only measures relative position change between the interferometer and the target retroreflector. It does not measure absolute position. Several factors affect the accuracy of the measurements made. These factors include the environmental conditions (e.g. temperature, pressure), optical alignments and electronic noise. These factors and their effects must be taken into consideration to evaluate the performance of the overall LISM apparatus. This will be discussed in detail in Chapter 5.

3.3.2 Measurement and Reference Beam Offset

The measurement and reference beams should have zero offset at the fibre optic pick-up for the entire range of displacement. The interference signal is produced only from the overlapped area of the measurement and reference beams. A beam offset between the measurement and reference beam will cause a reduction in the interference signal. The beam offset can arise from two types of misalignment:

1) Lateral misalignment due to the movement of target reflector or the interferometer.
2) Angular misalignment due to the angular offset between the measurement beam and the axis of displacement being measured.

Lateral misalignment will produce a beam offset (equivalent to tracking offset) corresponding to twice the lateral offset, which does not vary with beam displacement. Fig. 3.4 shows a beam offset due to lateral misalignment.

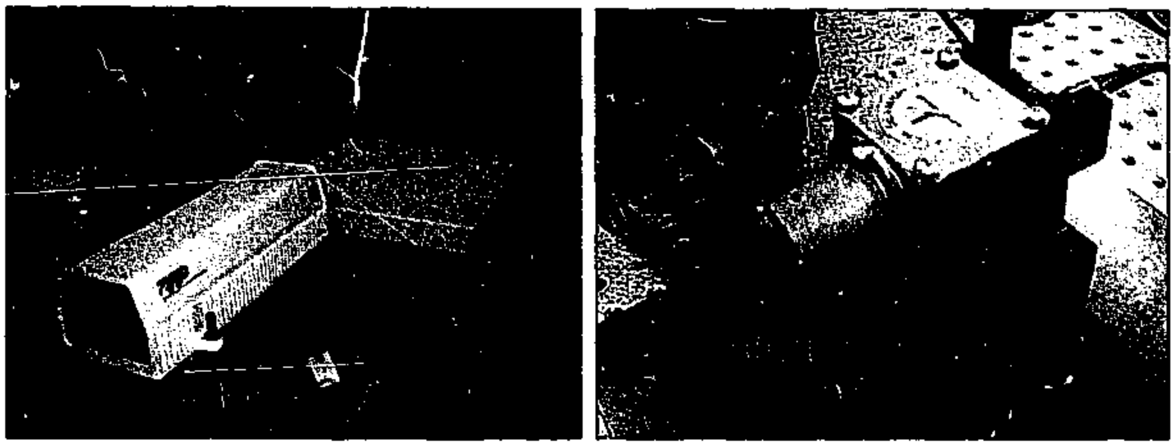


Figure 3.3: Photo of the ZMI-1000 laser interferometer measurement sub-system

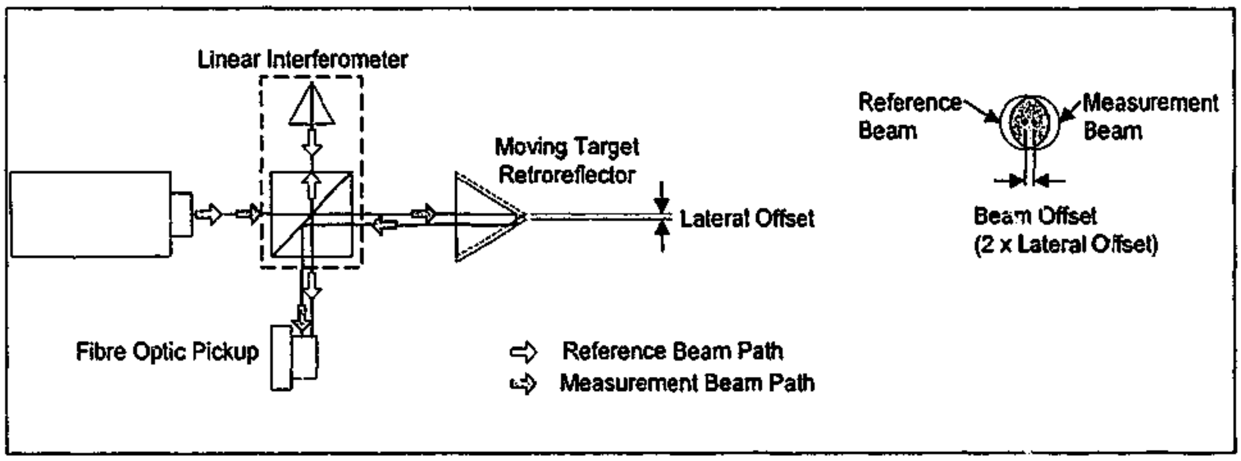


Figure 3.4: Beam offset due to lateral misalignment

During the normal operation of the LISM apparatus, the movement of the target retroreflector along the plane perpendicular to the laser beam will cause a lateral offset and thus a beam offset as described above. In order to maintain measurement, the maximum permissible beam offset must be determined. The maximum permissible beam offset was found to be ±3 mm. The maximum lateral offset that the retroreflector can accommodate is thus ±1.5 mm, above which the acquired measurement will not be reliable. In order to follow the retroreflector, the supervisory control unit must be able to detect this offset and to reduce this offset before it reaches the maximum.

For angular misalignment, the beam offset will vary by an amount directly proportional to the product of the misalignment angle (in radians) and the beam displacement. Fig. 3.5 shows the effect of beam offset due to angular misalignment.

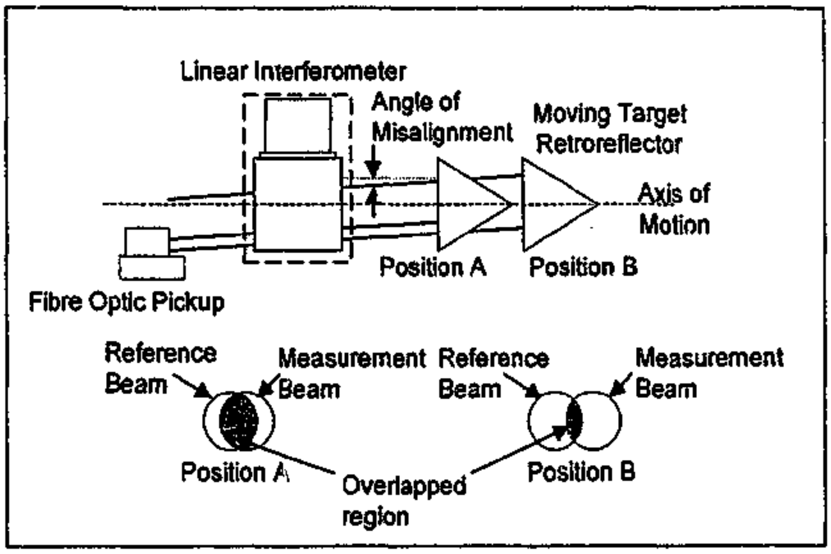


Figure 3.5: Beam offset due to angular misalignment

3.3.3 Sub-system Interface

The ZMI-1000 controller synchronises the measurements in the installed measurement board as well as relaying this information through the RS232 port or the GPIB (IEEE 488 General Purpose Interface Bus) port [25] to a host PC. In the current set-up, only the RS232 interface is used. The ZMI-1000 controller can sample up to 130 KHz. However, the update rate on a host PC relies heavily on the speed of transmission of user commands and sampled data. The position register is being read continuously, and a 32 bit data is returned at a maximum rate of 200Hz with RS232. This slower rate is due to the fact that the PC's processor is slow in processing the user commands and the sampled data. Moreover, due to the multitasking nature of the Windows operating system, other tasks may require the processor's attention. A faster processor would improve the sampling rate.

3.3.4 Instability in Laser Interferometry

During the measuring operation, there must not be any laser beam feedback into the laser head. The polarising beamsplitter and the quaterwave retardation plate are used to prevent this [25, 37]. However, due to the imperfect nature of these components, a small amount of reflected measurement beam is fed back into the laser head. The occurrence of this causes the emitted laser to become unstable, resulting in errors in the measurements recorded [25]. To prevent this, the set-up shown in Fig. 3.6 should be used. Instead of directing the laser beam through the centre of the polarising beamsplitter, it is offset by a fixed amount. The incident beam will not hit the centre of the target retroreflector and thus the reflected beam

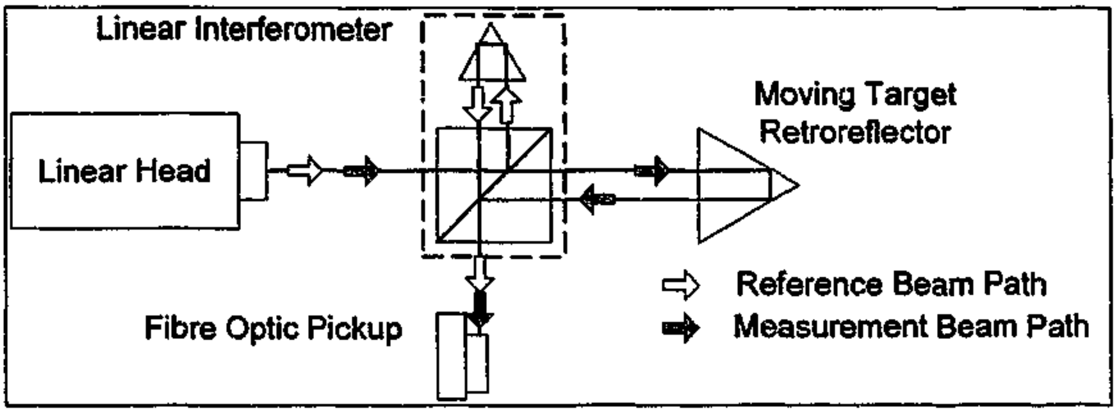


Figure 3.6: Alternate laser interferometry set-up

will also be offset from the centre by a corresponding amount. The fibre-optic pickup assembly is mounted as shown so that the reference and the returned measurement beam completely overlap each other. There will be no feedback of the beam to the laser head and thus improve laser stability.

3.4 PSD sub-system

3.4.1 Specifications

The PSD sub-system is responsible to provide the supervisory control unit with the tracking offset of the target. The sub-system consists of a position sensing detector (PSD), a signal conditioning device, and a data acquisition board. Fig. 3.7 shows the photo of this sub-system.

The detector used in this sub-system is a Lateral Effect Detector [28]. This detector has a position measuring range of ±9.4 mm along the two axes. The responsivity of the detector for the Zygo laser beam with wavelength of 632 nm is 0.42 A/W. This value determines the amount of photocurrent produced by the incident beam power. The dark current of the detector ranges from 12 nA to 250 nA. The dark current contributes to the noise in the measurement acquired by the detector. Rise time for this detector is 5 μs, which corresponds to a maximum update rate of 200 KHz.

The photocurrent signals generated by the detector must be conditioned and optimised for the input type and input range of the data acquisition board. This device is used to provide optimisation in the following areas:

1) Amplification- Photocurrent signals generated by the PSD are first converted to voltage signals. The amplitude of these voltage signals are generally very low level and should be amplified to increase resolution and reduce noise. Amplification and conversion is achieved by the use of an operational amplifier. Amplification factor is determined by the size of the resistor used, which in this case is 2.2 MΩ [26, 27, 28].
2) Filtering- The purpose of filters is to remove unwanted noise from the signals to be measured. Noise filters such as low pass filters are used to eliminate high frequency noise before the data acquisition board samples the signals. This is achieved by the addition of a capacitor on the operational amplifier [28, 38]. The resistance and the capacitance determine the cut off frequency, which in this case is 1000 Hz.

The data acquisition (DAQ) board samples and converts conditioned analogue voltage signals into digital code representations for computer processing and storage through analogue-to-digital (A/D) conversion. The DAQ board used in this application is a Data Translation DT303 [39]. Maximum sampling rate for this board is 440 kSamples/s. Several input channels can be sampled using multiplexing. However, the input circuit shares a common A/D converter and thus the number of input channels in use usually affects the sampling rate. The smallest detectable voltage change is 0.024% of the selected input range.

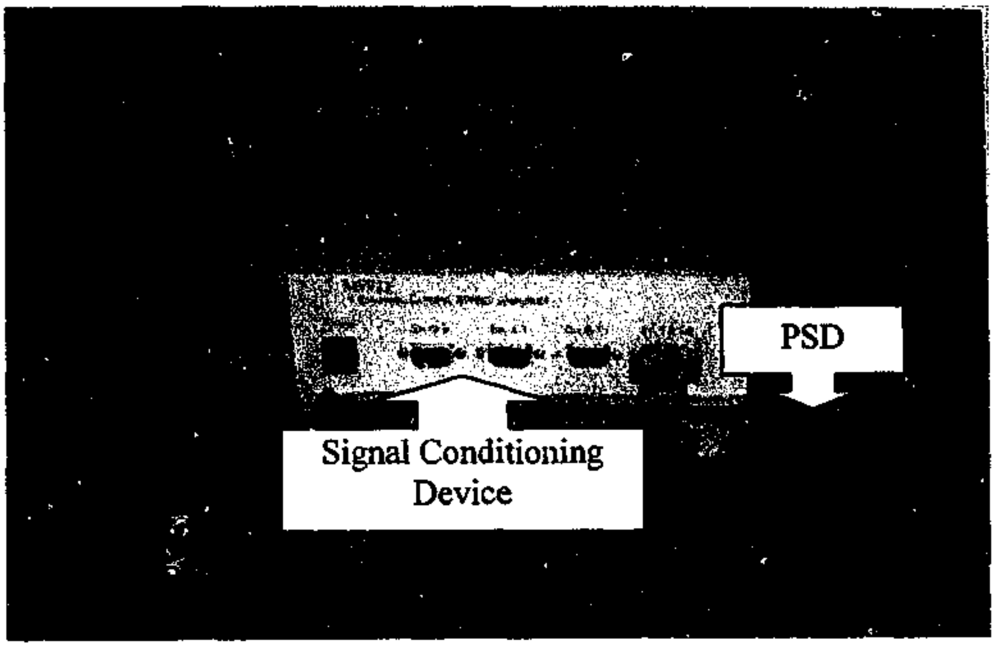


Figure 3.7: Photo of the PSD sub-system with the signal conditioning device

3.4.2 PSD Calibration

By using a beam diameter of 6 mm, the effective position measuring range of the detector is, as shown in Fig. 3.8, ±6.4 mm. This is because the laser spot has to fall on the sensing area at all times for valid measurement. Since the DAQ board is sampling voltage, calibration has to be carried out to obtain a relationship between the voltage sampled and the relative position of the laser beam. The PSD is first mounted onto a rotation stage and then onto a 2-axes translation stage as shown in Fig. 3.9. The translation stage is used to move the PSD along each of the 2 orthogonal axes independently and the rotation stage is used to ensure that the translations are along the principle axes of the PSD. During the calibration process, a laser beam is directed onto the centre of the PSD and the PSD is translated along the x- and y-axes independently at very small increments. The corresponding voltages from all 4 channels are recorded. The voltage ratio at each position is calculated as follows [28]:

$$\text{VRatioX} = \frac{V_3 - V_1}{V_3 + V_1} \tag{3.1}$$

$$\text{VRatioY} = \frac{V_2 - V_4}{V_2 + V_4} \tag{3.2}$$

where VRatioX and VRatioY are voltage ratio along PSD's x- and y-axes, respectively and V_c (c = 1,2,3 or 4) are voltages from corresponding channel number.

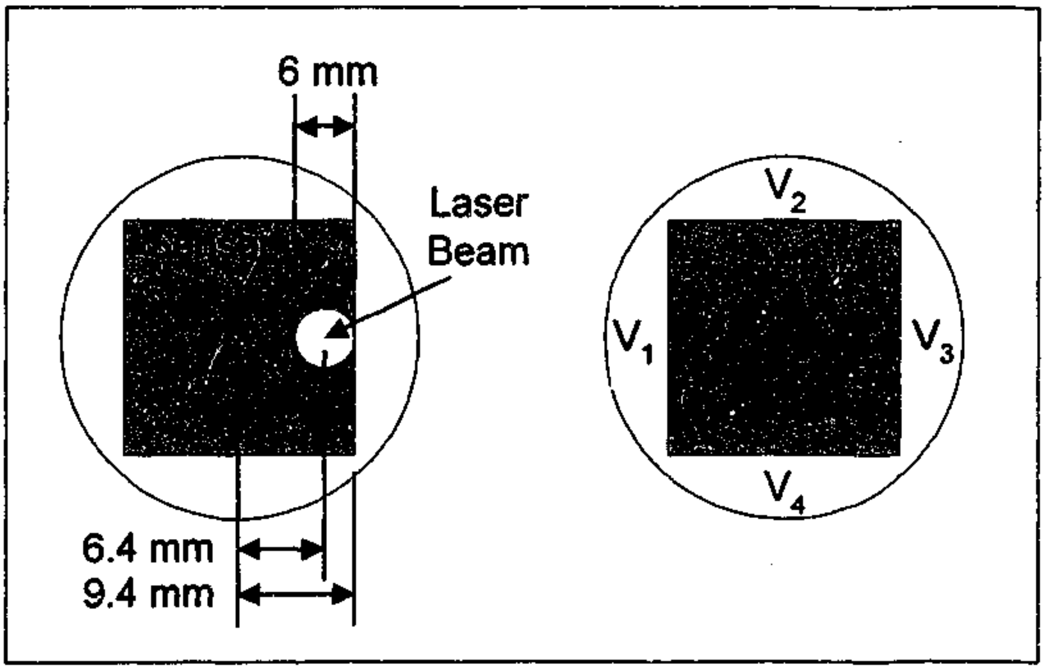


Figure 3.8: PSD effective range

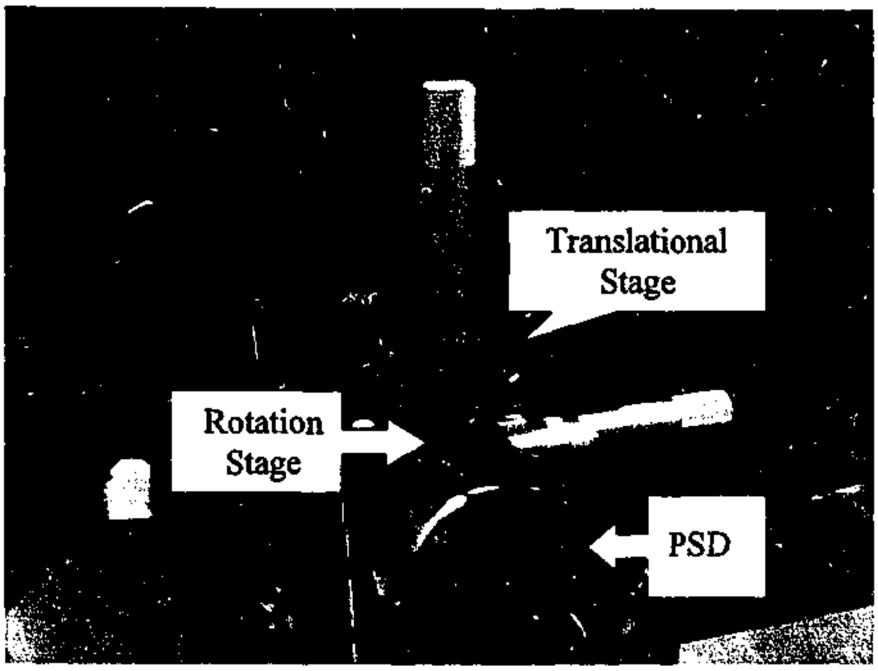


Figure 3.9: PSD calibration set-up

The voltage ratio is plotted against the position to obtain the relationship between the voltage ratio and the beam position. This must be performed carefully to ensure accurate measurements. Fig. 3.10 shows an example of the calibration graph for the full effective measuring range of the detector. The horizontal axis of the graph represents the voltage ratio whereas the vertical axis represents the actual beam displacement along the PSD x- and y-axes. Fig. 3.10 shows that this detector exhibits significant non-linearity when the beam is positioned more than 4 mm from the centre of the PSD. Therefore the range of ±3 mm is selected. This corresponds to a retroreflector's lateral offset of ±1.5 mm, which matched the tolerable offset for the Laser Interferometer sub-system.

Due to the non-linearity of the PSD, 2nd order polynomial are used to describe the relationship between the voltage ratio and the beam positions. The calibration graph is presented in Fig. 3.11 along with the equations describing the relationship as follows:

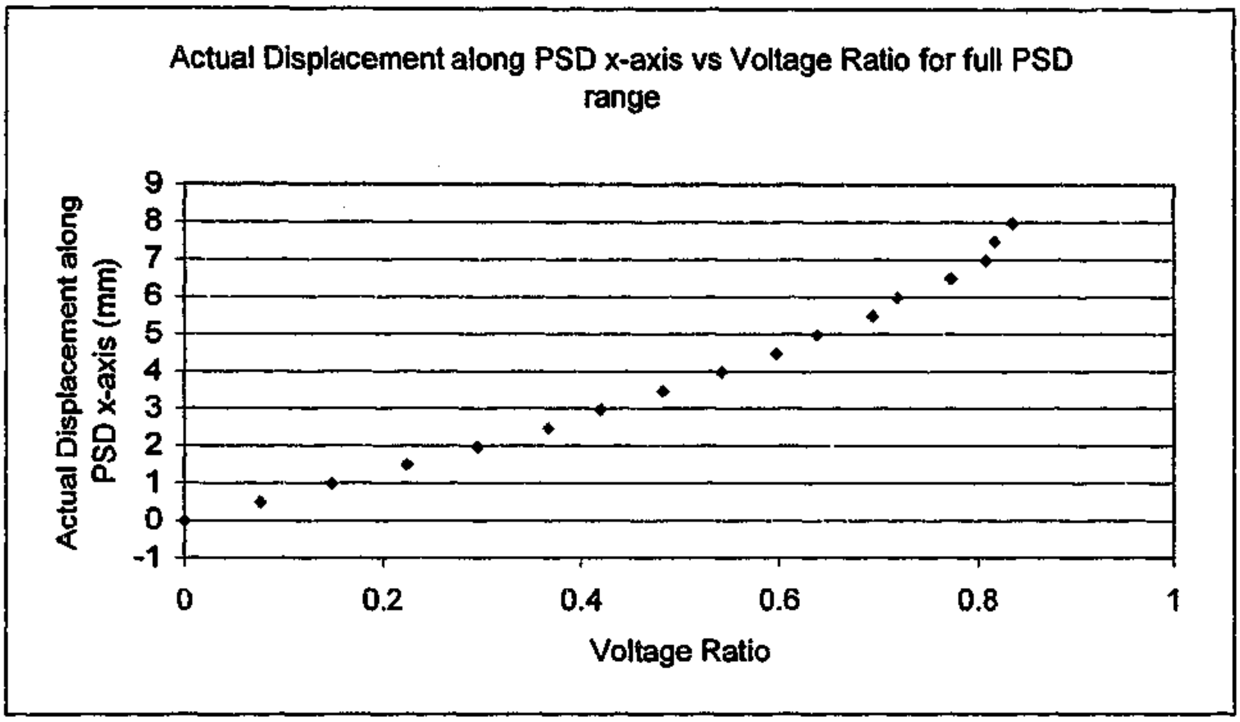
$$\text{PSD-x} = 1.519\times\text{VRatioX}^2 + 7.472\times\text{VRatioX} \quad \text{for VRatioX} \geq 0.0 \qquad (3.3)$$

$$\text{PSD-x} = -1.283\times\text{VRatioX}^2 + 7.391\times\text{VRatioX} \quad \text{for VRatioX} < 0.0 \qquad (3.4)$$

$$\text{PSD-y} = 1.769\times\text{VRatioY}^2 + 7.344\times\text{VRatioY} \quad \text{for VRatioY} \geq 0.0 \qquad (3.5)$$

$$\text{PSD-y} = -1.351\times\text{VRatioY}^2 + 7.834\times\text{VRatioY} \quad \text{for VRatioY} < 0.0 \qquad (3.6)$$

where VRatioX and VRatioY are voltage ratio along PSD's x- and y-axes, respectively, PSD-x and PSD-y are calculated beam positions along PSD's x- and y-axes, respectively.



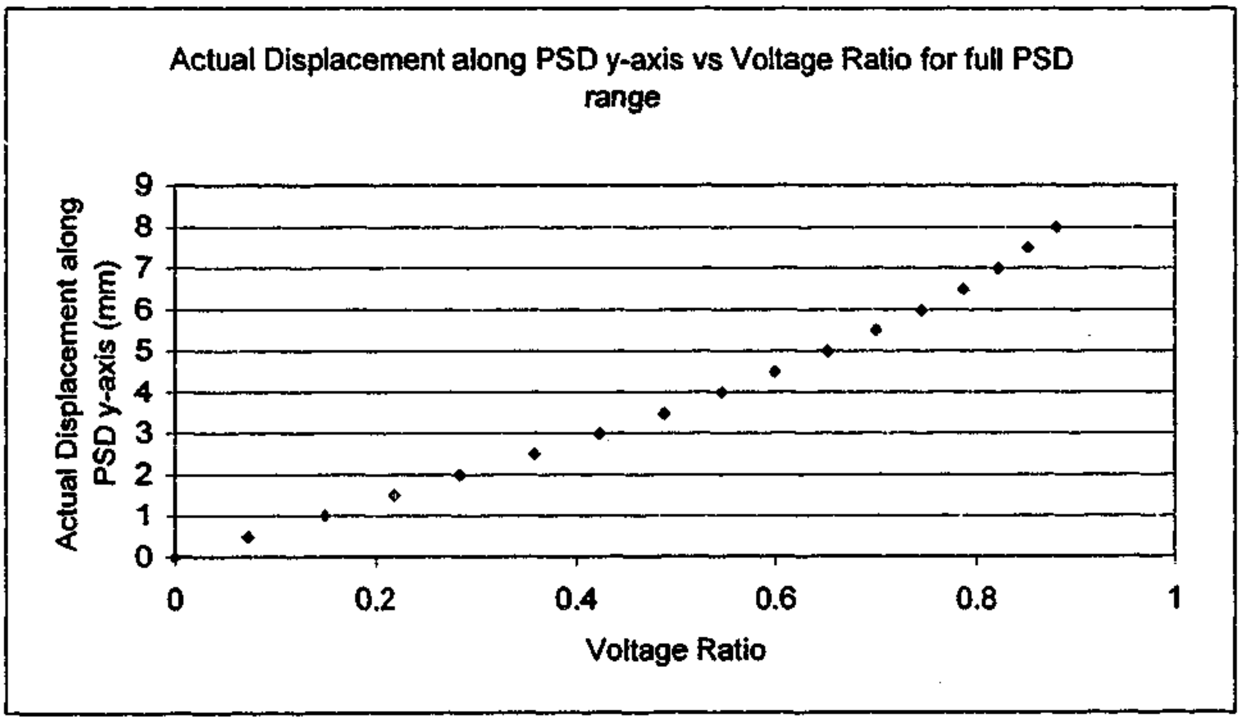


Figure 3.10: Calibration graph for full positive range of the detector along PSD x and y-axes

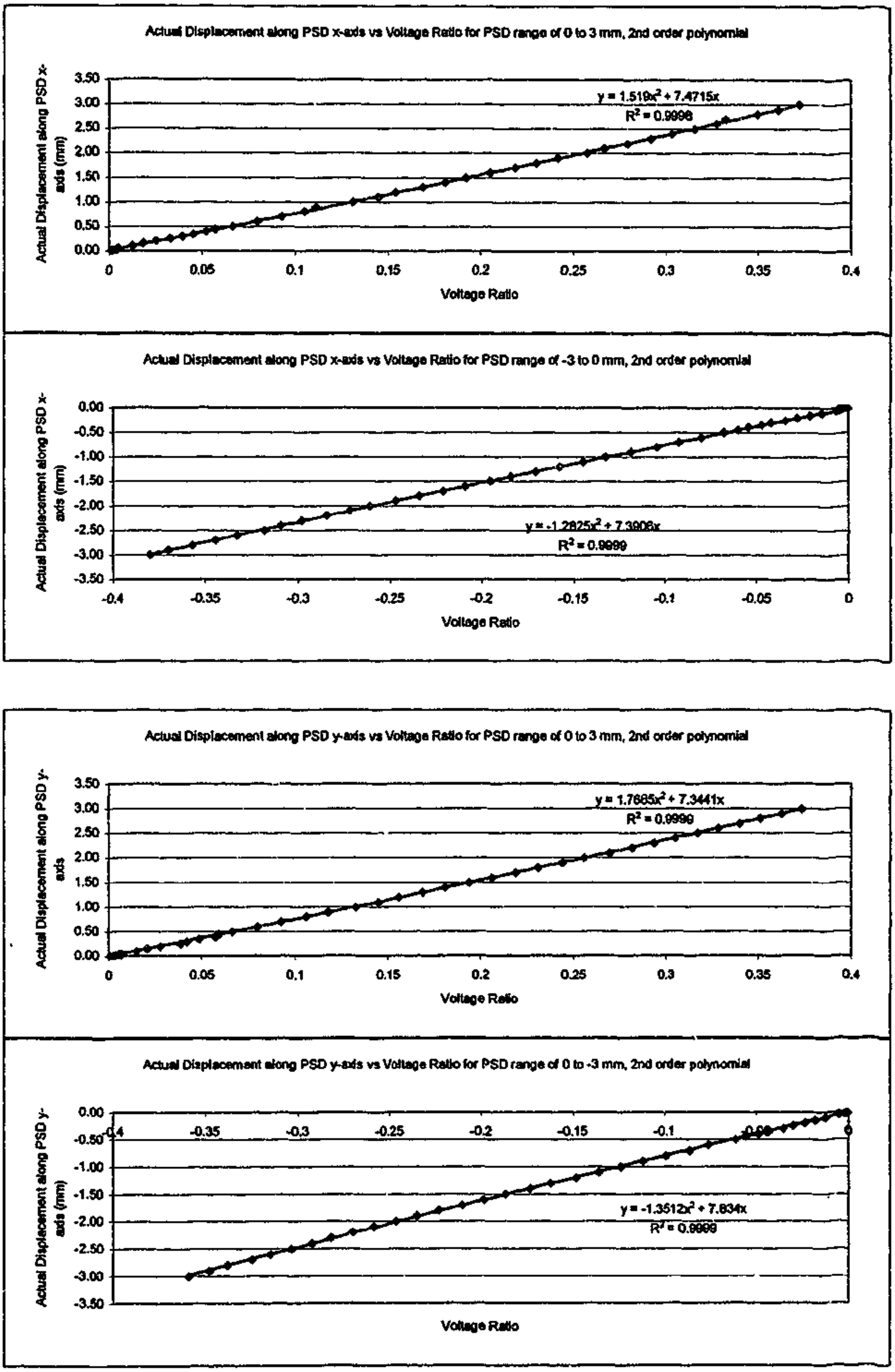


Figure 3.11: Calibration graph for the range of –3 to 3 mm along PSD x and y-axes using 2nd order polynomial

3.4.3 Experimental Analysis

The behaviour of the PSD sub-system is determined by displacing the PSD (thus moving the laser beam away from the centre) by a known distance and a collection of samples taken at each position. The positions calculated using Eqs. 3.3 to 3.6 are compared with the actual displacement. Fig. 3.12 and 3.13 show the mean errors and Root Mean Square (RMS) errors [40] between the calculated and the actual positions along the x and y-axes, respectively. From the figures, it can be observed that the mean error of the positions calculated using Eqs. 3.3 to 3.6 is ±0.16 mm. The calculated positions also have large fluctuations with a maximum RMS error of 0.17 mm.

To determine the resolution of the PSD sub-system, the following calculation is used. Using an input range of ±10 V, the smallest detectable voltage is 0.0048 V. Assuming that the PSD is linear within the ±3 mm range (Fig. 3.11), a beam displacement that will increase V_1 and V_2 by 0.0048 V will decrease V_3 and V_4 by the same amount. The resolution can then be determined by differentiating Eq. 3.3 and 3.5 to obtain:

$$\begin{aligned}\text{resolution along X} &= 4M_{x1}\left(\frac{-V_1^2 \times dV_3 - V_3^2 \times dV_1}{(V_1+V_3)^3}\right)\\ &\quad + 2M_{x2}\left(\frac{V_1 \times dV_3 - V_3 \times dV_1}{(V_1+V_3)^2}\right)\end{aligned} \tag{3.7}$$

$$\begin{aligned}\text{resolution along Y} &= 4M_{y1}\left(\frac{-V_4^2 \times dV_2 - V_2^2 \times dV_4}{(V_4+V_2)^3}\right)\\ &\quad + 2M_{y2}\left(\frac{V_4 \times dV_2 - V_4 \times dV_2}{(V_4+V_2)^2}\right)\end{aligned} \tag{3.8}$$

where $M_{x1} = 1.519$ and $M_{x2} = 7.472$ when $V_3 \geq V_1$;
$M_{x1} = -1.283$ and $M_{x2} = 7.391$ when $V_3 < V_1$;
$M_{x1} = 1.769$ and $M_{x2} = 7.344$ when $V_2 \geq V_4$;
$M_{x1} = -1.351$ and $M_{x2} = 7.834$ when $V_2 < V_4$;
V_c (c = 1,2,3 or 4) are voltages from corresponding channel number;
dV_c is the change in voltage.

The resolution for both axes was found to be 15 μm (refer to Appendix A).

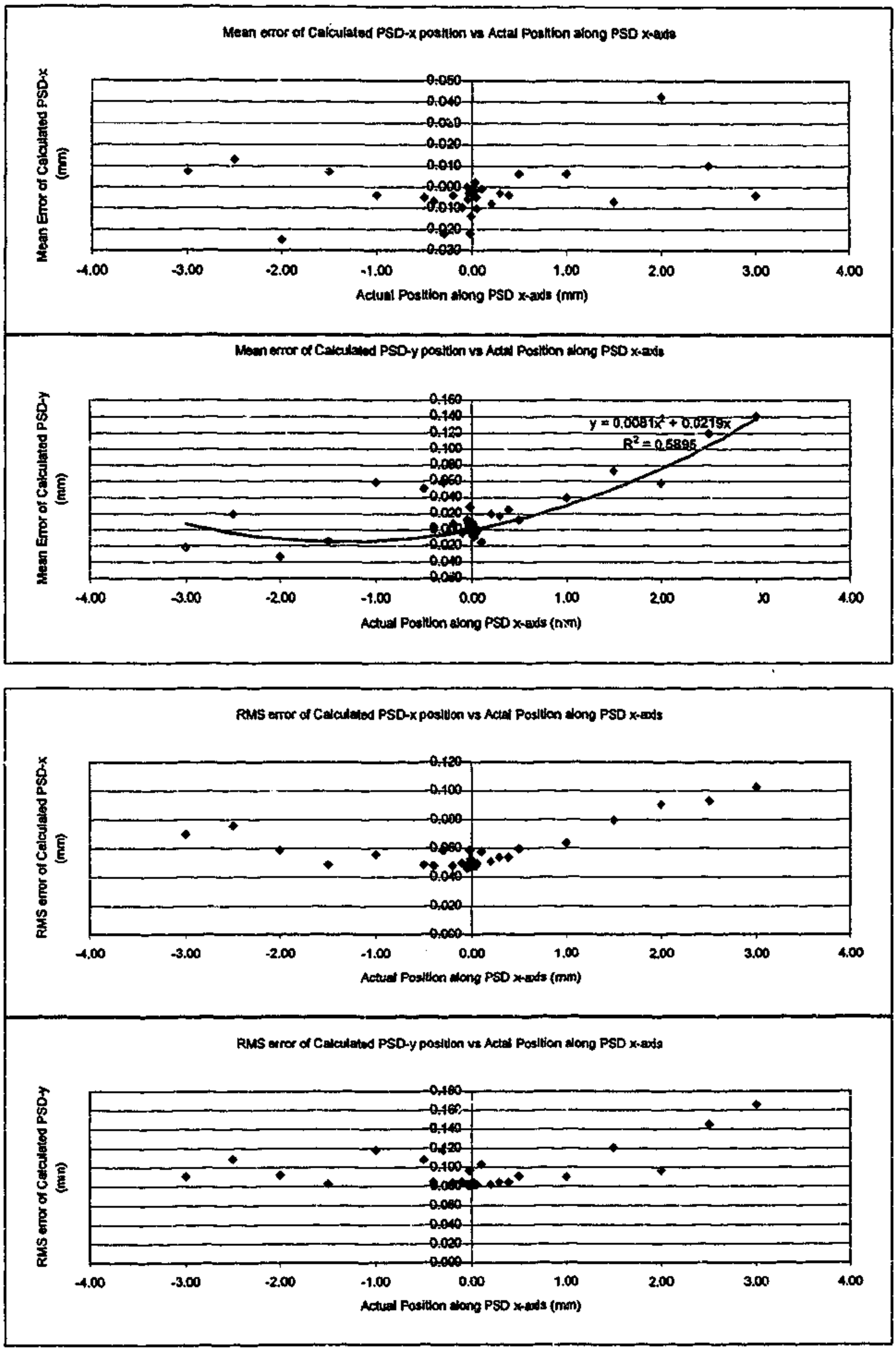


Figure 3.12: Mean error and RMS error between calculated and actual beam positions along PSD x-axis

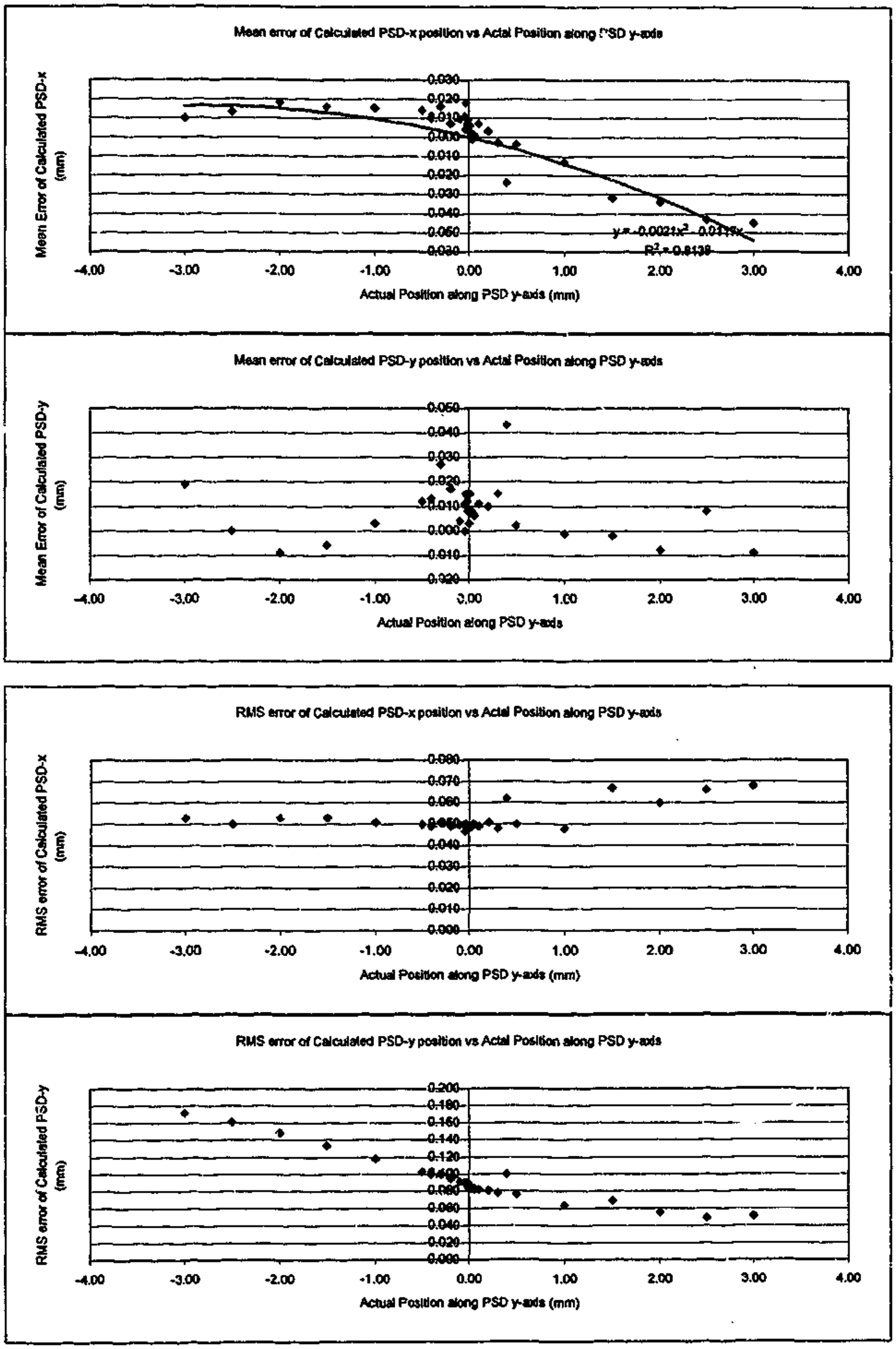


Figure 3.13: Mean error and RMS error between calculated and actual beam positions along PSD y-axis

The characteristics, thus performance of the PSD sub-system is dependent on several factors. These include:

1) Beam power directed onto the PSD;
2) Environment and equipment noise;
3) PSD uniformity and misalignment;
4) DAQ board resolution and accuracy.

Since the amount of photocurrent produced from each channel is directly proportional to the magnitude of the beam power incident on the PSD sensing area, a lower beam power generates a lower photocurrent. The ratio of this signal-carrying photocurrent to the ratio of the dark current (i.e. signal to noise (S/N) ratio) produced by the detector is therefore lower. This will reduce the resolution and accuracy of this sub-system. With the rise of temperature, the responsivity of the PSD decreases while the magnitude of dark current increases. This will also lower the S/N ratio, which further reduces the resolution and accuracy.

In addition to the dark current, there are other factors that will affect the performance. Fig. 3.14 shows the frequency analysis for a PSD channel. The frequency plot shows peaks at the frequency of 50Hz and at subsequent harmonics. A possible source of these peaks is the noise currents generated by the amplifier in the signal conditioning device, the PC and other surrounding electronics [26, 41]. The noise currents were determined by sampling the voltages from the PSD by covering the sensing surface of the PSD. In view of the fact that no light falls on the PSD, the voltages from all channels should be at minimum (i.e. 0.00V in this case). However, experimental results show that all channels exhibit high voltage fluctuation when the sensing surface of the PSD is covered. An example of a voltage plot is shown in Fig. 3.15. The effect of these noises can be examined by rewriting Eq. 3.7 and 3.8 as follows:

$$\text{dPSD-x} = 4M_{x1}\left(\frac{-I_1{}^2 \times dI_3 - I_3{}^2 \times dI_1}{(I_1 + I_3)^3}\right) + 2M_{x2}\left(\frac{I_1 \times dI_3 - I_3 \times dI_1}{(I_1 + I_3)^2}\right) \quad (3.9)$$

$$\text{dPSD-y} = 4M_{y1}\left(\frac{-I_4{}^2 \times dI_2 - I_2{}^2 \times dI_4}{(I_4 + I_2)^3}\right) + 2M_{y2}\left(\frac{I_4 \times dI_2 - I_4 \times dI_2}{(I_4 + I_2)^2}\right) \quad (3.10)$$

where dPSD-x and dPSD-y represents the change in PSD-x and PSD-y, respectively, dI_{nc} and I_c (c = 1,2,3 or 4) are noise currents and photocurrents from each channel, respectively. From these equations, it can be observed that higher signal-carrying photocurrents from each channel will increase both the numerator and denominator, but at a higher rate for the denominator. Therefore, the change of PSD-x or PSD-y due to the noise currents will be smaller. Moreover, a smaller coefficient M_{xi} or M_{yi} (i = 1, 2) will also reduce the effect of noise currents. However, this coefficient is directly related to the measuring range of the detector and can only be reduced by using a smaller detector [28].

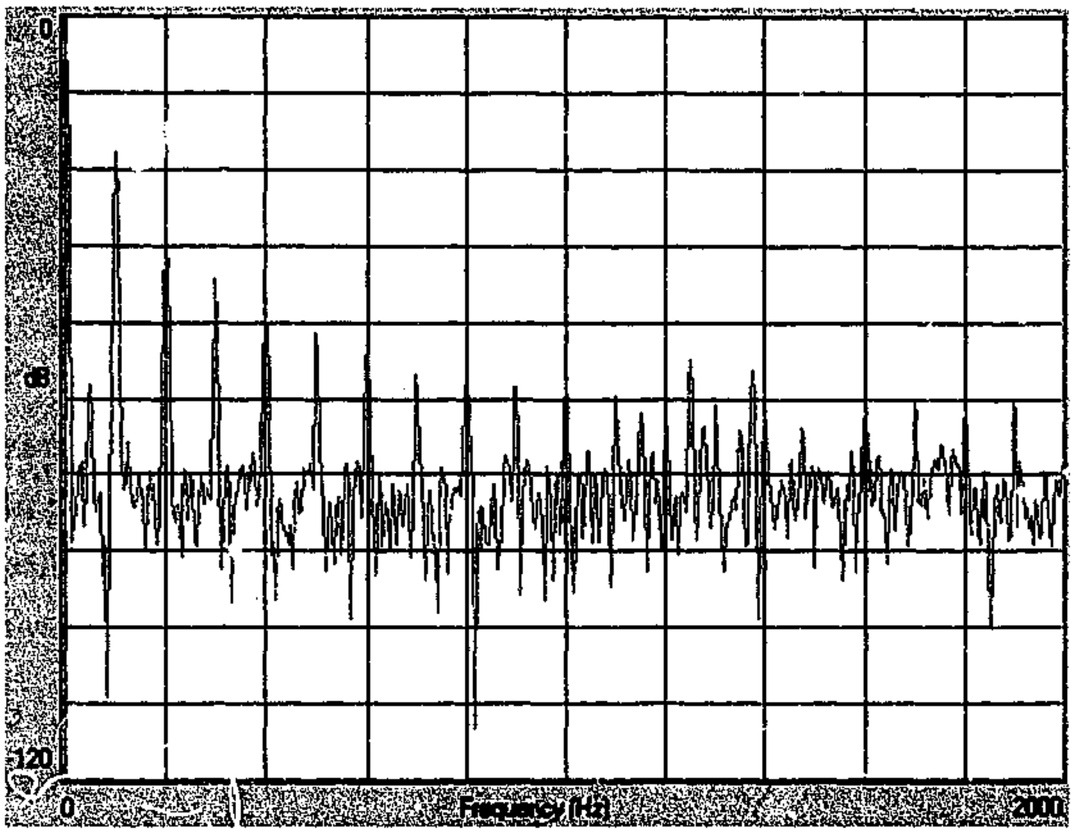


Figure 3.14: Fourier frequency transform of PSD's channel 12 sampling at 4000 Hz

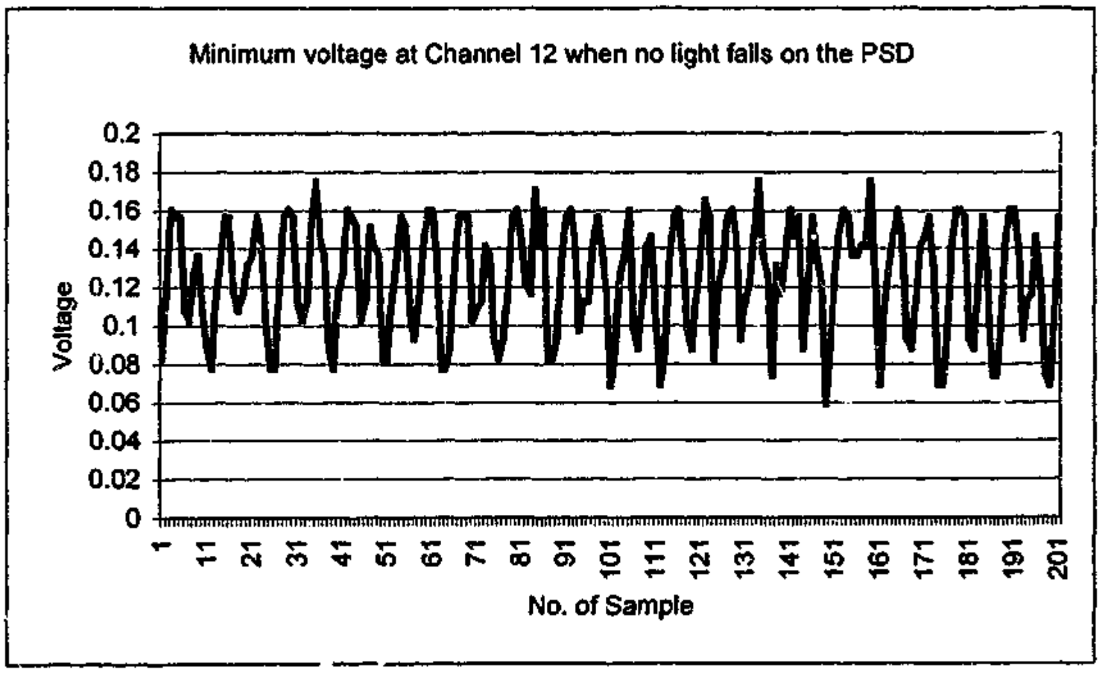


Figure 3.15: Minimum voltage from PSD's channel 12 with no light falling on PSD

To reduce the effect of noise, a moving average technique is used. The average of a fixed number of previous samples with the current sample is acquired. This will diminish the effect of the noise current, while allowing the detection of voltage change due to the actual offset of the beam from the centre of the PSD. Fig. 3.16 and 3.17 show the RMS errors of calculated beam positions along PSD x- and y-axes, respectively, using a moving average technique with a period of 5. By comparing with Fig. 3.12 and 3.13, it can be observed that the RMS errors reduces by an order of 2 with the use of moving average to smooth the signal. The corrected signal is therefore more stable. The number of samples used in the averaging should be resolved with the consideration that higher number of samples used will decrease the responsivity of the tracking controller to the sudden change in actual beam offset. The tracking controller will be lagging by a factor of 5 in this case.

Other measures that can be used to remove noise without affecting the responsivity of the controller are by using a better operational amplifier and other electronic components in the amplifier circuit. Appropriate shielding of the amplifier circuit from the surrounding electronics can also be implemented.

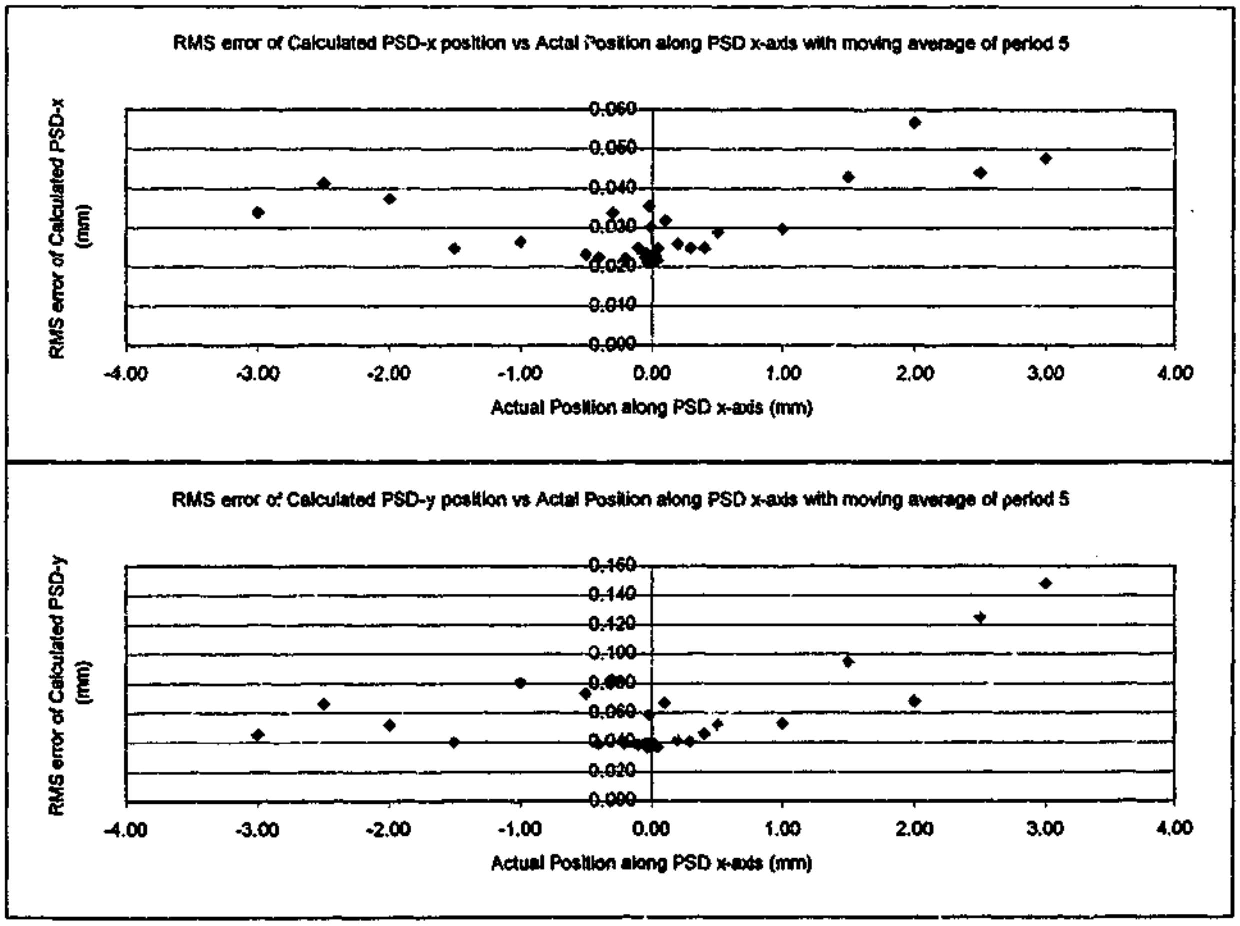


Figure 3.16: RMS error of calculated beam positions along PSD x-axis with moving average

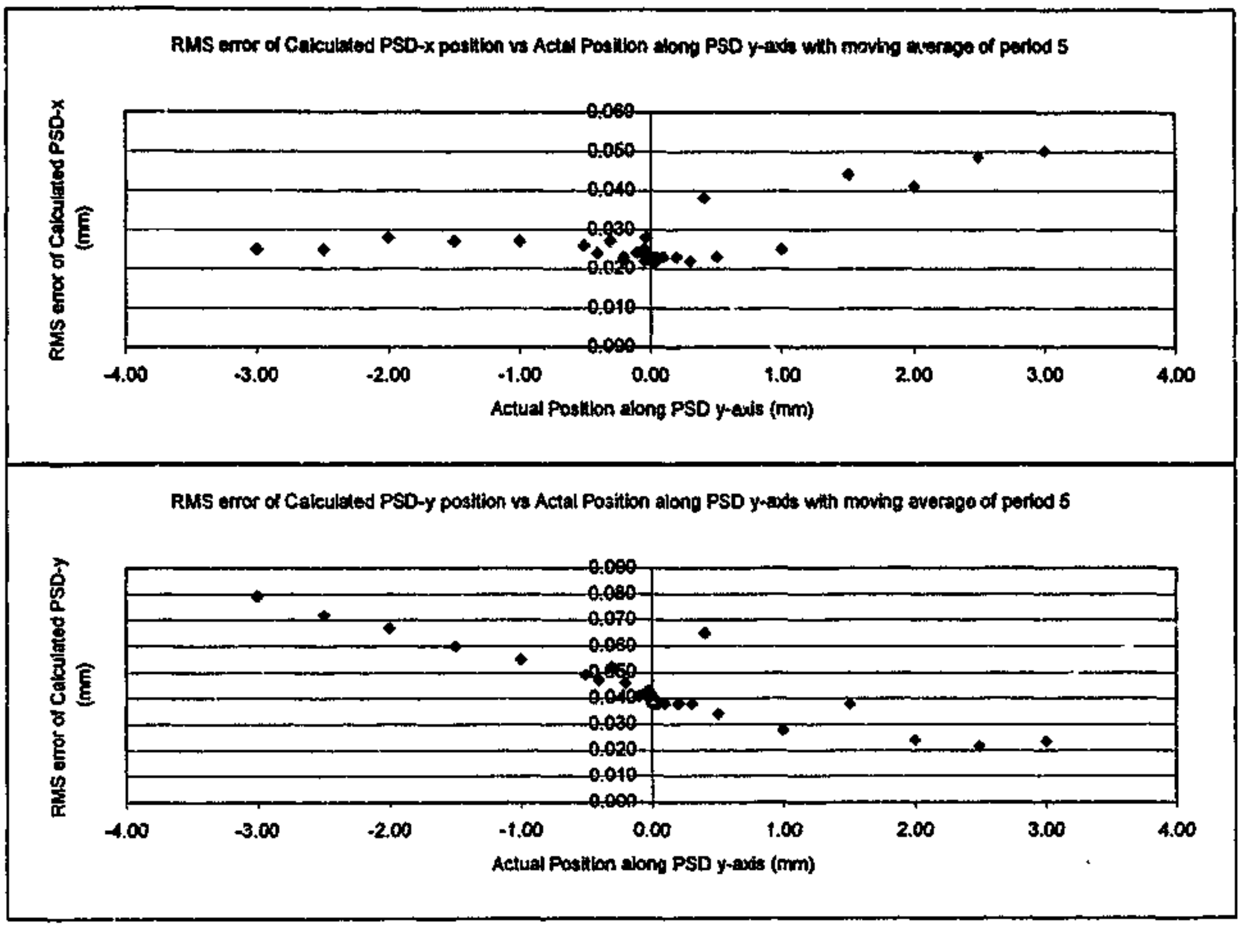


Figure 3.17: RMS error of calculated beam positions along PSD y-axis with moving average

A detailed analysis of Fig. 3.12 and 3.13 show that mean error of calculated beam position PSD-x changes with the translation of beam along PSD y-axis and vice versa. This is due to the non-uniform responsivity of the PSD throughout the surface as well as misalignment of the PSD. This relationship can be represented by the following equations:

$$\mathrm{ErrX} = -0.0021 \times \text{PSD-y}^2 - 0.0117 \times \text{PSD-y} \tag{3.11}$$

$$\mathrm{ErrY} = 0.0081 \times \text{PSD-x}^2 + 0.0219 \times \text{PSD-x} \tag{3.12}$$

where ErrX and ErrY represents the errors of calculated beam positions, PSD-x and PSD-y, respectively.

By compensating the error of calculated beam positions with results from Eqs. 3.11 and 3.12, followed by the moving average technique, the mean error of the calculated PSD positions has been reduced to ±0.07 mm with a maximum RMS error of 0.09 mm. The results for beam positions along the PSD x- and y-axes are plotted in Fig. 3.18 and 3.19.

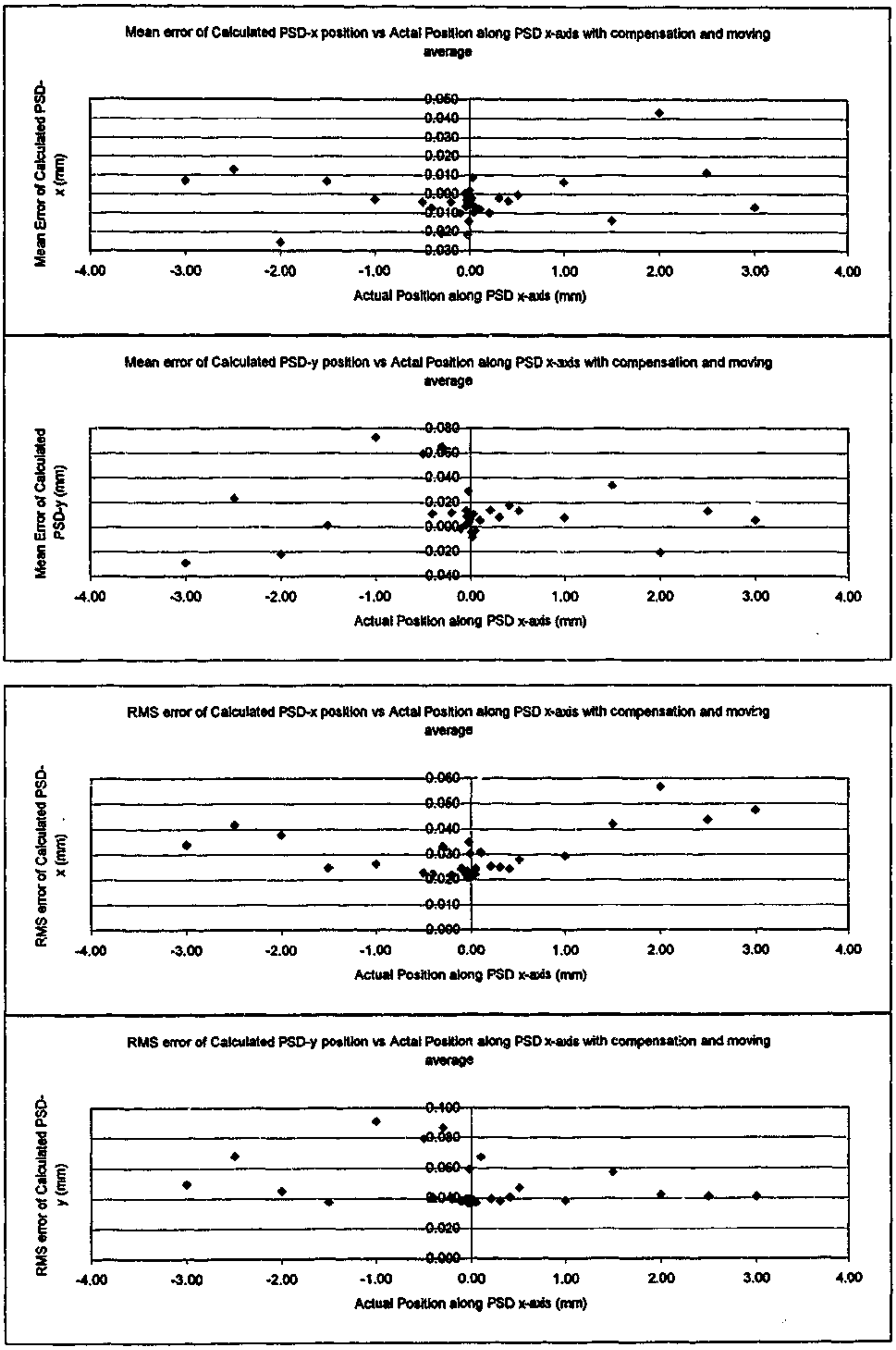


Figure 3.18: Mean error and RMS error of calculated beam positions along PSD x-axis with compensation and moving average

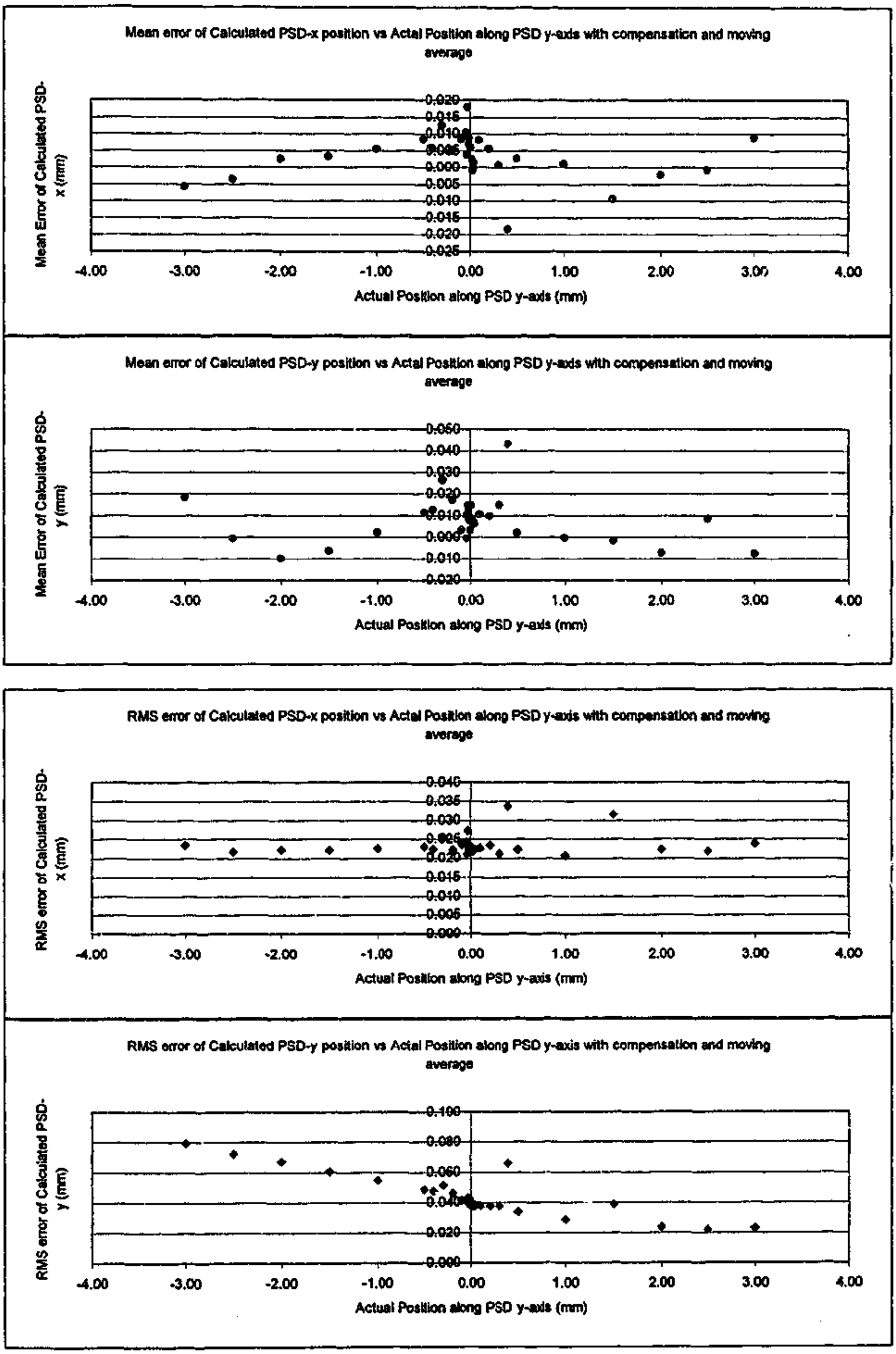


Figure 3.19: Mean and RMS of calculated beam positions along PSD y-axis with compensation and moving average

It must be emphasised that the improved mean error and RMS error using above calibration techniques applies only to this particular PSD connected to specific channels on the signal conditioning device. Calibration has to be performed all over again in the event of a change of PSD or change in its set-up, as well as a switch to other channels.

As mentioned before, the DAQ board uses an A/D converter to convert the input analogue voltage into a digital representation. A higher resolution A/D converter will provide a more accurate digital representation of the analogue voltage. In addition, when the input voltage is increased, the digital representation should also increase linearly. In reality, some non-linearity exists and this is measured by the Effective Number of Bits (ENOB) [42, 43], which measures the overall accuracy of the DAQ board. A higher ENOB DAQ board will provide more accurate measurements.

3.4.4 Sub-system Interface

Two different types of input operations are provided by the DAQ software development kit (SDK) [44]. They are the single value operation and the continuous buffered input operation. The single value operation is preferred because real time data can be obtained at a higher rate. In the single value operation, each data value is being sampled and returned immediately. For a 4 channels sampling, the channels are sampled sequentially by changing the channel in the software program. The software calculates the relative position of the beam after sampling all 4 channels. The current maximum update rate is 1000 Hz (4000 Samples per second). This rate is slow given that the maximum sampling rate specified is 1×10^5 samples per second for 4 channels. This is again due to the fact that the PC's processor is slow in processing the user commands and the sampled data as well as the multitasking nature of the Windows operating system. This rate can be increased by the use of a faster processor.

The PSD sub-system is the most important sub-system in the LISM apparatus. The performance of this sub-system will directly affect the performance of the LISM apparatus. With an update rate of 1000 Hz (500 Hz in Nyquist Frequency) and a maximum position measuring range of ±3 mm, the maximum target velocity of 750 mm/s in the direction perpendicular to the laser beam can be achieved (1 mm lateral offset of the target retroreflector generates 2 mm tracking offset on the PSD). As mentioned previously, the use of moving average will reduce the responsivity of the controller and thus lower the maximum target velocity. A moving average with a period of 5 will reduce the maximum target velocity to about 150 mm/s.

3.5 Beam Steering sub-system

3.5.1 Specifications

This sub-system consists of a stepper motor with an optical encoder and a servo motor with resolver. These motors used to provide the horizontal and vertical rotations of the beam steering mirror, allowing the laser beam to be directed towards the centre of the retroreflector. The vertical axis is driven by a Compumotor DR1100E direct drive servo motor system with resolver feedback [45]. The resolution of this motor is 0.00059° with an accuracy of 0.0125°. A maximum speed of 1.5 revolutions per second is permitted. The horizontal axis is driven by a Rorze stepping motor driver with microstep capability of up to 80000 steps per revolution (0.0045°) [46]. The encoder attached to the motor has a resolution of 40000 steps per revolution (0.009°). The maximum velocity of this axis is 6.25 revolutions per second. This axis is driven in open-loop as the motor may become unstable in a closed-loop configuration. In order to satisfactorily implement a closed-loop stepper motor system, the motor must have a resolution 4 times higher than the encoder [47]. With the current encoder, a motor resolution of 160000 steps per revolution is required.

Both motor systems are connected to the Acroloop ACR2000 4-axes motion controller card [48]. This is a 32-bit floating point, multi-tasking Digital Signal Processor (DSP) based motion controller. The controller has a separate operating system with onboard multi-tasking executive. This separate operating system allows the time critical events to be handled by the Acroloop controller, which thus release the supervisory control unit's processor for other tasks.

3.5.2 Experimental Analysis

The most important function of the beam steering sub-system is to provide optimal and smooth tracking of the moving target retroreflector. As the motion of the target retroreflector in 3D workspace is random, the motor controller of the beam steering sub-system must have fast response to follow the random motion of the target. A high response is also critical in maintaining the interference between the measurement and the reference beams at all times during tracking for dynamic measurements. As the ACR2000 controller employs a full PID control algorithms, the PID settings and the motion profile of the motors must be properly tuned to improve performance [49].

Another function of the beam steering sub-system is to provide the supervisory control unit with the current angular positions of both motors for the calculation of the Cartesian co-ordinates of the target. This requires that the steady-state error of the motors to be as small as possible in order to obtain high accuracy measurements. High motor resolution is also necessary for smooth tracking and accurate measurements.

In this study, the beam steering sub-system's configuration shown in Fig. 3.20 is employed. In this configuration, the first mirror's position is fixed and is used to direct laser beam towards the second mirror directly on top. The second mirror is rotated by the 2 motors to direct the beam towards the target retroreflector. As there is only 1 moving mirror, this configuration is easier to fine-tune. However, the laser beam path is perpendicular to the rotation axis of the 2nd motor. The effective resolution of the 2nd motor is reduced by a factor of 2, as a motor rotation of 1° will result in a beam angular displacement of 2°.

The performance of the motors was determined by investigating the response of each motor to a step input. By varying the motion profile (i.e. velocity and acceleration settings) and the PID values, the steady state error, rise time and percentage of overshoot were determined from the response curve to a step input. The step sizes selected was for a tracking offset of 1.5 mm with retroreflector at various radial distances (beam displacement from the intersection of the horizontal and vertical rotation axes). A sample response curves for both motors are presented in Fig. 3.21 and 3.22, with the other response characteristics tabulated in Table 3.1.

From Table 3.1, it can be observed that the servo motor has negligible steady state error compared to the stepper motor. This is due to undetected position loss in the open-loop stepper system. The percentage of overshoot in the servo motor is high due to high gains. Beam offset produced due to overshooting is nonetheless within the ±3 mm range, so beam interference is not interrupted. The stepper motor has zero overshoot. The rise time for both motors are about 20 ms. However, it can be noticed that both motors require a considerable amount of time (about 300 ms) to reach steady state. For the reasons that a new motion command will not be executed when one is currently active and that the commanded position cannot be changed on the fly, this long delay can affect the performance of the LISM apparatus during a target following process.

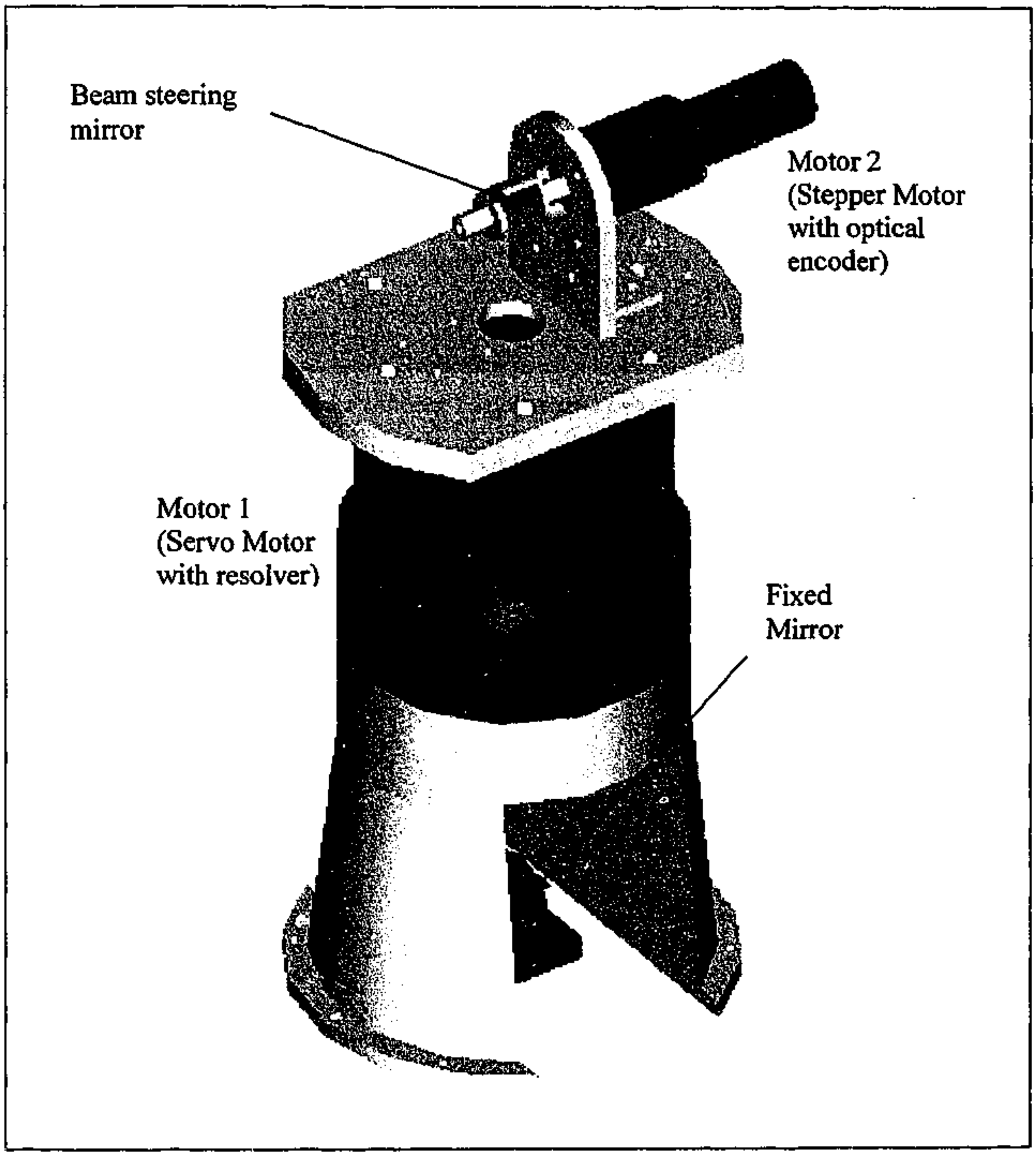

Beam steering mirror
Motor 2 (Stepper Motor with optical encoder)
Motor 1 (Servo Motor with resolver)
Fixed Mirror

Figure 3.20: Beam steering sub-system with 2 mirrors

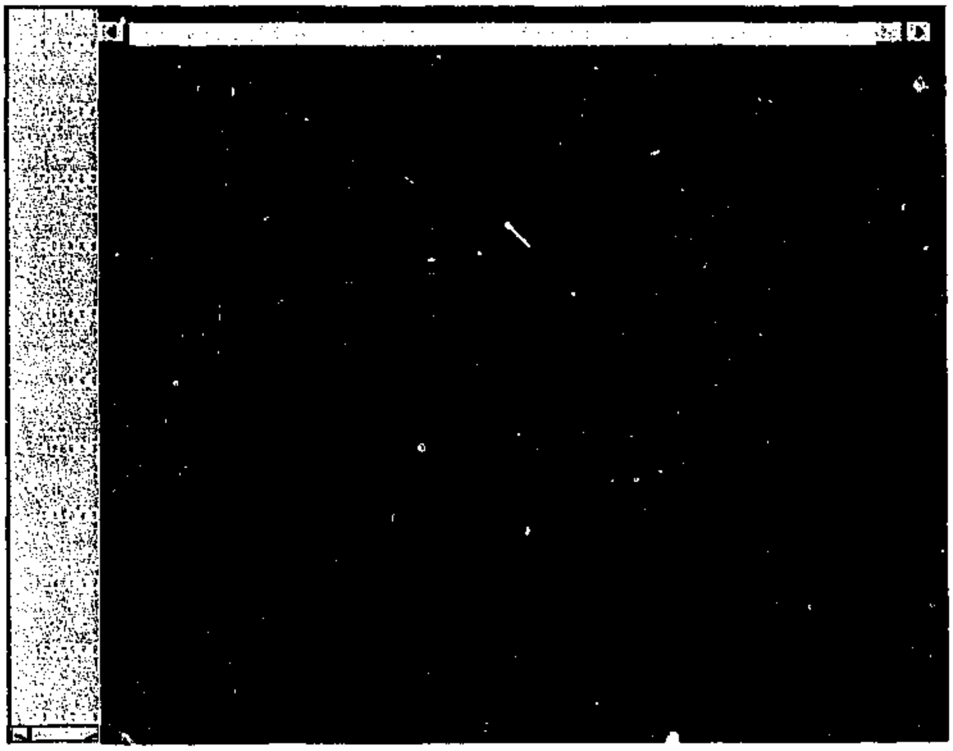


Figure 3.21: Motor 1 response with target at 1000mm

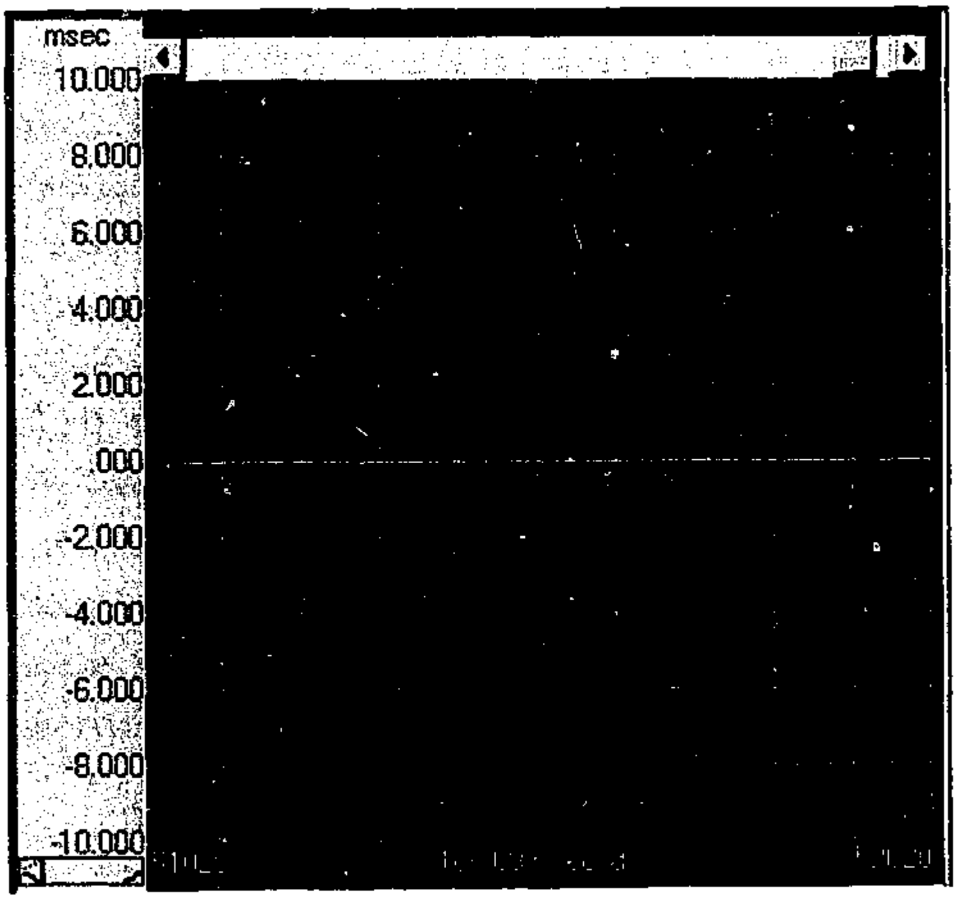


Figure 3.22: Motor 2 response with target at 1000mm

Table 3.1: Motor response characteristics

Motor 1							
Radial Distance (mm)	Input step size	Final angular displacement (steps)	Overshoot (steps)	(%)	Rise Time (ms)	Steady state error	Beam offset due to overshoot (mm)
1000	147	146	33	22.45	20	1 step	0.337
500	293	292	17	5.80	20	1 step	0.087
300	489	488	67	13.70	20	1 step	0.206
100	1467	1466	133	9.07	20	1 step	0.136

Motor 2							
Radial Distance (mm)	Input step size	Final angular displacement (steps)	Overshoot (steps)	(%)	Rise Time (ms)	Steady state error	Beam offset due to steady state error (mm)
1000	10	8	0	0.00	20	2 steps	0.314
500	19	20	0	0.00	20	1 step	0.079
300	32	30	0	0.00	20	2 steps	0.094
100	96	96	0	0.00	20	0	0.000

3.6 Supervisory Control Unit

3.6.1 Specifications

The supervisory control unit (SCU) used in this set-up is an Intel based Pentium III 866 MHz personal computer with 256Mb RAM. SCU is running on Windows NT and the control software is developed using Microsoft Visual C++ 6. The object-orientated programming architecture in C++ enables decomposition and abstraction of the sub-system's operations into object modules. The decomposition and abstraction offer easier management of the complexity of the program due to the amount of code and the range of activities [50, 51]. Further, all the details of the operations of the sub-systems can be hidden from the user. This program architecture is also more efficient when the hardware used in each sub-system is changed, as only the object responsible for the operation of the sub-system has to be modified [38]. Fig. 3.23 shows the schematic diagram of the sub-systems' structure in the SCU PC. The software architecture of the sub-systems is described in the next section.

Fig. 3.23 shows that the main module holds the Target Position Computation (TPC). TPC requires the current motors' angular displacements, along with the tracking offset measured by the PSD and radial distance recorded by the laser interferometer. These values are obtained via the sub-systems' objects, which communicate with the respective hardware's drivers to send commands and poll data from the hardware. As described in Section 3.5.1, the on-board operating system in the Acroloop controller allows the off-loading of the time-critical motion program into the controller. In this set-up, the Target Following Control Algorithm (TFCA) will be loaded into the on-board program. User variables can be defined in the on-board program to store external data transmitted into the Acroloop hardware registers [52]. Variables for the PSD readings and the Zygo measurement will be updated by the PSD and ZYGO object's thread, respectively, at its sampling rate. Encoder object provides the main module with the sampled encoder counts.

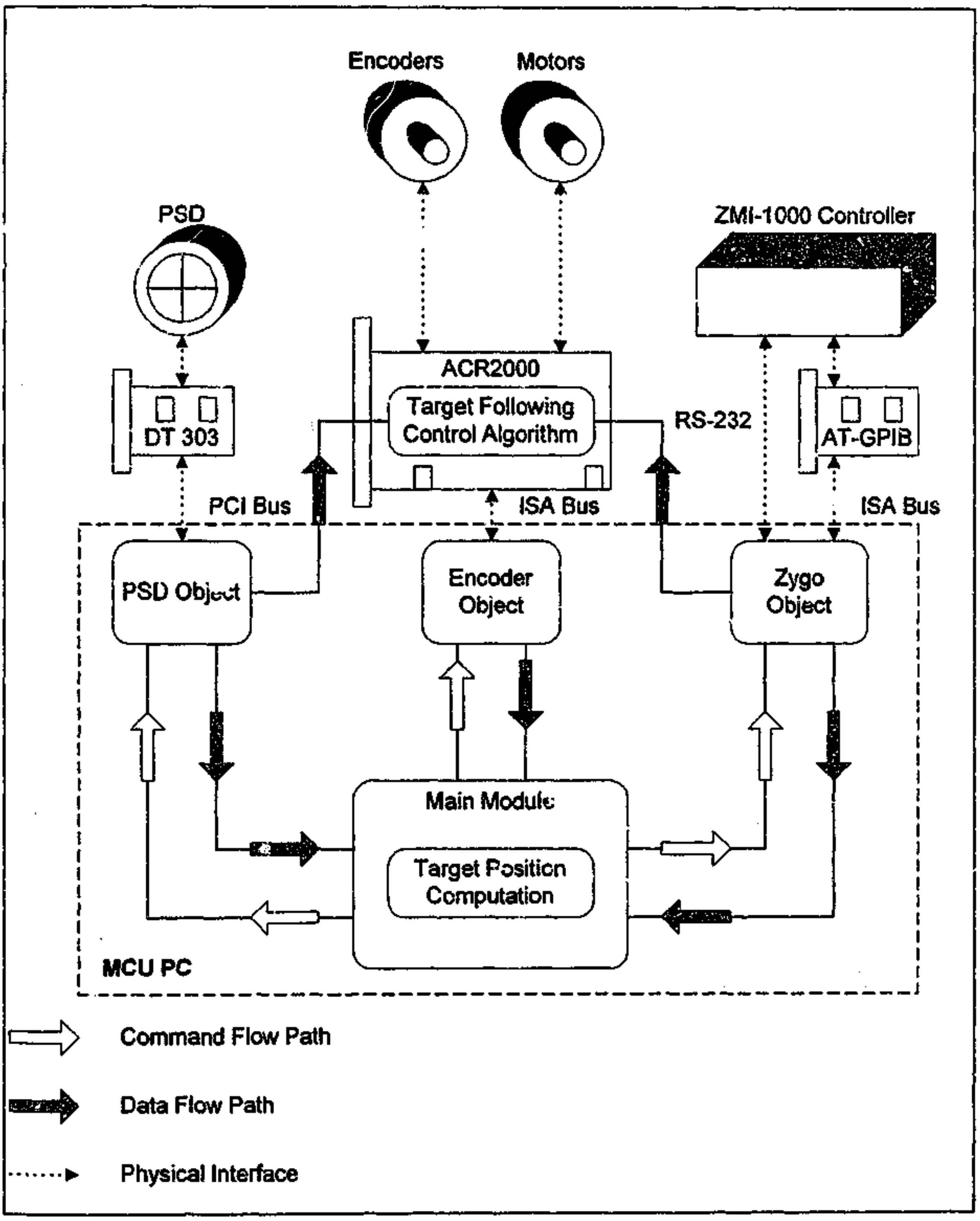


Figure 3.23: Schematic diagram of LISM sub-systems' structure

3.6.2 Sub-systems' Software Architecture

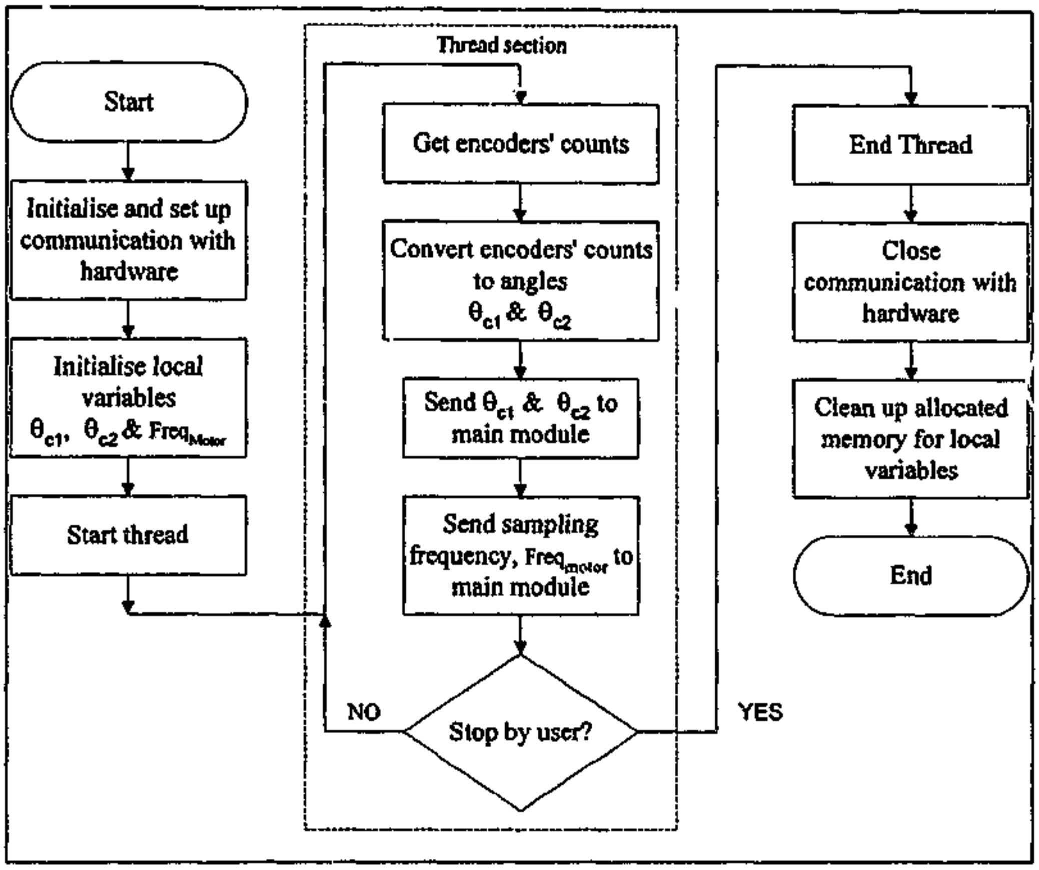
Three ohjects are used to represent three sub-systems. These are the Encoder Object, PSD Object and Zygo Object. Each uses different code to operate its associated hardware. Fig. 3.24 to 3.26 show the flowcharts of the software architectures for these objects.

Each object, once started, will first initialise and commence communication with the corresponding hardware. Variables used to store the corresponding raw values are then declared and initialised. As shown in Fig. 3.24 to 3.26, each object consists of a thread. A thread is a C++ programming facility providing the support for pre-emptive multitasking (i.e. the ability to run more than one section of code simultaneously) [53, 54] within a program by switching rapidly between running sections. The thread in each object continuously loops through its section of code to sample the raw values from the corresponding hardware. The raw values are converted into informative data that can be used by the TFCA and the TPC in the main module. These data and the monitored sub-systems' frequency are transmitted to the Acroloop hardware registers and the main module, respectively, when requested. The operating system decides which section of code is to be run at a given time, based upon factors such as priority setting and whether a particular section is waiting to read or write. By switching between threads, data sampling from all sub-systems appear to be running simultaneously. However, it is not desirable to have too many threads running at the same time due to the processor's and computer's memory limitations. Therefore, careful consideration must be taken as to whether each thread is necessary. Further, the programmer has no control over how often a thread is executed. A thread with a higher priority setting or associated with simpler task will be scanned through more often.

When the user stops the control program, each object's sampling thread will first be terminated. The communication with the hardware will then be closed and any memory allocated for the variables will be cleared.

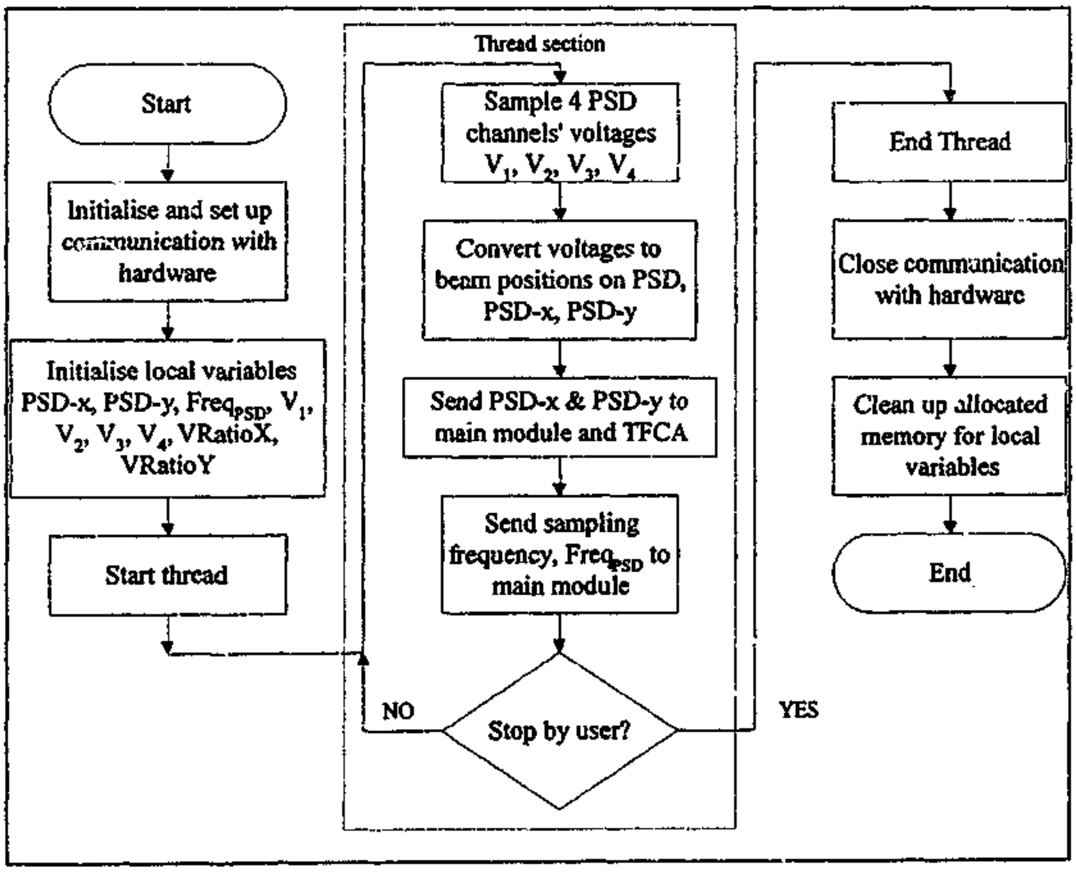
3.6.3 Control Program Software Architecture

When the control program is executed, the link with all objects is first initialised. Variables used in the program will then be declared and initialised. A window interface as shown in Fig. 3.27 will then be presented to the user. Fig. 3.28 shows the flowchart of the software architecture for the control program.



Thread section
Start
Initialise and set up communication with hardware
Initialise local variables θc1, θc2 & FreqMotor
Start thread
Get encoders' counts
Convert encoders' counts to angles θc1 & θc2
Send θc1 & θc2 to main module
Send sampling frequency, Freqmotor to main module
Stop by user?
NO
YES
End Thread
Close communication with hardware
Clean up allocated memory for local variables
End

Figure 3.24: Software architecture of Encoder object



Thread section
Start
Initialise and set up communication with hardware
Initialise local variables PSD-x, PSD-y, FreqPSD, V1, V2, V3, V4, VRatioX, VRatioY
Start thread
Sample 4 PSD channels' voltages V1, V2, V3, V4
Convert voltages to beam positions on PSD, PSD-x, PSD-y
Send PSD-x & PSD-y to main module and TFCA
Send sampling frequency, FreqPSD to main module
Stop by user?
NO
YES
End Thread
Close communication with hardware
Clean up allocated memory for local variables
End

Figure 3.25: Software architecture of PSD object

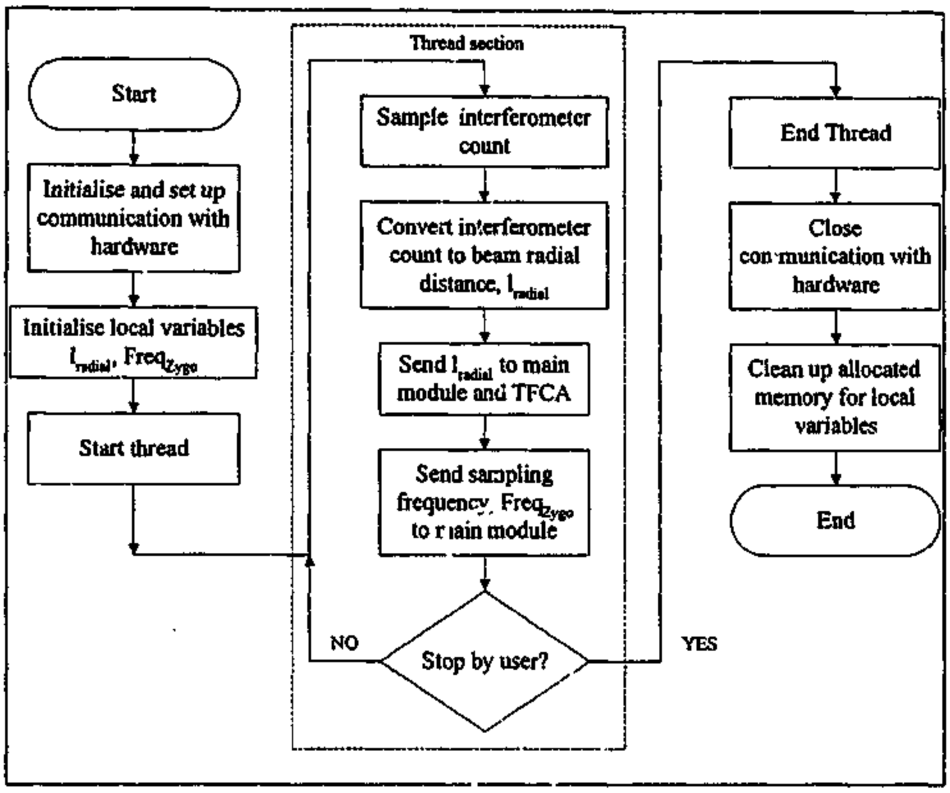

Thread section
Start
Initialise and set up communication with hardware
Initialise local variables l_{radial}, $Freq_{Zygo}$
Start thread
Sample interferometer count
Convert interferometer count to beam radial distance, l_{radial}
Send l_{radial} to main module and TFCA
Send sampling frequency, $Freq_{Zygo}$ to main module
Stop by user?
NO
YES
End Thread
Close communication with hardware
Clean up allocated memory for local variables
End

Figure 3.26: Software architecture of Zygo laser interferometer object

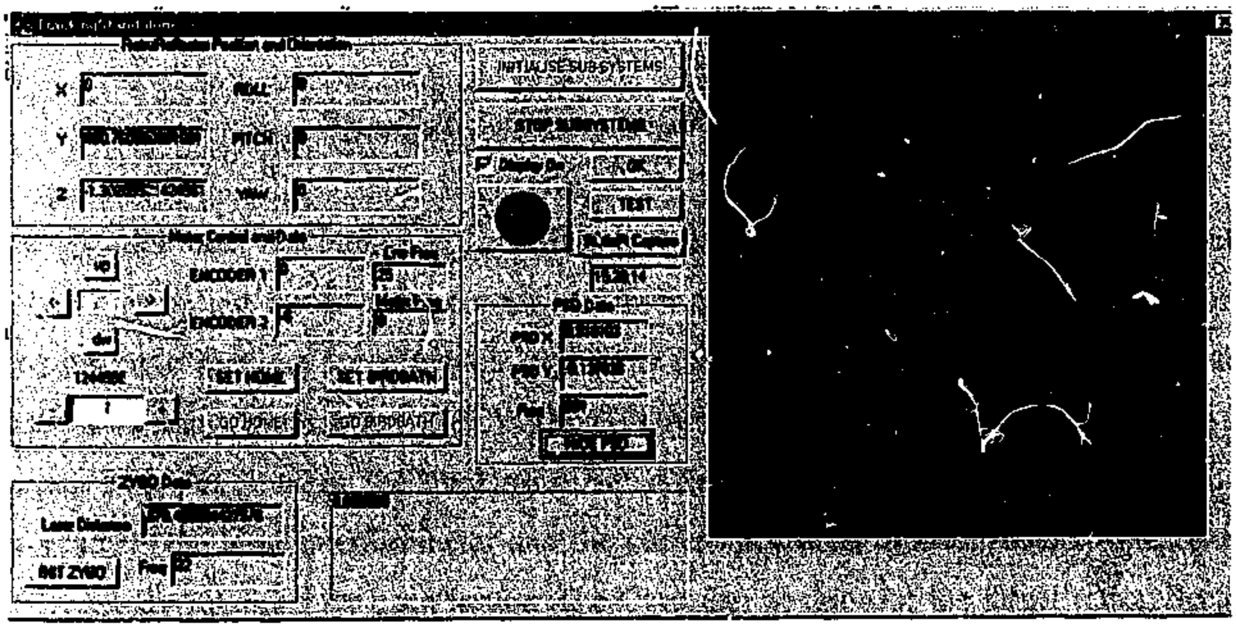


Figure 3.27: Control software user interface

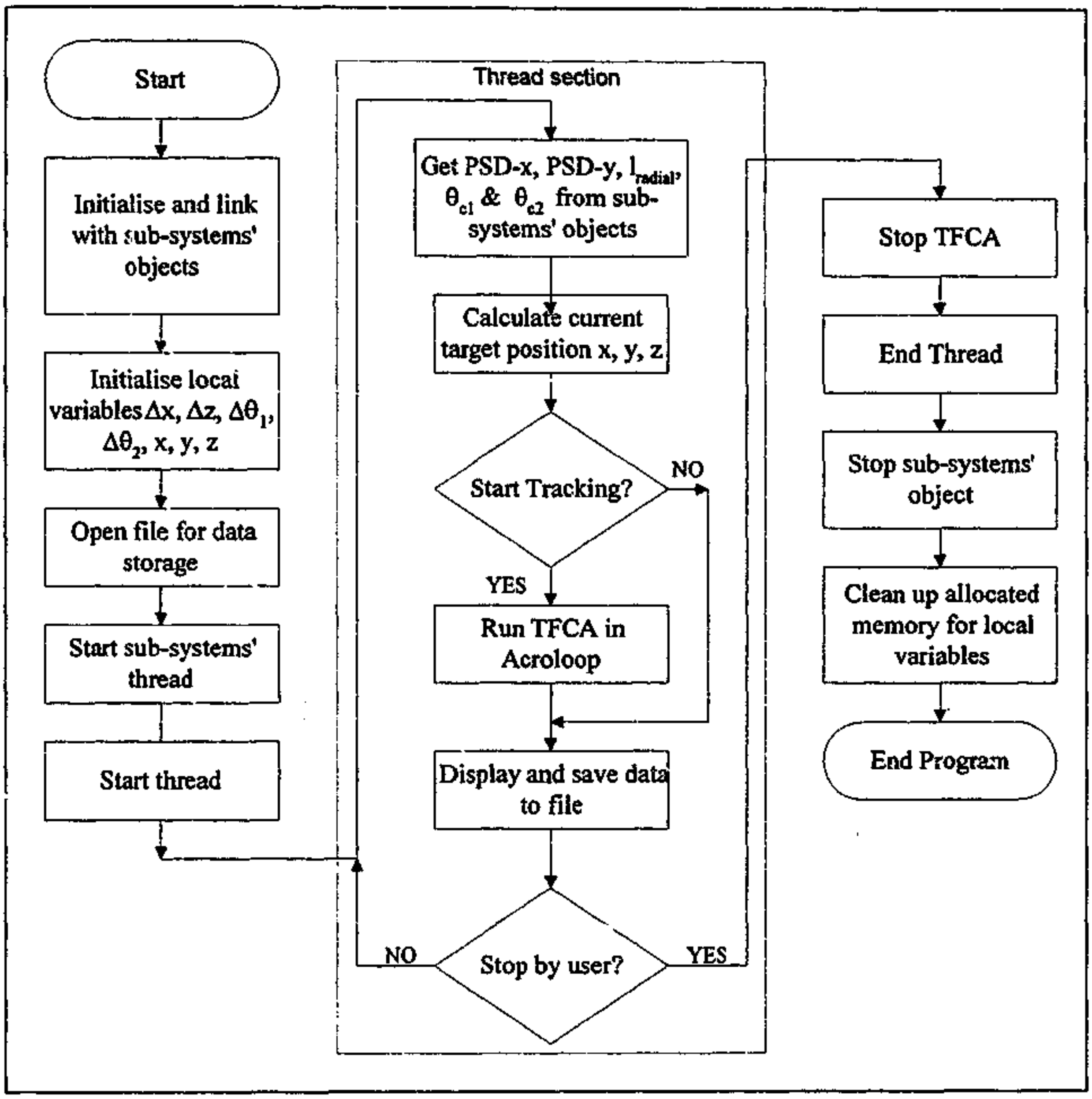

Start
Initialise and link with sub-systems' objects
Initialise local variables Δx, Δz, $\Delta\theta_1$, $\Delta\theta_2$, x, y, z
Open file for data storage
Start sub-systems' thread
Start thread
Thread section
Get PSD-x, PSD-y, l_{radial}, θ_{c1} & θ_{c2} from sub-systems' objects
Calculate current target position x, y, z
Start Tracking?
NO
YES
Run TFCA in Acroloop
Display and save data to file
Stop by user?
NO
YES
Stop TFCA
End Thread
Stop sub-systems' object
Clean up allocated memory for local variables
End Program

Figure 3.28: Software architecture of control program

The operator uses the buttons to initialise each sub-system and start or stop each sub-system's sampling thread. The control program consists of another thread. This thread contains code sections for kinematic equations (to be discussed in the next chapter) for Target Position Computation (TPC). The TFCA will only be carried out if tracking is started. The TFCA takes the tracking offset measured by the PSD, the radial distance recorded by the laser interferometer and the current positions of both motors to compute the necessary rotations for each motor. The Cartesian position of the target retroreflector with respect to a reference co-ordinate frame will be calculated using the kinematic equations and stored in the allocated variables, updating on the screen every second. The Cartesian position, together with the measurements recorded from the sub-systems, will also be written into a file in this thread for off-line analysis purposes. When the program is stopped, each object will be called to terminate its sampling thread. The target following, target position calculation and file writing processes will then be terminated. Link with objects and memory allocated for local variables will be cleared.

Table 3.2 tabulates sampling frequency of all sub-systems and the rate of execution of control program's thread. From the table, it can be observed that the limiting factor that affects the speed of tracking is the execution rate of the TFCA in the motor controller. An overall LISM update rate of 50 Hz (20 ms) can be achieved, which is just sufficient for real-time target following. Computation of the target position is carried out at 15 Hz.

The tracking and measuring rate in the current set-up is limited by the low processing power of the PC. Maximum sampling rate for the sub-systems cannot be achieved due to limited real-time capability of the Windows operating system. The slow sampling rate is also due to high processor load caused by the large number of sampling threads in the control program. The tracking and measuring rate can be improved by adding an analogue and RS232 input on the Acroloop controller card. PSD and Zygo data can be transmitted directly into the Acroloop hardware registers, which thus reduced the load of the PC processor. Additional DAQ board and hardware driver are not required, which will improves the sampling rate of the sub-systems. High data transfer rate in the Acroloop DSP (Texas Instruments^{TM} TMS320C3X [48, 55]) also allow for higher sampling rate. The SCU PC will be used to provide user interface and TPC, while the DSP in the Acroloop controller will be employed solely for TFCA. This design should result in a more efficient PC-DSP hybrid LISM control architecture.

Table 3.2: Sampling frequencies of sub-systems and thread in control program

Processes	Min (Hz)	Max (Hz)
PSD sampling	514	584
Encoder sampling	10	26
ZYGO sampling using RS232	81	88
Computation with TPC	10	15
TFCA in Acroloop Controller	40	50

3.7 Target Following Control Algorithm (TFCA)

As mentioned in the previous section, the motor controller includes an inner-loop PID controller. The TFCA converts the measurements from the sub-systems to the command signal for the motor controller. This section outlines the algorithm used for this conversion.

3.7.1 Estimation of Birdbath's Laser Radial Distance and Motor Angles

Prior to the target following process, a tracking reference (TR) point has to be located. It was decided that this point should be the intersection point between the horizontal and vertical rotation axes. The LISM control system records the radial distance from point TR to the target, together with encoders' values and the PSD offsets to calculate the necessary rotations for both motors. Moreover, an initial reference target position, called the birdbath, was used to initialise the laser interferometer as well as the azimuth and elevation angles of both motors. Fig. 3.29 shows the schematic representation of the beam steering mechanism, illustrating the birdbath and its relative distance from the TR.

3.7.2 Motor Angular Rotation Calculation

As Motor 2 of the beam steering mechanism is responsible for maintaining tracking in the LISM z direction, calculation for the amount of rotation required for this motor was derived from the schematic shown in Fig. 3.30. The figure shows the retroreflector's initial position at point A at time t=0 with the laser beam path shown in red. The laser beam is hitting the centre of the retroreflector and there is no beam offset. Point B indicates a new retroreflector position at time $t=t_u$ (where t_u is the LISM update period). The dotted line indicates the propagation of the laser beam before any rotation of the mirror has taken place. The current radial distance recorded by the interferometer will be half of the distance of the path TR-C-D-E-F. Since C-D = E-F = ½ x A-C, this distance is equivalent to the distance TR-A (refer to Appendix B). The target lateral offset Δd_2 is calculated as described in the next section. The commanded angular displacement for motor 2 $\Delta\vartheta_2$ is calculated using the following equation:

$$\Delta\vartheta_2 = \frac{1}{2}\tan^{-1}\left(\frac{\Delta d_2}{l_{radial}}\right) \tag{3.13}$$

where l_{radial} represents the current radial distance recorded by the laser interferometer and Δd_2 represents the current target lateral offset for motor 2.

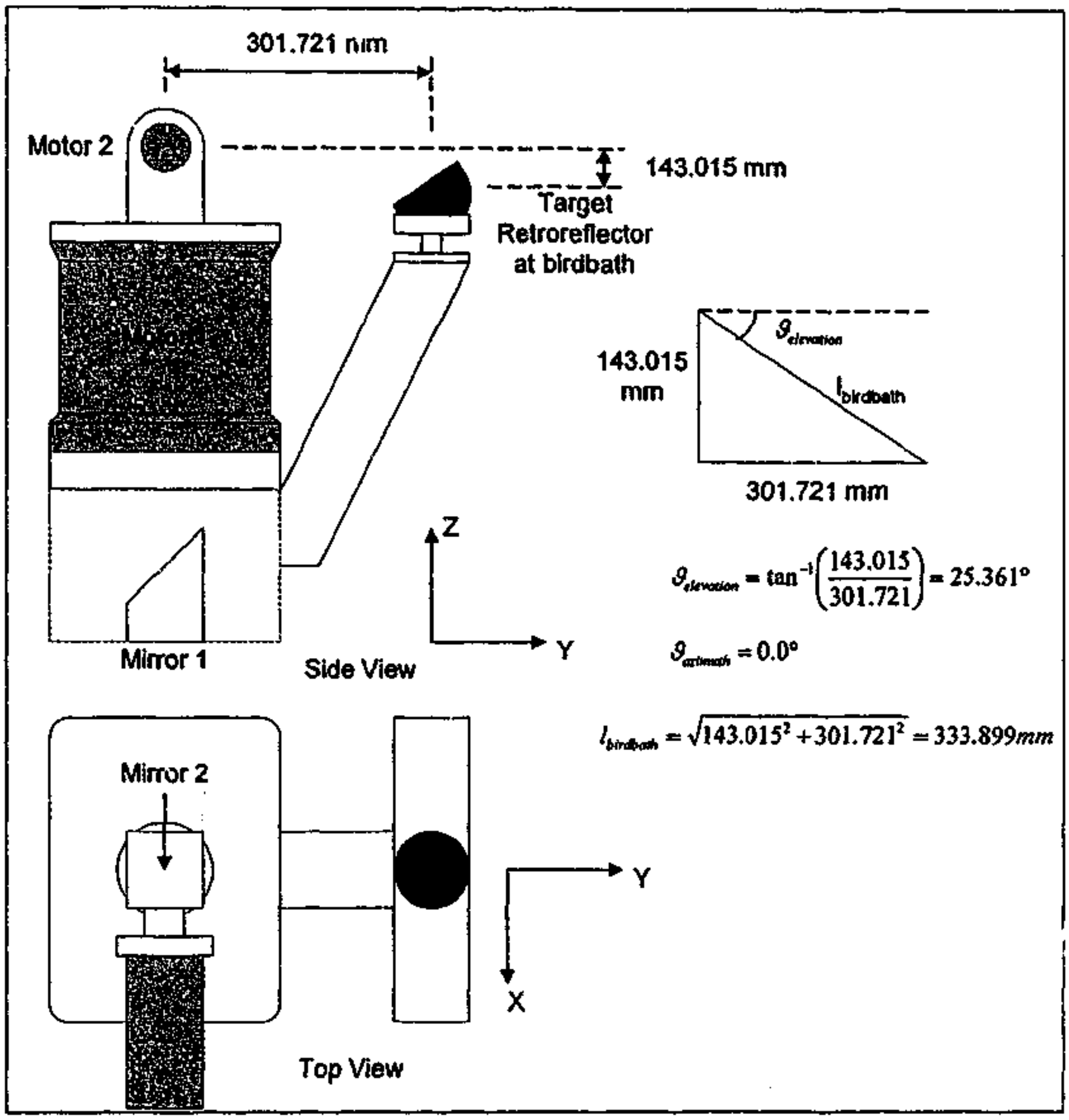

301.721 mm
Motor 2
143.015 mm
Target
Retroreflector
at birdbath
143.015
mm
$\vartheta_{elevation}$
$l_{birdbath}$
301.721 mm
Z
Y
$\vartheta_{elevation} = \tan^{-1}\left(\frac{143.015}{301.721}\right) = 25.361°$
$\vartheta_{azimuth} = 0.0°$
$l_{birdbath} = \sqrt{143.015^2 + 301.721^2} = 333.899mm$
Mirror 1
Side View
Mirror 2
Y
X
Top View

Figure 3.29: Schematic representation of beam steering mechanism

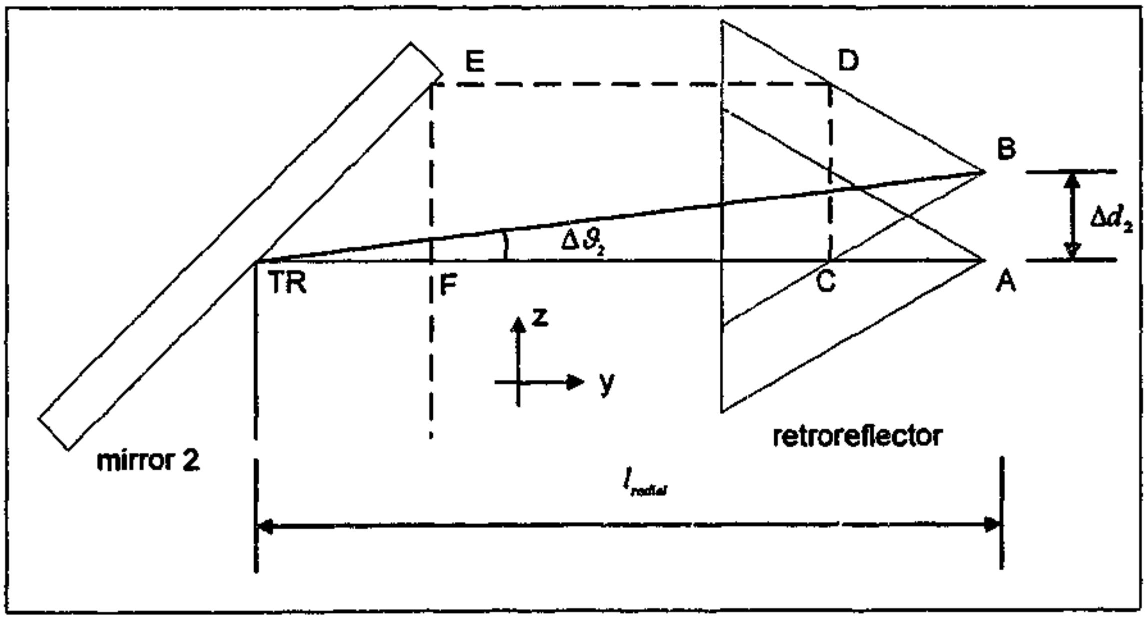


Figure 3.30: Angular rotation for Mirror 2 with target moving along LISM z-axis

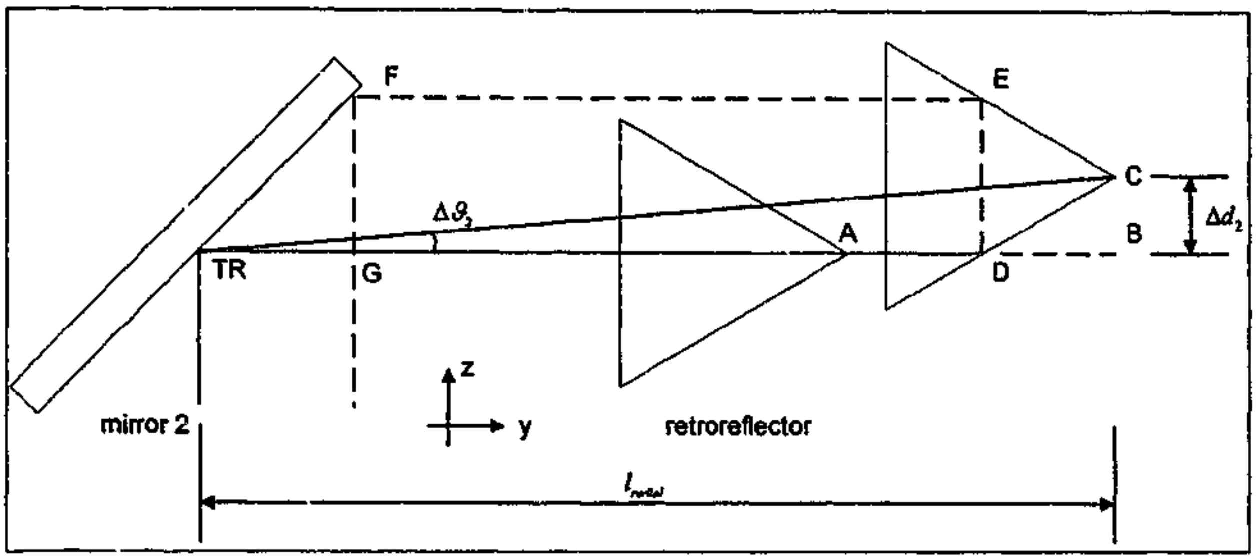


Figure 3.31: Angular rotation for Mirror 2 with target moving along LISM yz-plane

The factor of ½ is used because a motor rotation of 1° will result in a beam angular displacement of 2° for motor 2. A similar approach was taken for motor 1 and the commanded angular displacement for motor 1 $\Delta\vartheta_1$ is calculated from:

$$\Delta\vartheta_1 = \tan^{-1}\left(\frac{-\Delta d_1}{l_{radial}\cos 2\vartheta_{c2}}\right) \tag{3.14}$$

where ϑ_{c2} represents the current angle of motor 2 and Δd_1 represents the current target lateral offset for motor 1.

Fig. 3.31 shows another schematic representation of the beam propagation when the retroreflector is displaced along yz plane. There are now additional displacements along y-axis. The current laser interferometer radial distance recorded by the interferometer will be half of the distance of the path TR-D-E-F-G. This distance is equivalent the distance of TR-B (Appendix B). The rotation angle required for motor 2 and motor 1 can also be calculated using Eq. 3.13 and 3.14, respectively.

3.7.3 Target Lateral Offset Determination

Fig. 3.32 and 3.33 show the diagrams of beam steering mechanism when motor 1 is at 0° and at 90°, respectively. These figures show that when the target retroreflector is displaced along LISM z-axis, the beam offset detected by the PSD will be along the PSD y axis in Fig. 3.32 and PSD x-axis in Fig. 3.33. Beam offsets detected by the PSD have to be scaled by the following equations to determine the magnitude of Δd_1 and Δd_2 in Eqs 3.15 to 3.16. Δd_1 and Δd_2 are only dependent on the current angular displacement of motor 1.

$$\Delta d_1 = \frac{1}{2}\left(\text{PSD-x} \times \cos\vartheta_{c1} + \text{PSD-y} \times \sin\vartheta_{c1}\right) \tag{3.15}$$

$$\Delta d_2 = \frac{1}{2}\left(-\text{PSD-x} \times \sin\vartheta_{c1} + \text{PSD-y} \times \cos\vartheta_{c1}\right) \tag{3.16}$$

where PSD-x and PSD-y are the beam offsets detected by the PSD along the PSD's x- and y-axes, respectively and ϑ_{c1} represents the current angular displacement of motor 1.

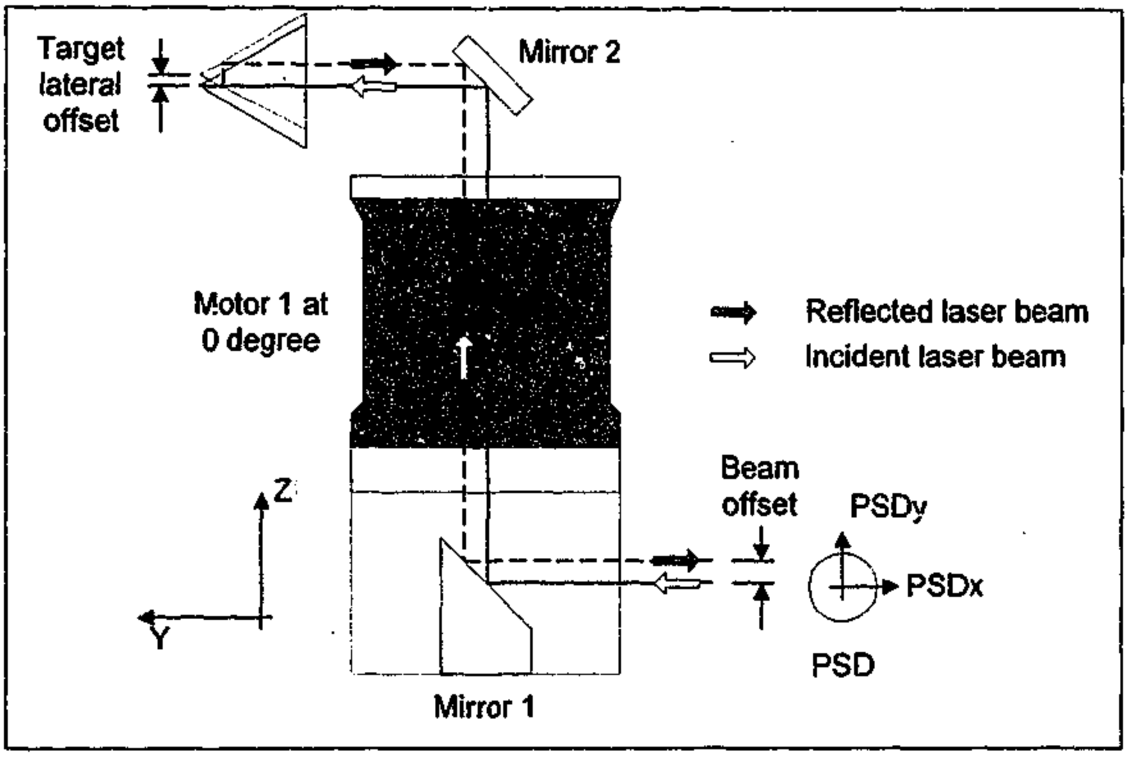


Figure 3.32: Beam steering mechanism when Motor 1 at 0 degree

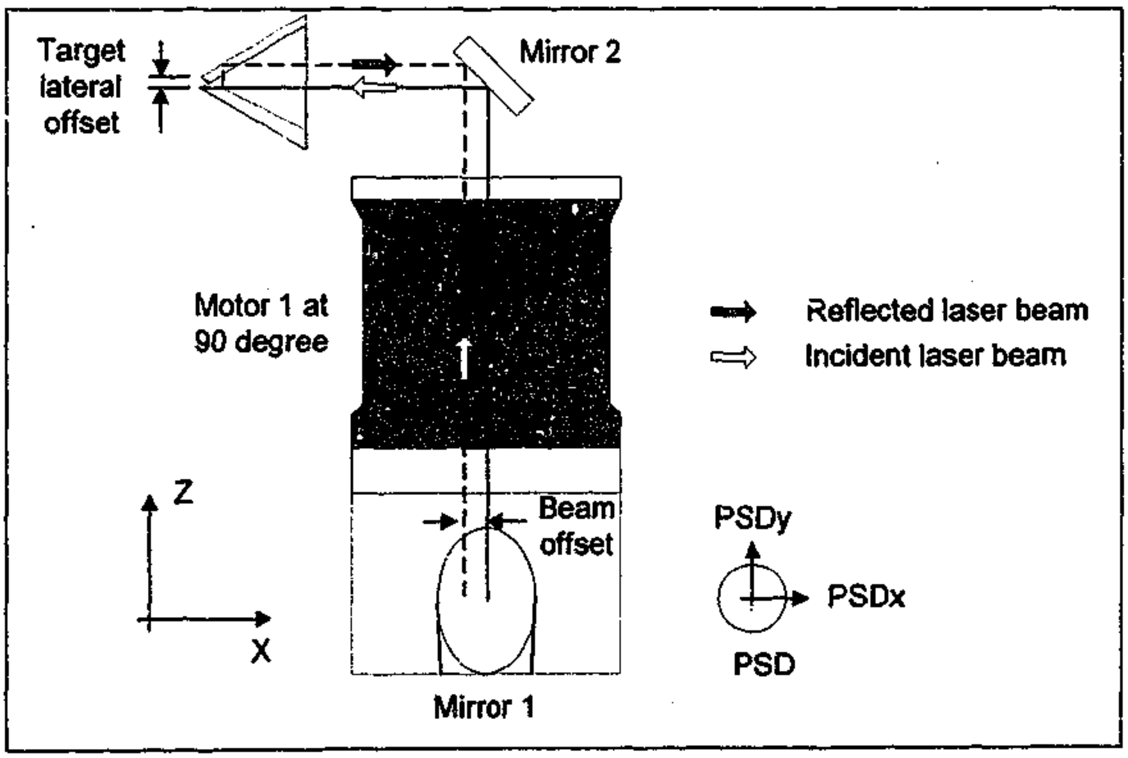


Figure 3.33: Beam steering mechanism when Motor 1 at 90 degree

3.7.4 Control Algorithm- Position Mode and Velocity Mode

In position mode, the rotational angular displacements calculated from Eqs. 3.13 and 3.14 are sent to the motor controller. The motors will then go through the specific motion profile to move by the commanded angular displacement. Since the commanded angular displacement cannot be changed on the fly in this particular controller, the motors have to come to a halt before the next motion command is executed. Fig. 3.34 shows the block diagram of this control approach. In velocity mode, the motors are commanded to run continuously where the velocity is controlled. Each motor's velocity could be changed on the fly. Thus, the motors do not have to stop and start between positional changes when tracking. The commanded velocities are computed so that the motors' motion does not exceed the desired angular displacement before the next velocity command. With an overall LISM update of 50 Hz, the commanded velocity can be calculated by the following equation:

$$\dot{\vartheta}_i = \frac{\Delta\vartheta_i}{40\times10^{-3}s} \tag{3.17}$$

where i is the motor number, $\Delta\vartheta_i$ is the angular displacement calculated using Eq. 3.13 and 3.14, and $\dot{\vartheta}_i$ is the commanded angular velocity for motor i. A period of 40 ms is used to account for the time required to accelerate and decelerate to the desired velocity. The control block diagram is as shown in Fig. 3.35.

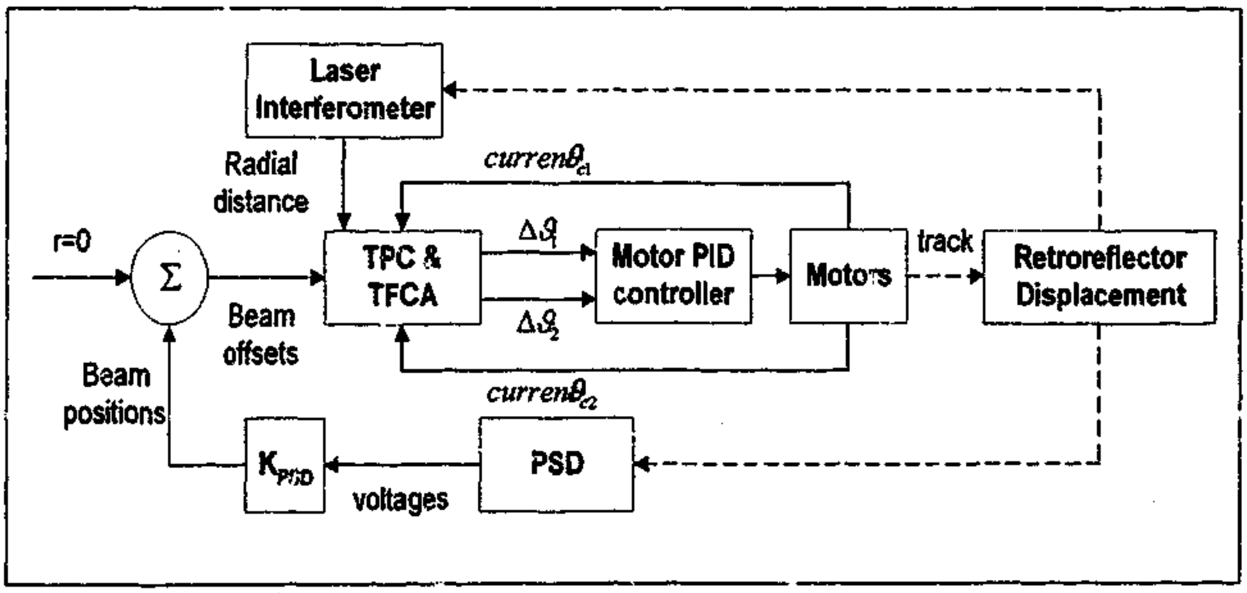


Figure 3.34: LISM position control block diagram

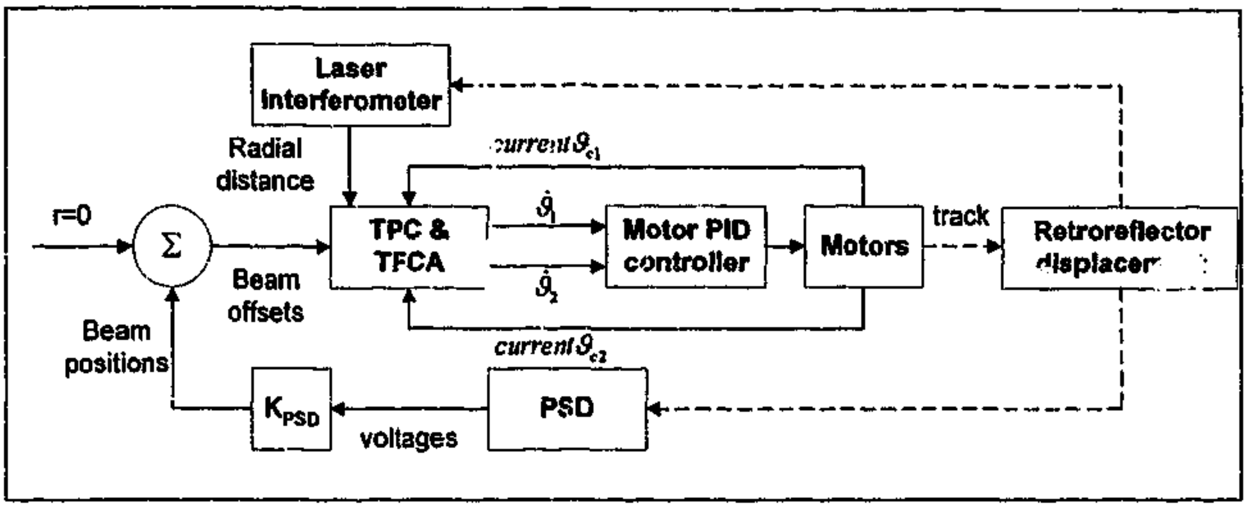


Figure 3.35: LISM velocity control block diagram

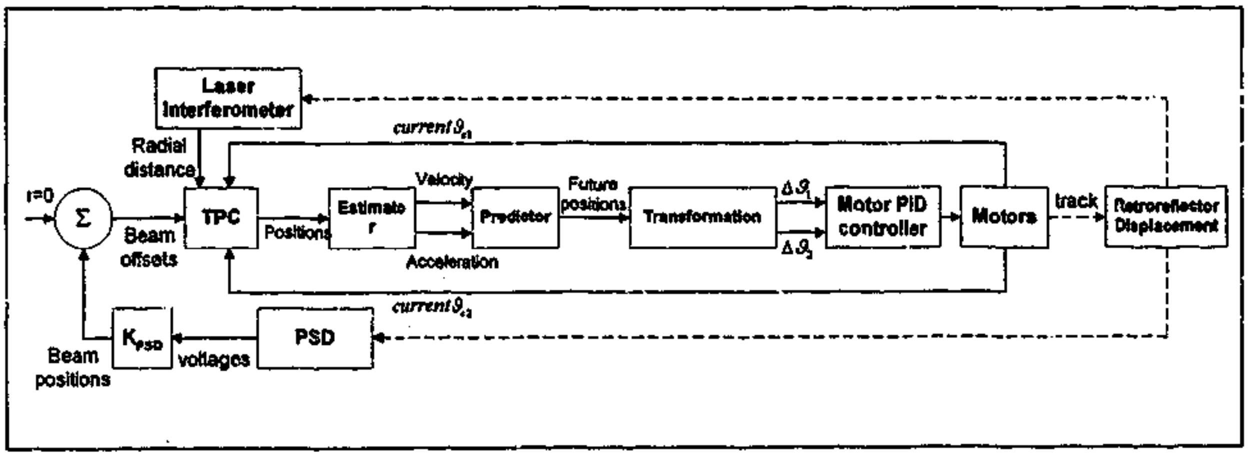


Figure 3.36: LISM predictive control block diagram

3.7.5 Predictive Control Algorithm

In order to improve the tracking speed of the target retroreflector, a predictive control algorithm has been implemented [36]. The proposed algorithm requires the position of the retroreflector in Cartesian space to be known during tracking so that the velocity and acceleration of the retroreflector can be predicted. This control algorithm comprises an estimator and a predictor. Fig. 3.36 shows the block diagram of LISM controller with the predictive control algorithm.

The TPC computes the position of the retroreflector based on the sensor's data from the sub-systems and sends these data to the estimator. The estimator uses the current and past retroreflector positions to estimate the velocity and acceleration of the target. This is undertaken using the following equations:

$$\dot{x}(k) = \frac{x(k) - x(k-1)}{t_u} \tag{3.18}$$

$$\ddot{x}(k) = \frac{\dot{x}(k) - \dot{x}(k-1)}{t_u} = \frac{x(k) - 2x(k-1) + x(k-2)}{t_u^{\ 2}} \tag{3.19}$$

where t_u is the LISM update interval, $x(k)$ is the actual position at time k, $\dot{x}(k)$ and $\ddot{x}(k)$ are the estimated velocity and acceleration, respectively. This method has the advantage of computational efficiency but suffers from sensitivity to measurement noise.

The predictor module predicts the next position of the target using the current measured position, estimated velocity and estimated acceleration. This prediction is based on the assumption that the states of the target cannot change drastically within a few sample intervals due to inertia. The next position of the target retroreflector is calculated using the following equation:

$$x(k+1) = x(k) + \dot{x}(k) \cdot t_u + \frac{1}{2}\ddot{x}(k) \cdot t_u^{\ 2} \tag{3.20}$$

where $x(k+1)$ is the predicted position at time k+1. The predicted positions are transformed to the future set point angular displacement of the motors.

To convert the incremental position information calculated from Eq. 3.20 into incremental angular displacement of the motors, the relationship between the current position of the centre of the retroreflector and the required angular displacement of the motors must first be established. Fig. 3.37 shows the schematics of the beam path for retroreflector moving along LISM yz-plane. From Fig. 3.37, l_{radial}, θ_{c2} and Δd_2 are the measurements obtained from the sub-systems. As shown in Fig. 3.37, there is an additional displacement of the laser beam when the laser beam is not pointing precisely at the centre of the target retroreflector. By using the laser displacement detected, together with the PSD measurements and the current angular displacement of the motors, the current z position $z_{current}$ of the retroreflector's centre can be obtained from:

$$z_{current} = z_{imaginary} + \frac{\Delta d_2}{\cos 2\vartheta_{c2}} \tag{3.21}$$

$z_{imaginery}$ is an imaginary z position directly on top or below $z_{current}$. This imaginary position can be determined by:

$$z_{imaginary} = l_{imaginary} \sin 2\vartheta_{c2} = (l_{radial} - \Delta d_1 \tan \vartheta_{c1} - \Delta d_2 \tan 2\vartheta_{c2}) \sin 2\vartheta_{c2} \tag{3.22}$$

where $l_{imaginary}$ is the radial distance from the tracking reference TR to the imaginary position and l_{radial} is the current radial distance recorded by the laser interferometer.

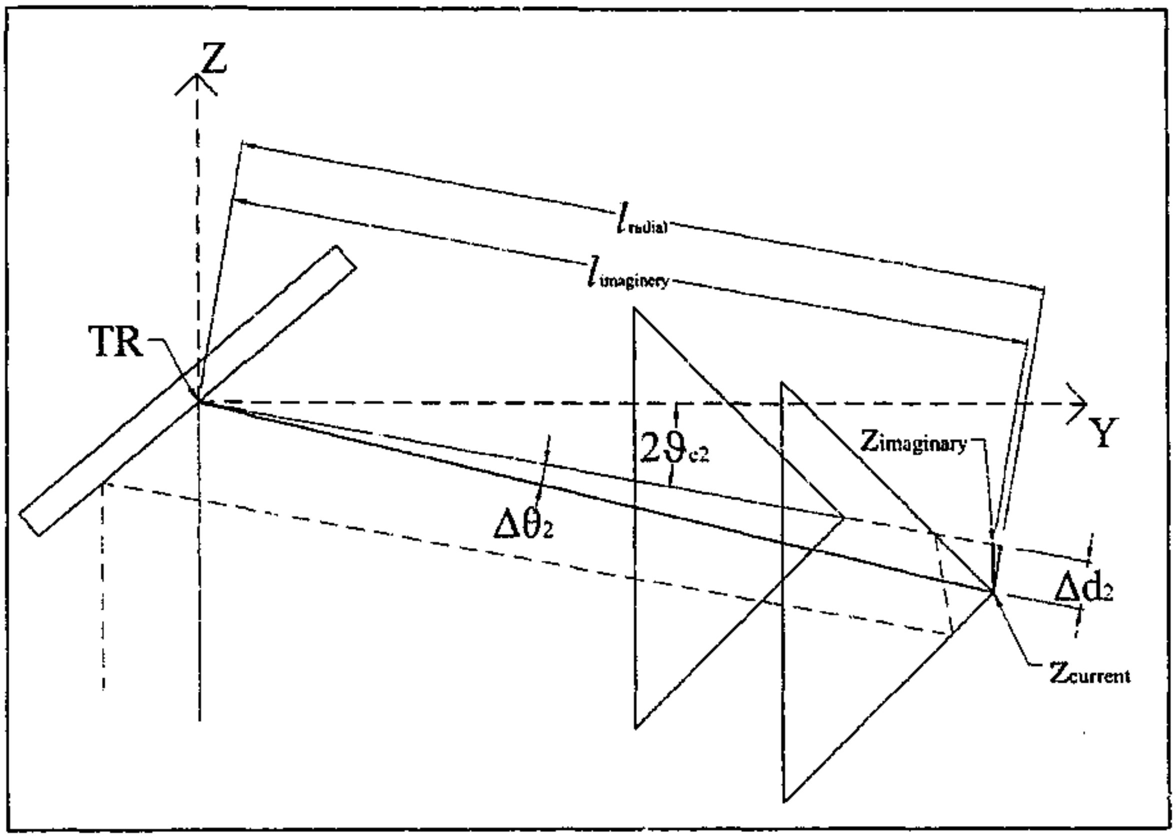


Figure 3.37: Laser beam path with target moving along LISM yz-plane

Substituting Eq. 3.21 into Eq. 3.13 gives:

$$\Delta\vartheta_2 = \frac{1}{2}\tan^{-1}\left(\frac{\Delta d_2}{l_{radial}}\right) = \frac{1}{2}\tan^{-1}\left[\frac{\left(z_{current} - z_{imaginery}\right)\cos 2\vartheta_{c2}}{l_{radial}}\right] \tag{3.23}$$

Fig. 3.38 shows the schematics of the beam path for the retroreflector moving along LISM xy-plane. Current x position of the retroreflector's centre $x_{current}$ can be obtained by:

$$x_{current} = x_{imaginary} + \frac{\Delta d_1}{\cos\vartheta_{c1}} \tag{3.24}$$

$$\begin{aligned} x_{imaginary} &= l_{imaginary-xy}\sin\vartheta_{c1} \\ &= (l_{radial} - \Delta d_1 \tan\vartheta_{c1} - \Delta d_2 \tan 2\vartheta_{c2})\cos 2\vartheta_{c2}\sin\vartheta_{c1} \end{aligned} \tag{3.25}$$

where $l_{imaginary\text{-}xy}$ is the radial distance from the tracking reference TR to the imaginary point on xy-plane.

By substituting Eq. 3.24, Eq. 3.14 can be written as:

$$\Delta\vartheta_1 = \tan^{-1}\left(\frac{-\Delta d_1}{l_{radial}\cos 2\vartheta_{c2}}\right) = \tan^{-1}\left[\frac{-\left(x_{current} - x_{imaginary}\right)\cos\vartheta_{c1}}{l_{radial}\cos 2\vartheta_{c2}}\right] \tag{3.26}$$

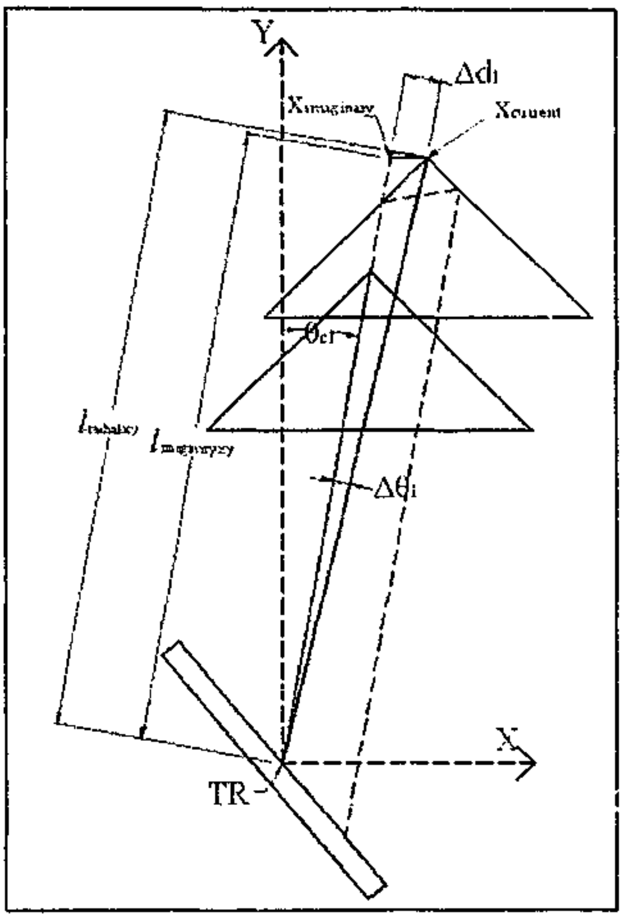


Figure 3.38: Laser beam path with target moving along LISM xy-plane

With the implementation of the predictive control algorithm, the current positions of the retroreflector's centre is replaced by the predicted positions. Therefore, Eq. 3.21 and 3.24 can be rewritten as follows:

$$\Delta\vartheta_2 = \frac{1}{2}\tan^{-1}\left[\frac{\left(z_{predict} - z_{imaginery}\right)\cos s\,\vartheta_{c2}}{l_{radial}}\right] \tag{3.27}$$

$$\Delta\vartheta_1 = \tan^{-1}\left[\frac{-\left(x_{predict} - x_{imaginary}\right)\cos\vartheta_{c1}}{l_{radial}\cos s\,\vartheta_{c2}}\right] \tag{3.28}$$

where $z_{predict}$ and $x_{predict}$ are the predicted z and predicted x positions, respectively.

For velocity control, the new commanded velocity is calculated by the following equation:

$$\dot{\vartheta}_i = \frac{\Delta\vartheta_i}{40\times10^{-3}} \tag{3.29}$$

where i is the motor number, $\Delta\vartheta_i$ is the angular displacement calculated using Eq. 3.27 and 3.28, and $\dot{\vartheta}_i$ is the commanded angular velocity for motor i.

In this approach, instead of only correcting the tracking offset, the LISM controller will rotate the beam steering mechanism to position the laser beam at the predicted position. When the beam is being directed to this position, the retroreflector would have moved and there will again be a tracking offset. The next tracking loop will take the new sub-systems' measurements to update the estimates of the velocity and acceleration. The velocity and acceleration is again used for the prediction of the following position. Tracking behaviour can be increased considerably, since knowledge about the future is used to minimise the tracking offset.

3.8 Experimental Investigation

Experiments were conducted to determine the behaviour of the control algorithms presented for the LISM apparatus. The criterion for evaluation of the behaviour and effectiveness of the methodology is the magnitude of the tracking offsets detected by the PSD. The lower the magnitude, the more effective the algorithm. In addition, the algorithms were also evaluated for their ability to follow the target running at different velocities. The working range of the LISM apparatus was also determined. The experiments conducted involved the tracking of the motion of the target retroreflector mounted on a Motoman SK120 robot manipulator. The robot was commanded to move along a 3D path at different velocities. The results are presented in Figs. 3.39 to 3.44.

Figs. 3.39 to 3.41 show the PSD readings for the Target Following Control Algorithm (TFCA) in position mode. From Figs. 3.39 to 3.41, it can be observed that there is a significant amount of oscillation in the PSD readings. These oscillations are due to the noise in the PSD. The TFCA picks up the tracking offsets due to target displacement with noise values and perform corrections by driving the corresponding motors, thus causing under or over correction of the tracking offsets. The other reason is due to the overshooting of the motors. The tracking offsets detected by the PSD due to overshoot were corrected when the robot moves towards the next position. This will direct the beam away from the centre of the retroreflector in the direction opposite to the travel of the target, creating a larger tracking offset. This leads to the swinging of the beam about the centre of the retroreflector. Other possible reasons are the coupling effect between both motors, vibration of the mirror due to tumbling of motors and hysteresis effect [47].

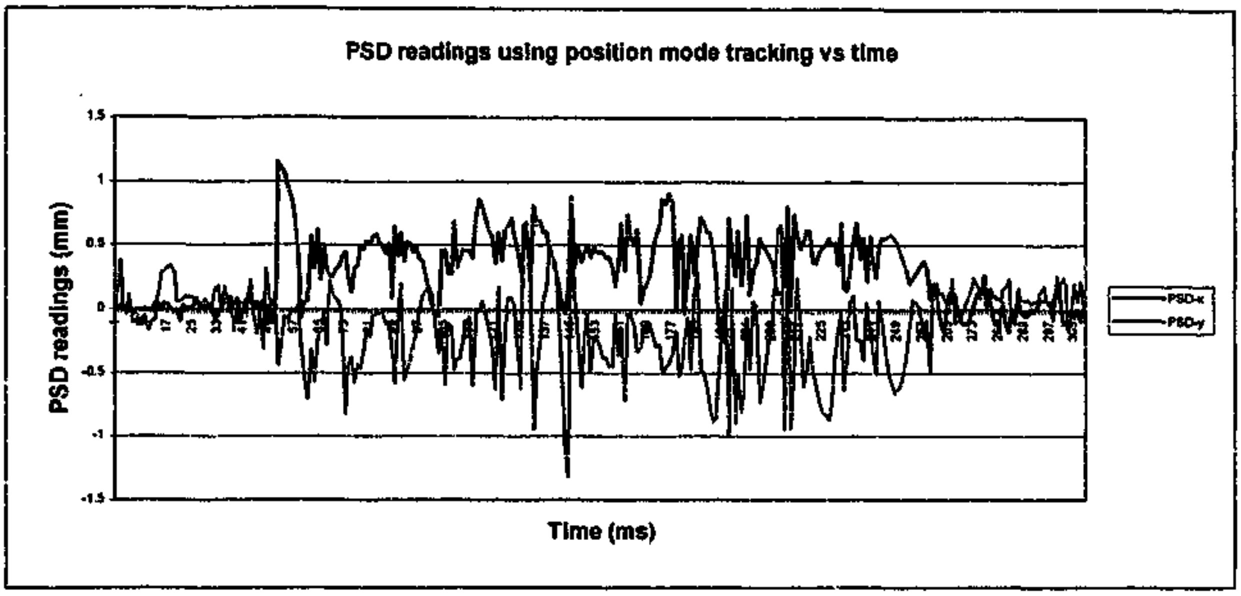


Figure 3.39: PSD readings using position mode tracking with robot moving at velocity of 10mm/s and acceleration of 10mm/s² with target at 0.6m

Mean PSD-x = 0.4mm
PSD-x oscillation = 1.15mm, -0.26mm
Mear ⁚SD-y = -0.4mm
PSD-y oscillation = 0.67mm, -1.32mm

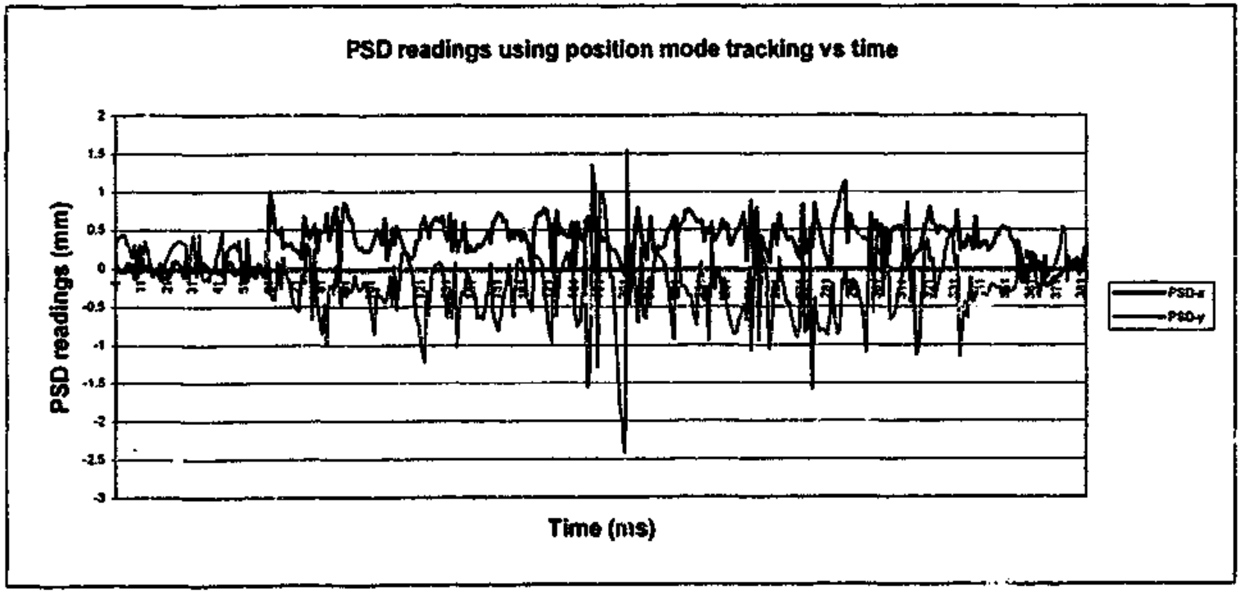


Figure 3.40: PSD readings using position mode tracking with robot moving at velocity of 10mm/s and acceleration of 10mm/s² with target at 1m

Mean PSD-x = 0.4mm
PSD-x oscillation = 1.14mm, -0.14mm
Mean PSD-y = -0.4mm
PSD-y oscillation = 1.54mm, -2.41mm

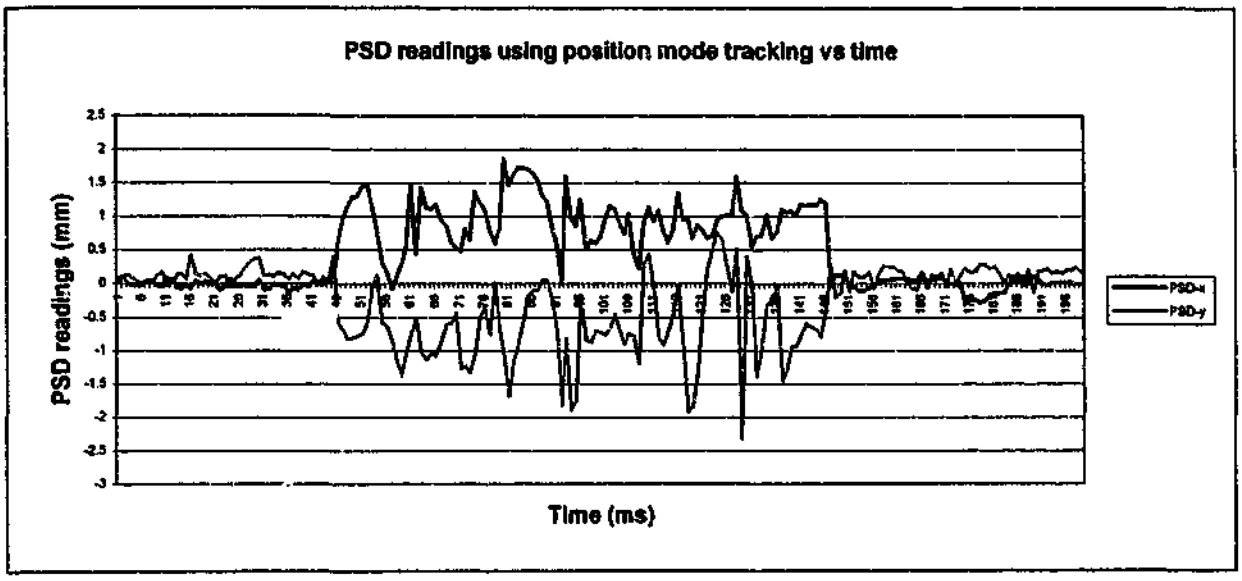


Figure 3.41: PSD readings using position mode tracking with robot moving at velocity of 20mm/s and acceleration of 10mm/s^2 with target at 0.6m

Mean PSD-x = 0.8mm
PSD-x oscillation = 1.87mm, -0.1mm
Mean PSD-y = -0.8mm
PSD-y oscillation = 0.77mm, -2.32mm

By comparing Fig. 3.39 with Fig. 3.40, it can be noticed that the magnitude of oscillations for PSD-y increased with the distance of the target from the LISM apparatus. This is caused by the inaccuracy of Motor 2, where there are undetected position losses in the open-loop stepper system. The position loss was shown in Table 3.1 to be higher with the target located further away from the LISM apparatus. Moreover, it can be observed from Fig. 3.22 that there is a delay of about 300ms before motor 2 reaches the commanded angular displacement. Within this period, the tracking offset detected is further corrected, causing over correction. Moreover, the mean offset should be smaller with the increase in target distance. However, no decrease in mean offset can be observed. This is due to the slow update of the TFCA algorithm. With a target moving at 10mm/s, the beam offset created along the PSD x- and y-axes at every update interval of 20ms is 2 x 10 x 20 x 10^{-3} = 0.4 mm. Another possible reason is due to imperfections in the retroreflector used. The laser beam was not being reflected through an angle of 180°, rather with a slight angular deviation [27]. Therefore, there could be a slight change in path of the beam and increases the tracking offset detected by the PSD. This offset due to beam angular deviation increases with the target distance.

By comparing Fig. 3.39 with Fig. 3.41, it can be observed that the magnitude of the mean offset and fluctuations for both PSD readings increased with the target moving at a higher velocity. This is again due to the limitation of the update interval. The laser beam is not able to keep up with the faster target motion and thus produces larger tracking offsets. The slow update interval also limits the maximum target acceleration that can be followed to 10 mm/s^2. The acceleration can be increased by having a higher update rate. With more updates, more corrections of the tracking offsets can be made before the laser beam gets beyond the 3mm mark.

From the experimental results for position mode tracking, it can be concluded that the TFCA in position mode can follow a target moving at a velocity of 10mm/s and acceleration of 10mm/s^2. The maximum working range is 1000mm from the intersection point between the horizontal and vertical rotational axes of the beam steering mechanism. The velocity can be increased to 20mm/s with a working range of 600mm.

Figs. 3.42 to 3.44 show the results of the PSD readings for the TFCA in velocity mode. From these figures, it can be observed that the oscillation of the PSD-x readings are higher compared to results obtained using position mode. This is due to the low update rate of the TFCA algorithm. The motors are commanded to move continuously based on the velocity calculated from the tracking offsets and the update rate. The effect of noise in the PSD readings introduces error in the calculated motors' angular velocities. The laser beam is moving at the commanded velocity until the next update rate, thus causing the wavering of the beam about the centre of the retroreflector.

One way of reducing the magnitude of oscillation is by increasing the period used in Eq. 3.17. However, this will lower the commanded velocity and thus reduces the permissible velocity of the target to be followed. Fig. 3.45 shows the results obtained using a period of 80ms for motor 1. The maximum target velocity that can be followed is 10mm/s at the range of 600mm, compared to 20mm/s at the same range shown in Fig. 3.44. The mean for the PSD-x has increased from 0.4mm to 1.2mm, indicating a slower tracking velocity. The range of oscillation has decreased from 2.72mm to 0.82mm. PSD-y has a similar magnitude of fluctuation compared to position mode because of slow response. The slower response of this motor can be further demonstrated by observing the higher mean for the PSD-y compared to Fig. 3.39. The response can be improved by decreasing the period used in Eq. 3.17. However, this will result in larger fluctuations due to noise. Fig. 3.45 shows

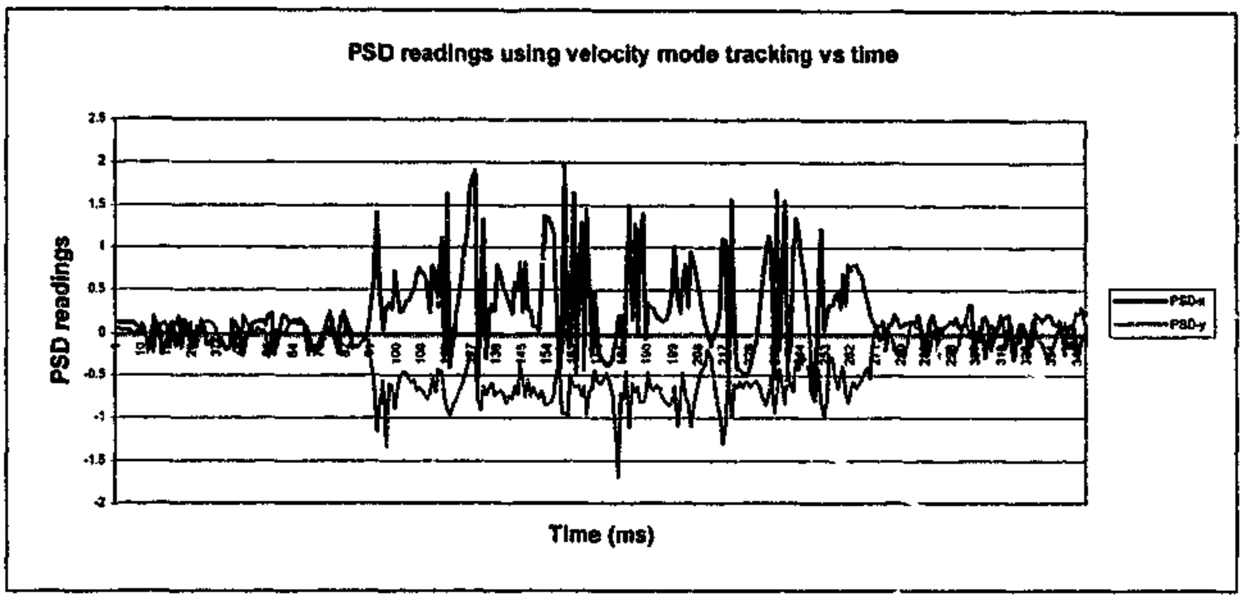


Figure 3.42: PSD readings using velocity mode tracking with robot moving at velocity of 10mm/ s and acceleration of 10mm/ s^2 with target at 0.6m

Mean PSD-x = 0.4mm
PSD-x oscillation = 1.97mm, -0.75mm
Mean PSD-y = -0.7mm
PSD-y oscillation = -0.25mm, -1.69mm

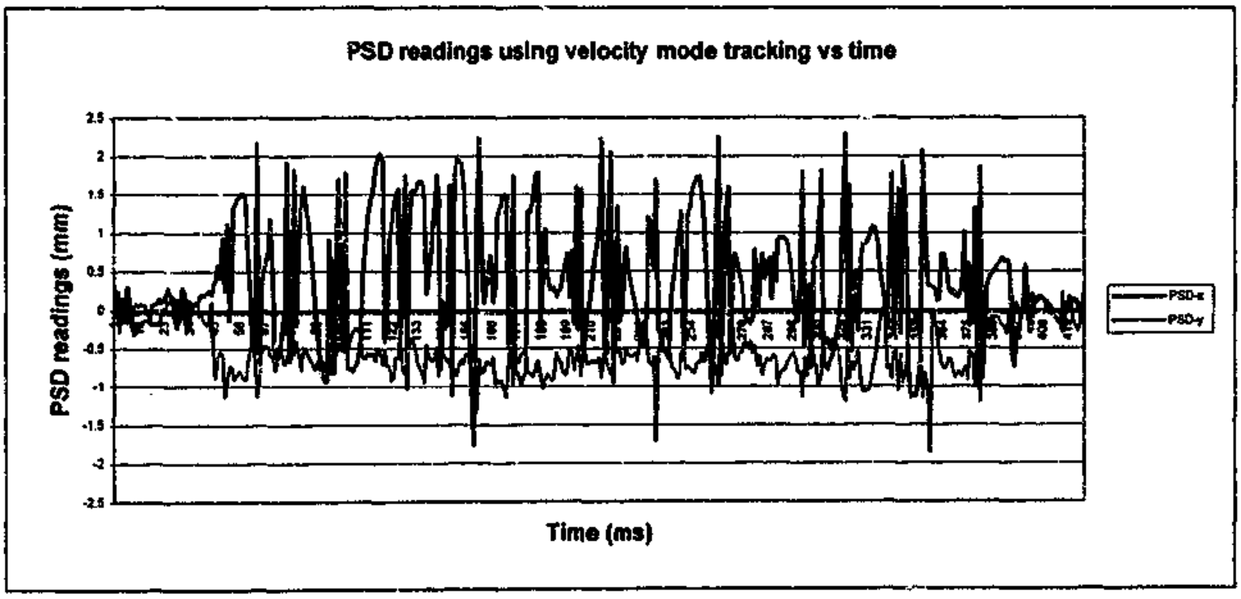


Figure 3.43: PSD readings using velocity mode tracking with robot moving at velocity of 10mm/ s and acceleration of 10mm / s^2 with target at 0.8m

Mean PSD-x = 0.5 mm
PSD-x oscillation = 2.29mm, -1.75mm
Mean PSD-y = -0.75mm
PSD-y oscillation = 0.05mm, -1.86mm

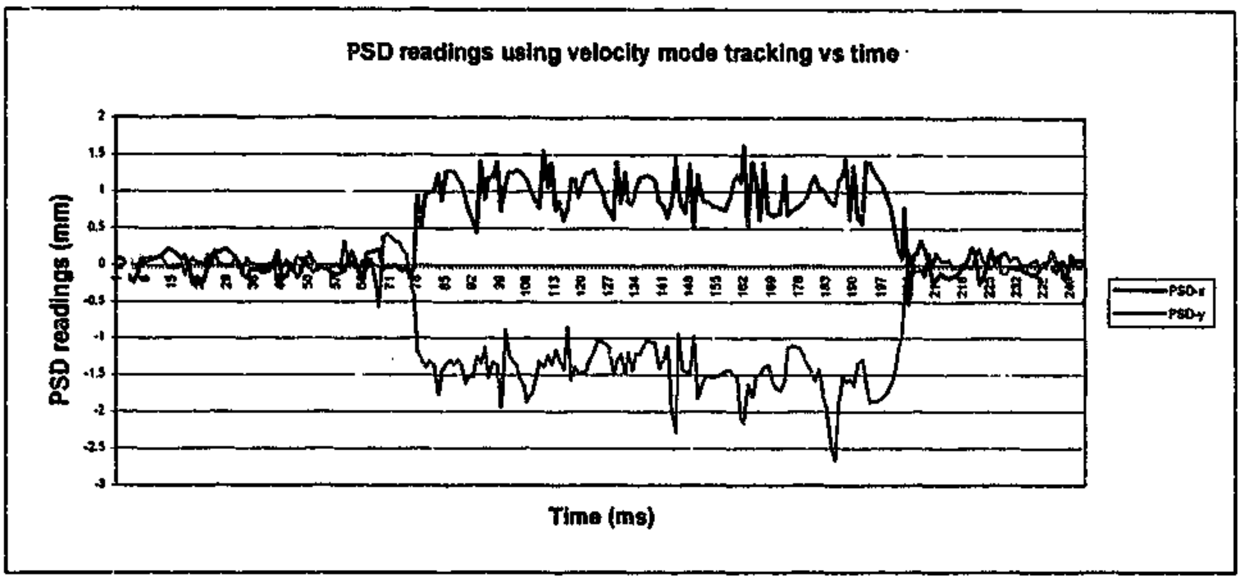


Figure 3.44: PSD readings using velocity mode tracking with robot moving at velocity of 20mm/ s and acceleration of 10mm / s^2 with target at 0.6m

Mean PSD-x = 0.8mm
PSD-x oscillation = 1.62mm, 0.5mm
Mean PSD-y = -1.5mm
PSD-y oscillation = -0.8mm, -2.66mm

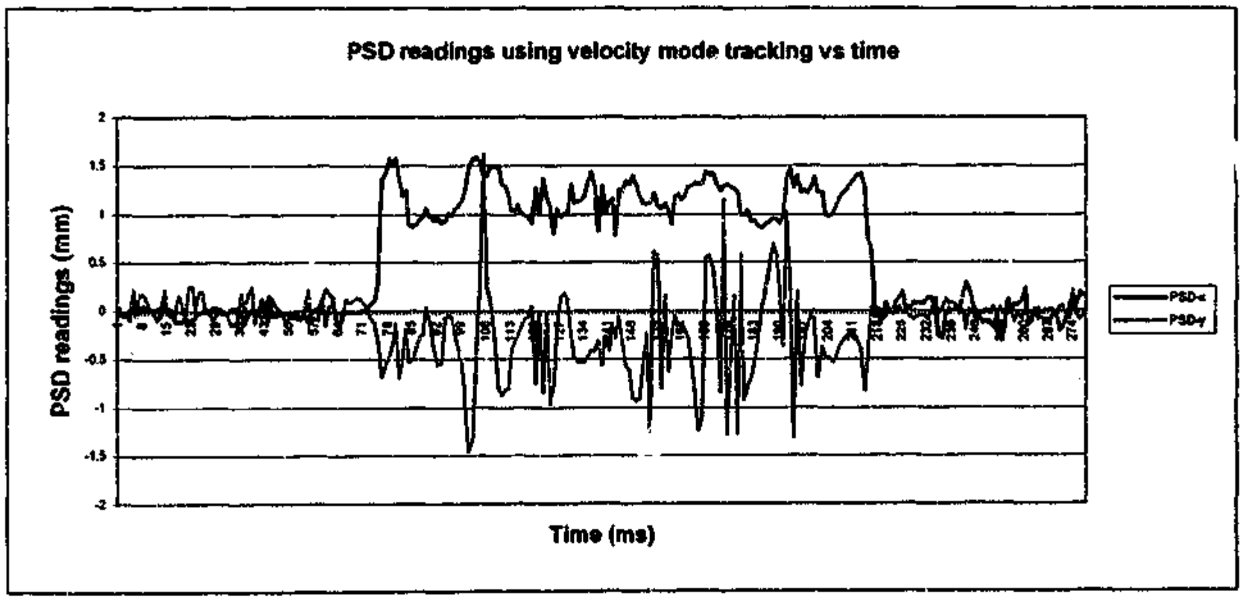


Figure 3.45: PSD readings using velocity mode tracking with robot moving at velocity of 10mm/ s and acceleration of 10mm / s^2 with target at 0.6m using different update period

Mean PSD-x = 1.2mm
PSD-x oscillation = 1.62mm, 0.8mm
Mean PSD-y = -0.4mm
PSD-y oscillation = 1.6mm, -1.5mm

the results obtained using a period of 20ms for motor 2. The mean error for PSD-y decreased from 0.7mm to 0.25 mm and the range of oscillation for PSD-y increased from 1.44mm to 3.1mm.

From Figs. 3.42 to 3.44, it can be concluded that the TFCA in velocity mode can follow a target moving at a velocity of 10mm/s and at an acceleration of 10mm/s^2. The maximum working range is 800mm from the intersection point between the horizontal and vertical rotational axes of the beam steering mechanism.

Figs. 3.46 and 3.47 show the results of the PSD readings for the TFCA in predictive position and predictive velocity mode, respectively. From the figures, it can be observed that there is no improvement in tracking performance. In fact, the maximum working range has been reduced to 500mm from the intersection point between the horizontal and vertical rotational axes of the beam steering mechanism. This is due to the large amount of oscillation in the PSD readings, which results from the error in the prediction of the target position. There are errors in the predicted positions mainly caused by the fact that the control algorithm is not updating quickly enough. Due to the slow update, the assumption that the states of the target cannot change drastically within a few sample intervals is invalid.

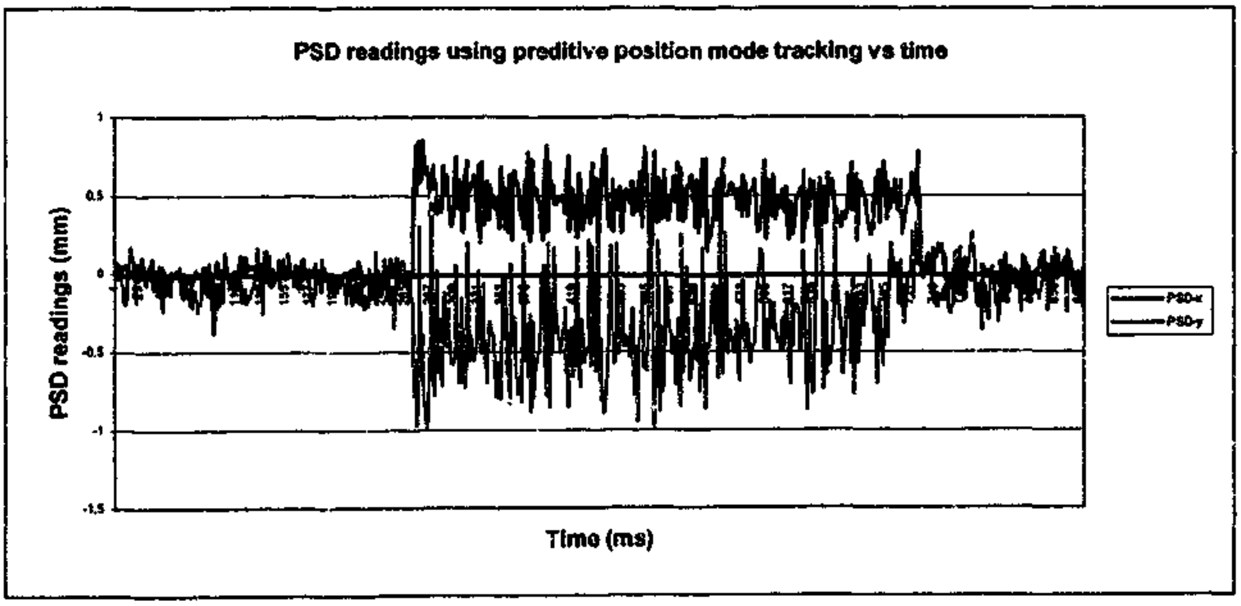


Figure 3.46: PSD readings using predictive position mode tracking with robot moving at velocity of 10mm/s and acceleration of 10mm/s^2 with target at 0.5m

Mean PSD-x = 0.5mm
PSD-x oscillation = 0.8mm, 0.2mm
Mean PSD-y = -0.5mm
PSD-y oscillation = 0.6mm, -1.0mm

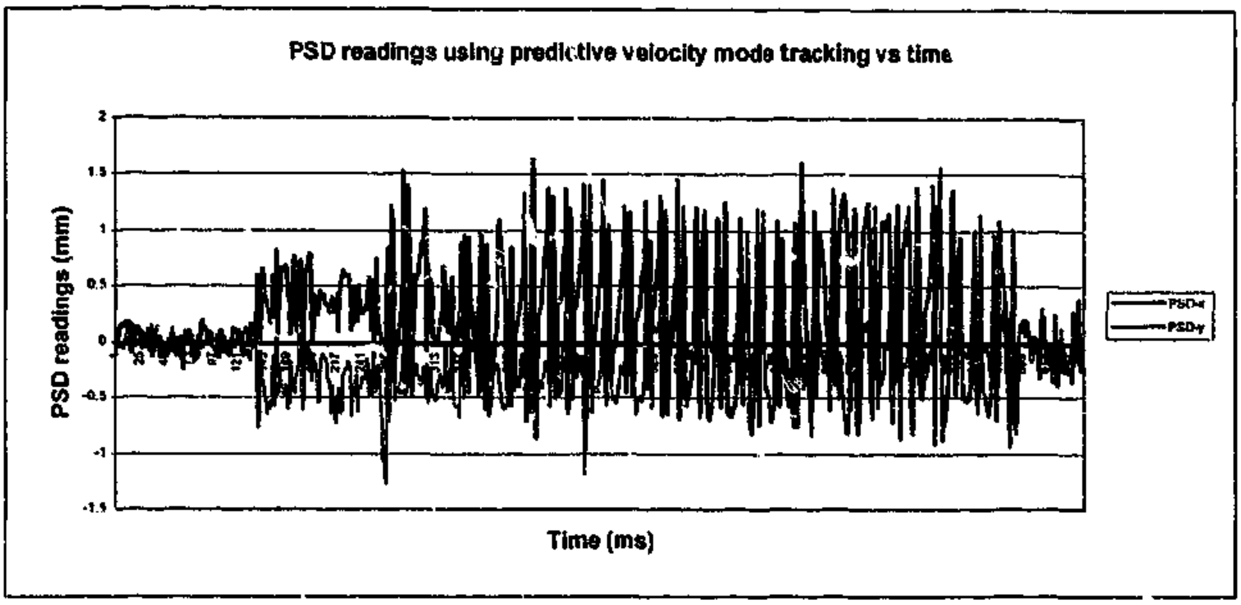


Figure 3.47: PSD readings using predictive velocity mode tracking with robot moving at velocity of $10mm/s$ and acceleration of $10mm/s^2$ with target at 0.5m

Mean PSD-x = 0.5mm
PSD-x oscillation = 1.6mm, -1.0mm
Mean PSD-y = -0.4mm
PSD-y oscillation = 0.8mm, -1.3mm

Another source of the error in the predicted positions is the noise in the position measurement. As stated in Section 3.7.5, the estimator module used to estimate the velocity and acceleration of the target retroreflector is sensitive to measurement noise. Inaccurate velocity and acceleration estimation will affect the accuracy of the predicted position of the target retroreflector calculated using Eq. 3.20.

3.9 Recommendations for Possible Improvements

To improve the target following performance, the following improvements may be considered:

1) Lower PSD noise - A lower noise in the PSD can reduce the oscillations of the beam. A target moving at a higher velocity can be followed with lower beam oscillations because the beam will stay within the ±3mm range. This can be achieved by implementing a better PSD with lower dark current and/or better quality components in the signal conditioning device. A more efficient filtering technique can also be implemented to reduce the noise in the PSD measurements.
2) Higher update rate - With a higher update rate, a higher target velocity and acceleration can be obtained. Appropriate corrections of the tracking offset can be

performed before the beam moves beyond the ±3mm range. A higher update rate can also allow the implementation of the predictive control algorithm. The update rate can be increased by the utilisation of more efficient hardware components and the implementation of the software architecture described in Section 3.6.3. A faster computer can also be considered.

3) Higher laser beam power and larger beam diameter - Fig. 3.48 shows the percentage of beam power emitted from laser head onto the PSD. With a PSD responsivity of 0.42 A/W, the photo-currents on the PSD is 7.35μA. The existing PSD has a dark current of 0.25μA. This contributes to a signal to noise (S/N) ratio of 29.4. Higher beam power will improve the signal to noise ratio of the PSD measurements. With a higher beam power, together with a larger beam diameter, the maximum beam offset permitted to maintain interference can also be increased. As a result, a higher target velocity and acceleration can be followed.
4) Dynamic modelling of the motors - A good dynamic model can improve the tracking performance by reducing the coupling effect of both motors and vibration.
5) Motor with higher resolution and better performance – A closed-loop configuration can be established for motor 2 with higher motor resolution. Higher motor resolution will also allow for the tracking of a target located further away from the LISM apparatus. More steps are required to correct a small tracking offset, thus decreasing the position loss. A better motor performance can also reduce the position loss, vibration, overshoot and response time, resulting in faster and smoother tracking of the target.

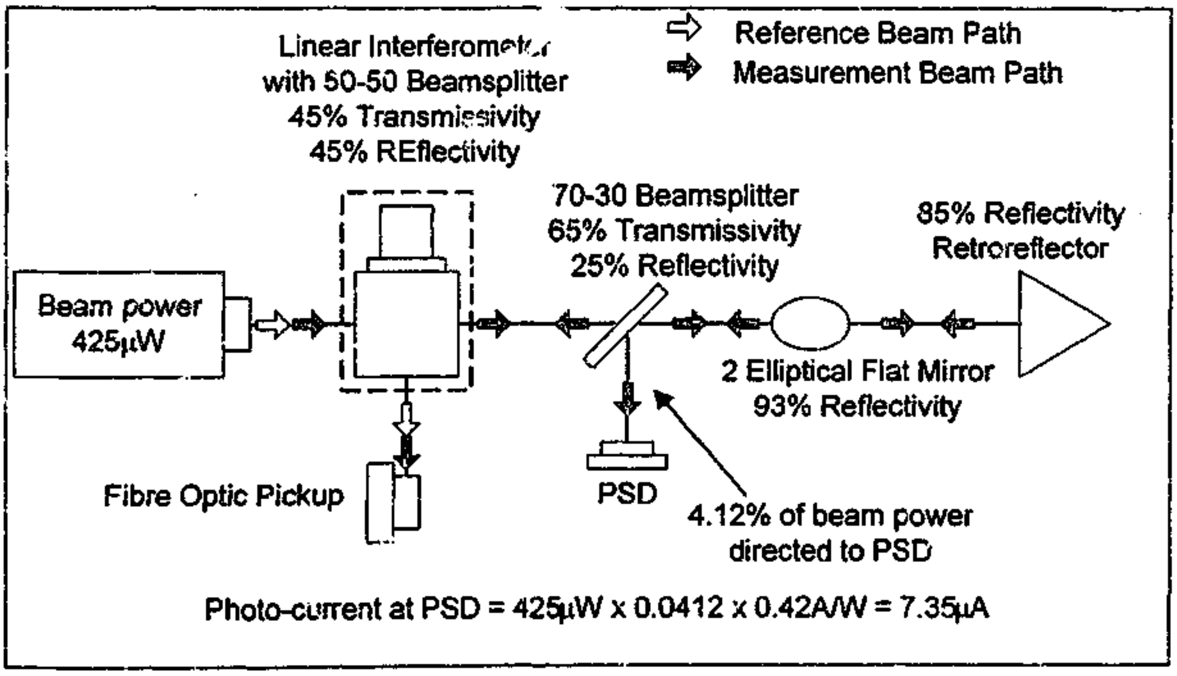


Figure 3.48: Optics efficiency of LISM apparatus

Table 3.3: Experimental LISM apparatus performance specifications

Performance Criteria	Specifications
Samp'ing rate	50 Hz
Maximum velocity of target	20 mm/s
Maximum acceleration of target	10 mm/s^2
Maximum target distance	1000 mm

3.10 Summary

In this chapter, the physical make-up, specification, functionality, and the measurement and analysis techniques employed for each sub-system implemented in the experimental apparatus have been presented. The software architecture of the control software has also been presented. The target following control algorithms used in this apparatus have been developed. The equations responsible for the calculation of the motors' angular displacements based on the measurements obtained from the sub-systems had been derived. Further, the effect of the control algorithms used has been examined and the limitations analysed. The performance specification for the LISM apparatus is summarised in Table 3.3. Position tracking mode is selected in the experimental LISM apparatus as this control mode provides for a higher tracking velocity, more stable tracking and a larger range. The experimental LISM apparatus is used for the development of the proposed closed-loop control and laser interferometry-based guidance of robot manipulators in the later chapters. Next chapter provides the study of orientation measurement technique, together with the experimental verification of the technique.

Chapter 4

Experimental Investigation of the Proposed Orientation Measurement Methodology

4.1 Introduction

Orientation measurement in laser interferometry-based measurement using the CCD camera and the CCD array based methods has been studied [11, 12, 13, 14]. The range of measurement using these methods is limited due to the small incident acceptance angle of the retroreflector. Moreover, the centre of the retroreflector has to be covered by the laser beam for a valid measurement to be acquired. Multi-laser interferometry-based technique that utilises triangulation has also been studied [19]. The set-up cost for this method will be higher compared to the single laser-interferometry based technique. Another approach to orientation measurement in LISM is the dual PSD-based orientation measurement [56, 57, 58]. This chapter presents the experimental investigation of the dual PSD-based orientation measurement methodology. The physical set-up and control algorithm of this methodology are the centre of focus in this chapter.

4.2 Principle of Dual PSD-based Orientation Measurement Methodology

Dual PSD-based orientation measurement methodology utilises a specially developed Gimbal unit as shown in Fig. 4.1. The Gimbal unit consists of two position sensitive diodes (PSDs), a beamsplitter, and a retroreflector mounted on the intersection of the 2 axes of rotations. When the laser beam from the experimental LISM apparatus passes through the 70-30 beamspiitter on the Gimbal unit, it splits into two beams. 30% of this beam power is incident on the second PSD and the remaining will be directed towards the retroreflector. The reflected beam will travel back to the beamsplitter, parallel to the incident beam. At the beamsplitter, 30% of the laser beam will again be directed to the third PSD and the remaining laser beam will return to the beam LISM apparatus. Orientation of the retroreflector with respect to the laser beam is calculated from the beam positions measurement along the x- and y-axes of the two PSDs and the geometry of the Gimbal unit. A photo of the experimental set-up is shown in Fig. 4.2.

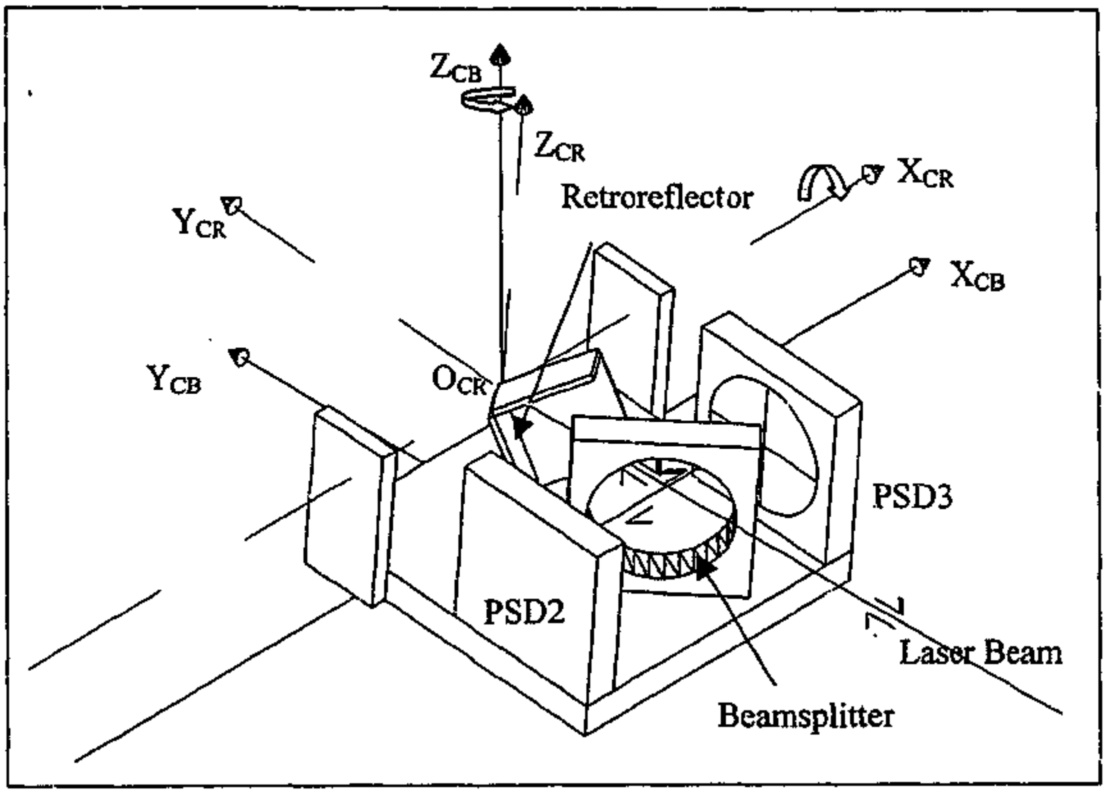


Figure 4.1: Gimbal unit assembly

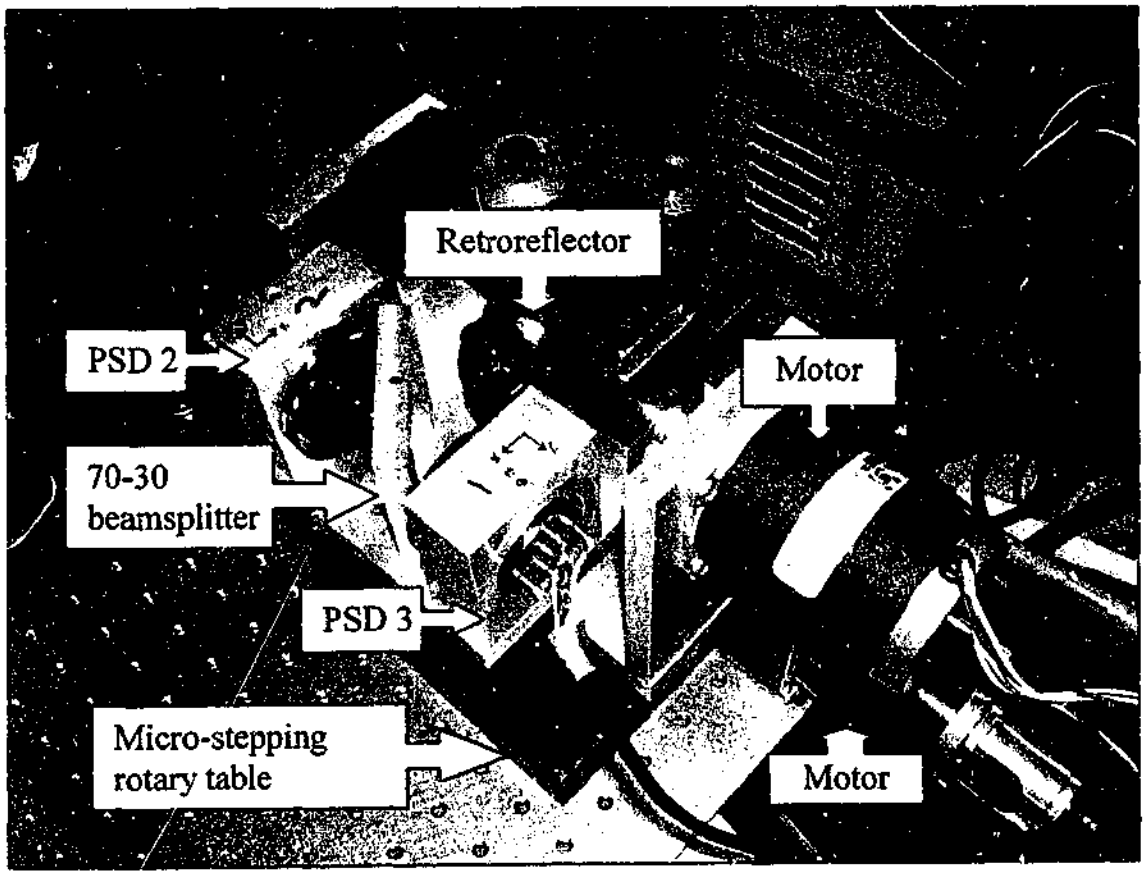


Figure 4.2: Orientation measurement experimental set-up

The experimental set-up consists of Gimbal unit mounted on a micro-stepping rotary table with a gear ratio of 45:1. A motor with resolution of 51200 steps per revolution is used to drive the table, providing rotation about the axis Z_{CR} shown in Fig. 4.1. This gives a maximum resolution of 2304000 steps per revolution. Another motor with a resolution of 50800 is used to provide rotation about the axis X_{CR} shown in Fig. 4.1. These motors are used to rotate the Gimbal unit (and thus the retroreflector) about the centre of the retroreflector based on the orientation calculated. The retroreflector will therefore always face the direction of the incoming laser beam. This improves the measuring range of the LISM apparatus.

4.3 Position Sensing Detector

The PSDs used are of lateral effect detector type as described in Section 3.4. Another set of calibrations had to be performed because different PSDs and sampling channels are employed. Moreover, it was observed that there is a significant amount of reflection from the surface of the PSDs. The reflection from PSD2 will cause corruption to the data sampled by PSD3 and vice versa. Quarter wave plates and polarisers are placed in front of both PSDs to impede reflection from the surface of one PSD to another. This again had to be taken into account during the calibration process. The software program for orientation measurement will refer to the calibration graph when the voltage values are detected to convert these into corresponding displacements. Figs. 4.3 and 4.4 show the graphical relationships between the position of the laser beam and the voltage difference detected by the x- and y-axes of both PSDs. A third order polynomial is used to represent the relationship as follows:

$$\text{PSD2-x} = 3.6791 \times \text{VRatioX}^3 + 0.1048 \times \text{VRatioX}^2 + 5.8582 \times \text{VRatioX} \qquad (4.1)$$

$$\text{PSD2-y} = 3.2779 \times \text{VRatioY}^3 + 0.2419 \times \text{VRatioY}^2 + 6.2109 \times \text{VRatioY} \qquad (4.2)$$

$$\text{PSD3-x} = 2.6272 \times \text{VRatioX}^3 + 0.1193 \times \text{VRatioX}^2 + 6.64 \times \text{VRatioX} \qquad (4.3)$$

$$\text{PSD3-y} = 2.5995 \times \text{VRatioY}^3 + 0.0682 \times \text{VRatioY}^2 + 6.6592 \times \text{VRatioY} \qquad (4.4)$$

where VRatioX and VRatioY are voltage ratio along the corresponding PSD's x- and y-axes, respectively, PSD2-x and PSD2-y are calculated beam positions along PSD2's x- and y-axes, respectively, PSD3-x and PSD3-y are calculated beam positions along PSD3's x- and y-axes, respectively.

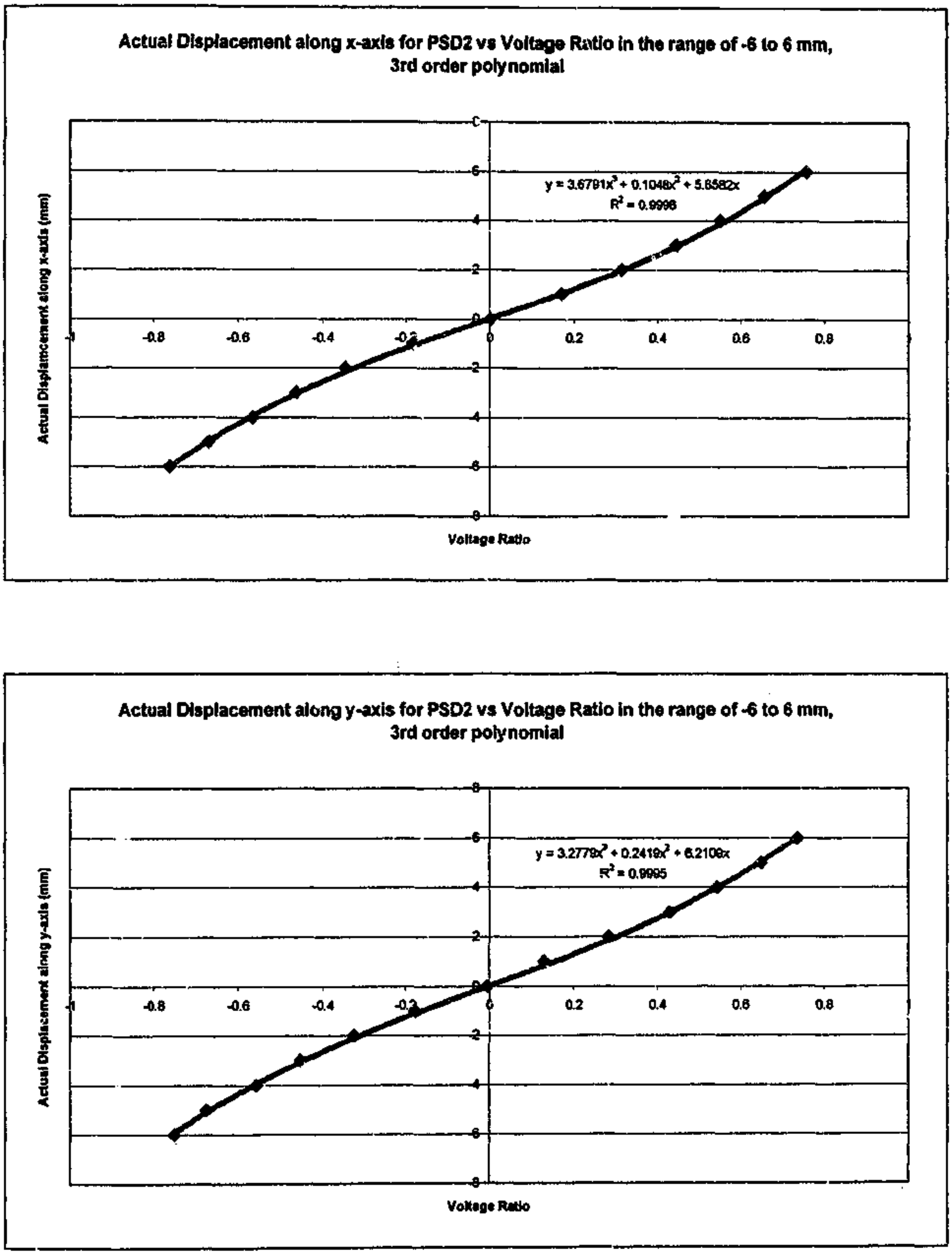


Figure 4.3: Calibration plot for PSD 2 in the range of –6 to 6 mm along x- and y-axes using 3rd order polynomial

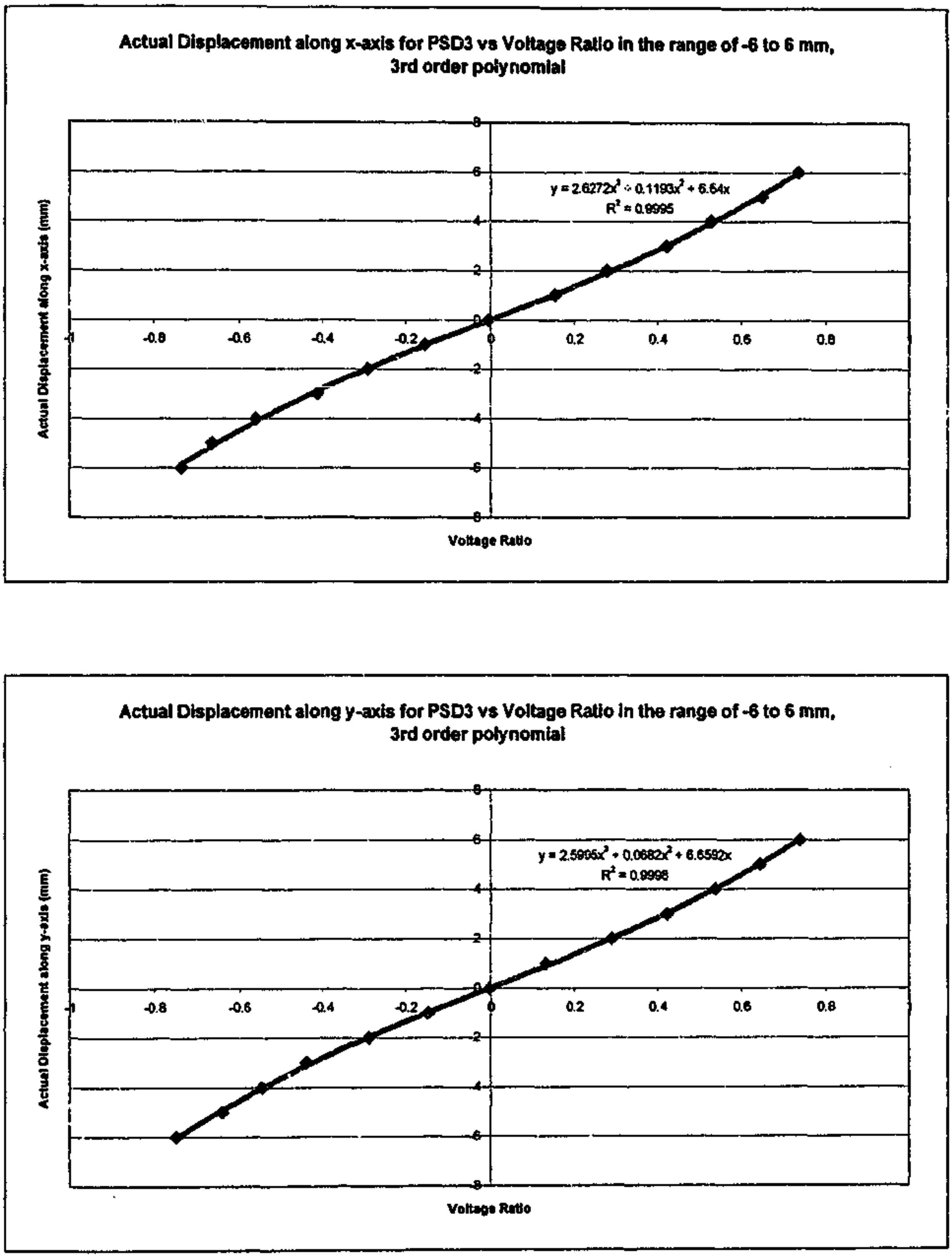


Figure 4.4: Calibration plot for PSD 3 in the range of –6 to 6 mm along x- and y-axes using 3rd order polynomial

4.4 Orientation Measurement Formulation

Fig. 4.5 shows a top view of the optical arrangement on the Gimbal unit. The governing equations to determine the orientation of the retroreflector (co-ordinate system *CR*) relative to the laser beam (co-ordinate system *CL*) are thus as follow [56, 57, 58]:

$$^{CL}T_{CR} = Rot(z,\varphi)Rot(y,\rho)Rot(x,\phi) \tag{4.5}$$

where ρ, ϕ, and φ are the roll, pitch and yaw angles, respectively.

From Fig. 4.5, it can be seen from the properties of the triangle that:

$$\varphi = \tan^{-1}\frac{D_{3x} - D_{2x}}{2l_1 - l_2 + l_3} \tag{4.6}$$

The pitch angle ϕ can be calculated by the following equation:

$$\phi = \tan^{-1}\frac{(D_{2y} + D_{3y})\cos\varphi}{2l_1 - l_2 + l_3} \tag{4.7}$$

where l_1 is the distance between the centres of the beamsplitter and the retroreflector, l_2 and l_3 are the distances between the centres of the beamsplitter and PSD2 and PSD3 respectively, D_{2x} and D_{3x} are the detected positions of the laser beam on PSD2 and PSD3 along the x-axis, respectively, D_{2y} and D_{3y} are the detected positions of the laser beam on PSD2 and PSD3 along the y-axis, respectively.

The roll angle, ρ, is calculated from the point of intersection between the laser beam and the xz-plane of the co-ordinate system *CR2*. This is carried out using the following equations:

$$B_{1x} = -D_{2x} + (l_2 - l_1)\tan\varphi \tag{4.8}$$

$$B_{1z} = D_{2y} - (l_2 - l_1)\frac{\tan\phi}{\cos\varphi} \tag{4.9}$$

where B_{1x} and B_{1z} are x and z co-ordinates, respectively, of the point of intersection between the laser beam and the xz-plane of the co-ordinate system *CR*.

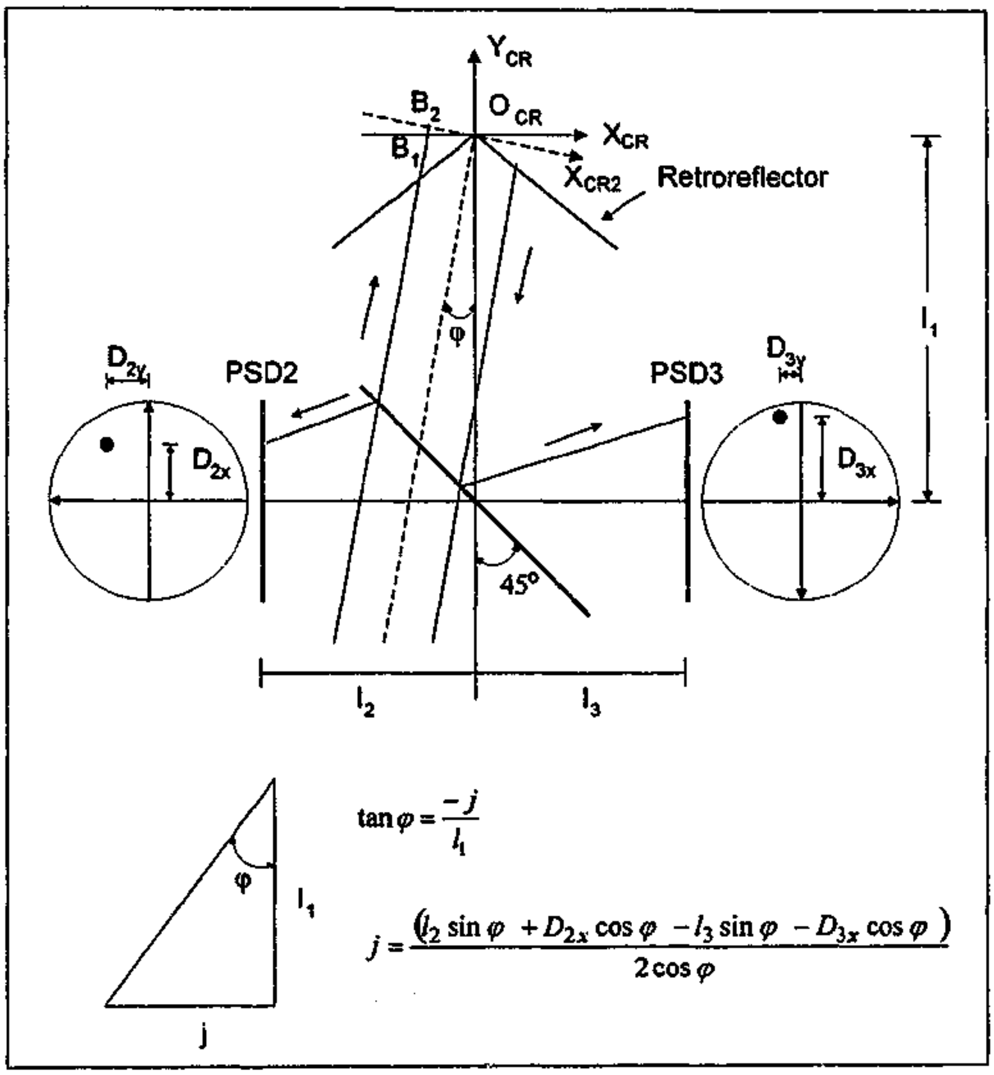


Figure 4.5: Top view of Gimbal unit

The co-ordinate system *CR2* is found from the transformation as follows:

$$
\begin{aligned}
{}^{CR}C_{CR2} &= Rot(z,\varphi)Rot(x,\phi) \\
&= \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \\
&= \begin{bmatrix} \cos\varphi & -\sin\varphi\cos\phi & \sin\varphi\sin\phi \\ \sin\varphi & \cos\varphi\cos\phi & -\cos\varphi\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}
\end{aligned}
\tag{4.10}
$$

The point of intersection between the laser beam and the xz-plane of the co-ordinate system *CR2* is therefore as follows:

$$B_{2x} = B_{1x}\cos\varphi + B_{1z}\sin\varphi\sin\phi \tag{4.11}$$

$$B_{2z} = B_{1z}\cos\phi \tag{4.12}$$

where B_{2x} and B_{2z} are x and z co-ordinates, respectively, of the point of intersection between the laser beam and the xz-plane of the co-ordinate system CR2.

The roll angle, ρ, is determined by:

$$\rho = \tan^{-1}\frac{B_{2z}}{B_{2x}} \tag{4.13}$$

4.5 Experimental Results

Preliminary experiments were performed in order to observe the capability of the Orientation Measurement Formulations (OMF) in determining the orientation of the Gimbal unit with respect to the laser beam. To acquire the pitch ϕ and yaw φ angles, the Gimbal unit was rotated about the horizontal and vertical rotation axes, individually. The orientations calculated by Eqs. 4.6 and 4.7 and the encoders' readings were recorded. The results are shown in Figs. 4.6 and 4.7.

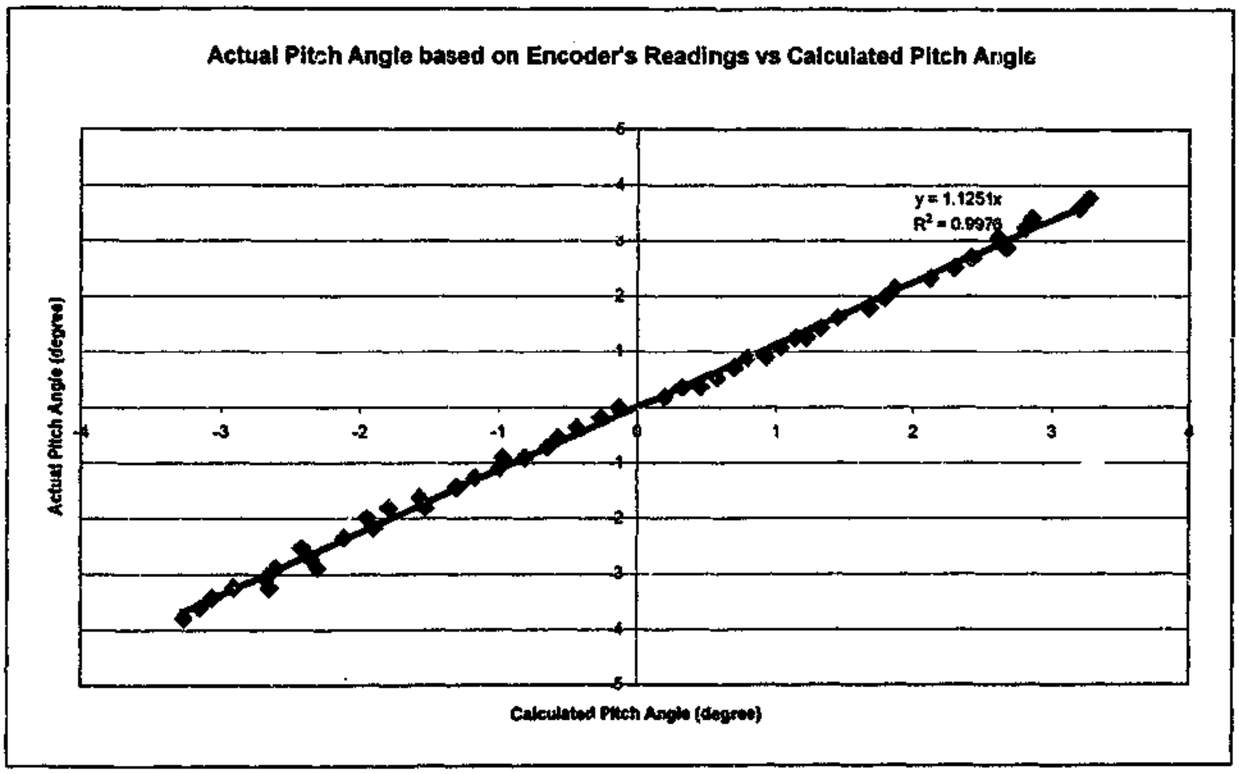


Figure 4.6: Actual pitch angle based on encoder's readings vs pitch angle calculated using OMF

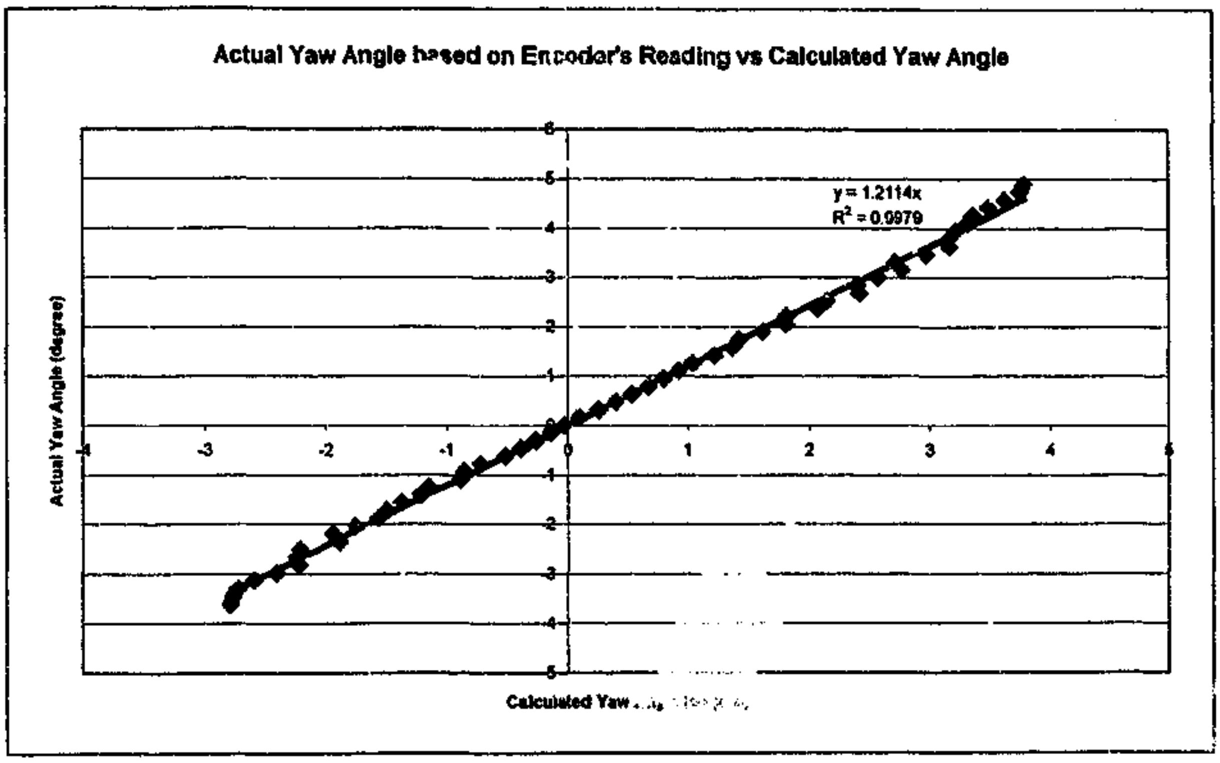


Figure 4.7: Actual yaw angle based on encoder's readings vs yaw angle calculated using OMF

Based on Figs. 4.6 and 4.7, the actual angles measured based on the encoders' readings can be related to the calculated angles by the following relationships:

$$\phi_{actual} = 1.1251 \times \phi_{OMF} \tag{4.14}$$

$$\varphi_{actual} = 1.2114 \times \varphi_{OMF} \tag{4.15}$$

where ϕ_{actual} and φ_{actual} are the actual pitch and yaw angles, respectively, ϕ_{OMF} and φ_{OMF} are the pitch and yaw angles calculated using OMF, respectively. By applying Eqs. 4.6, 4.7, 4.14 and 4.15, the errors between the calculated and actual angles are shown in Figs 4.8 and 4.9. Based on these figures, it can be concluded that the error in orientation calculated using OMF is 0.3° (0.005 rad) for pitch angle and 0.2° (0.0035 rad) for yaw angle. These errors are predominantly due to the noise in the PSDs and inaccurate geometric parameters used. Further, it can be observed that the magnitude of errors grows with larger angles. This is because as the angles of the retroreflector increase relative to the laser beam, the laser beam is being directed further away from the centre of the PSDs. Non-uniformity and non-linearity of the PSDs introduce errors in the PSD measurements and thus create higher deviations in the angles calculated. An error of 0.1° (0.0017 rad) can be obtained within the ±1° range. The maximum range of angular displacements for the retroreflector relative to the laser beam is ±3°. The incident laser beam will move out of the PSD range above ±3° and thus no measurements can be taken.

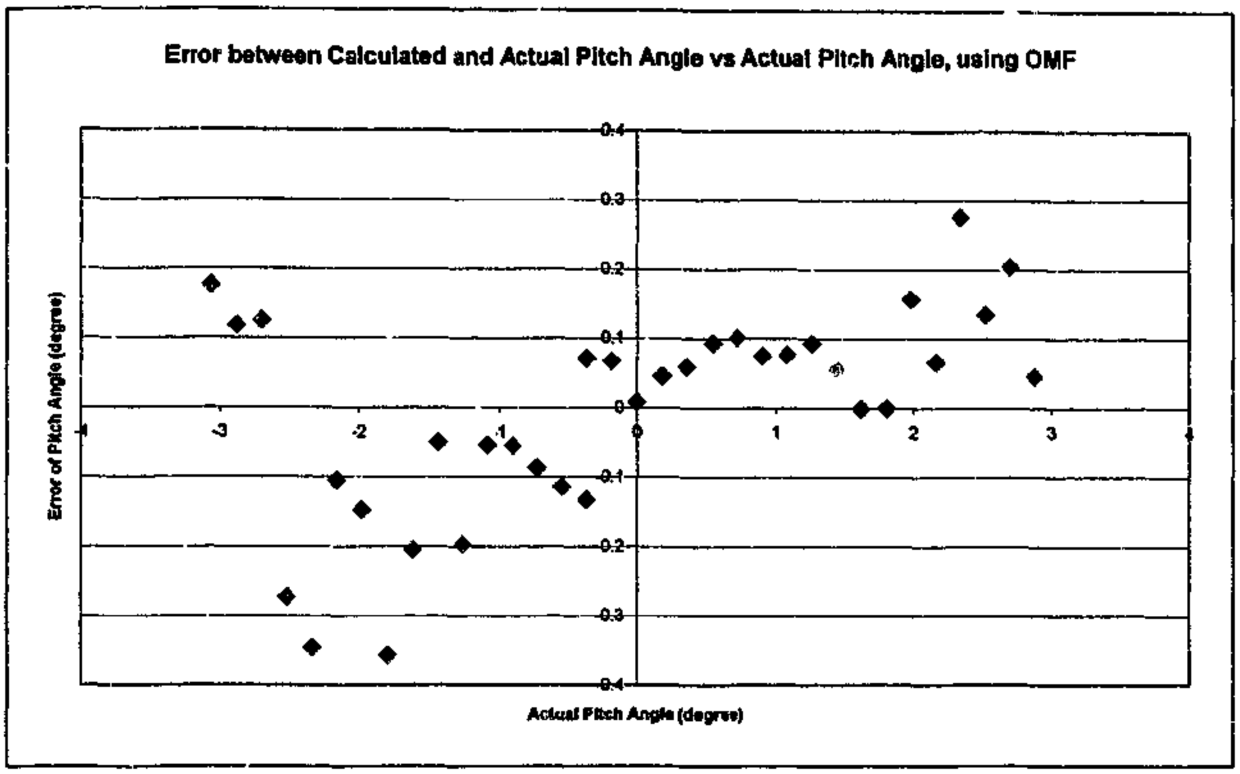


Figure 4.8: Error of calculated pitch angle using OMF vs actual pitch angle

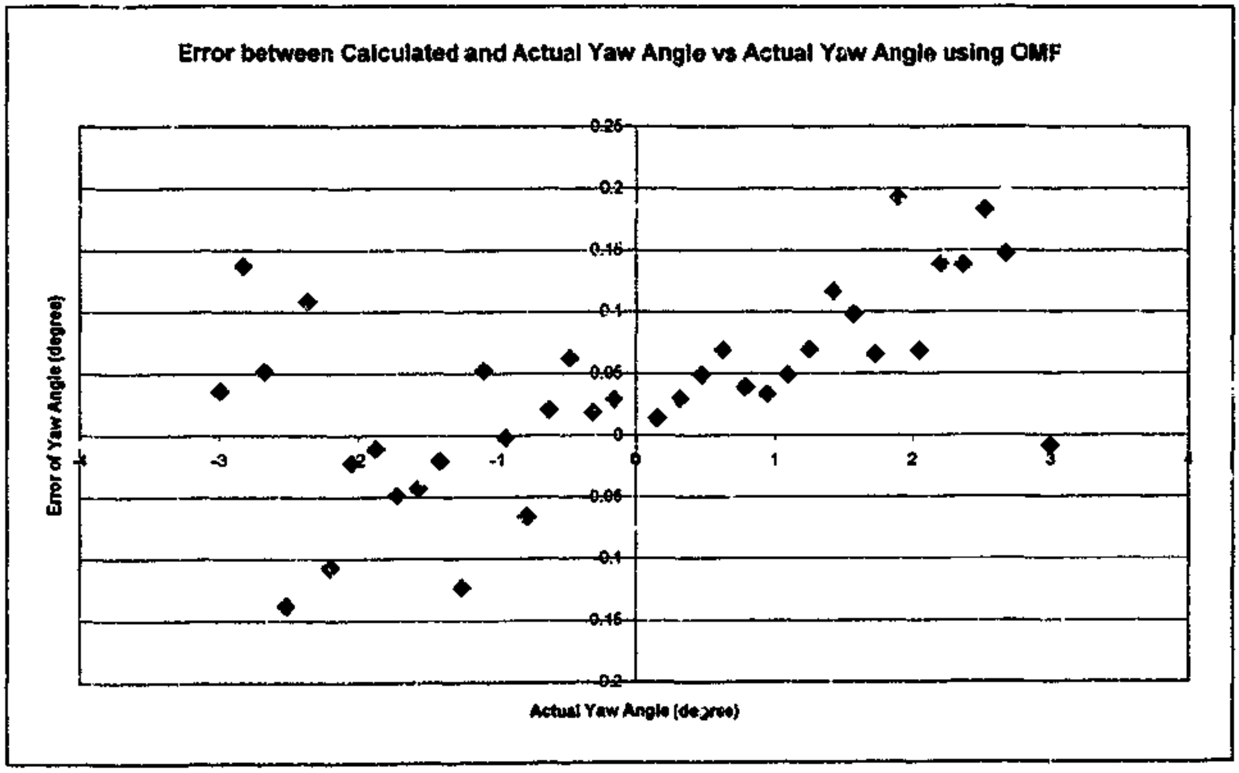


Figure 4.9: Error of calculated yaw angle using OMF vs actual yaw angle

In order to measure the roll angle, the incident laser beam has to be directed off the centre of the retroreflector as shown in Fig 4.10. By using a beam offset of 2 mm and with zero pitch and yaw angles, the roll angle is calculated using Eqs. 4.8 to 4.13. Fig. 4.11 shows the errors between calculated and actual angles. The error of the roll angle calculated using OMF is 2.5° (0.04 rad). This high amount of error is due to noise in PSD2. When the pitch and yaw angles are equal to zero, Eqs. 4.8 to 4.12 are rewritten as follows:

$$B_{1x} = -D_{2x} \tag{4.16}$$

$$B_{1z} = D_{2y} \tag{4.17}$$

$$B_{2x} = B_{1x} = -D_{2x} \tag{4.18}$$

$$B_{2z} = B_{1z} = D_{2y} \tag{4.19}$$

By substituting the above equations into Eq. 4.13, the roll angle is equal to:

$$\rho = \tan^{-1}\frac{B_{2z}}{B_{2x}} = \tan^{-1}\frac{D_{2y}}{-D_{2x}} \tag{4.20}$$

By using a beam offset of 2 mm, D_{2x} is equal to 1 mm. A noise value of 0.1 mm in D_{2y} will create an error of 5.71° (0.1 rad) in the calculated roll angle. This error can be reduced by using a larger beam offset. However, increasing the beam offset will reduce the range of angular displacements for the retroreflector. This is because the laser beam is being directed further away from the centre of both PSDs and will move out of the PSDs range more rapidly when there are changes in pitch and yaw angles. Furthermore, the non-linearity and non-uniformity of the PSDs also introduce error in the angles calculated.

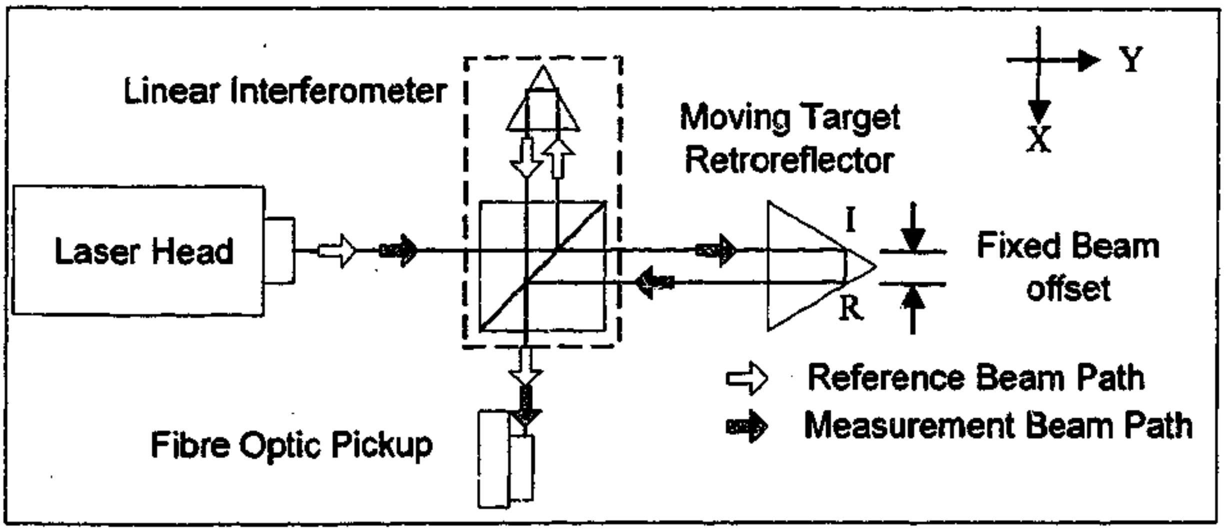


Figure 4.10: Laser interferometry set-up with incident beam off-centre

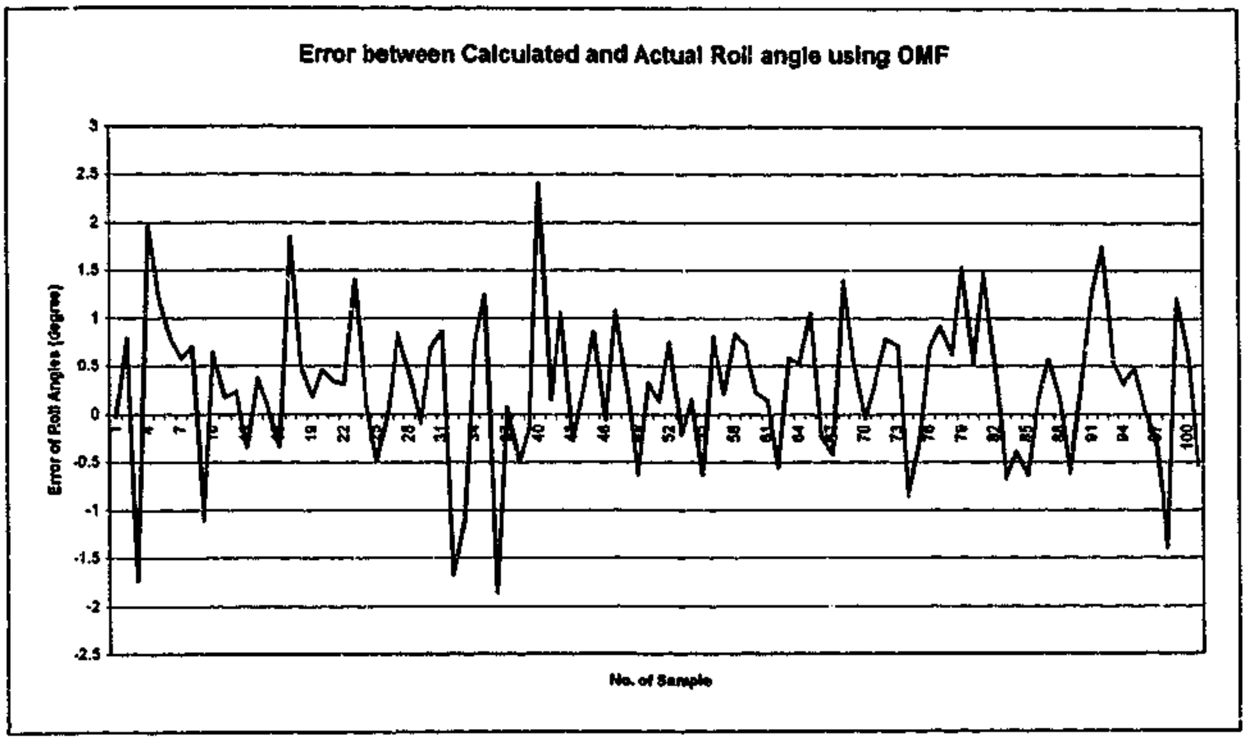


Figure 4.11: Error of roll angle calculated using OMF with pitch and yaw angles equal to zero

4.6 Orientation Compensation Algorithm

Based on the angles calculated using OMF, the relative angles between the laser beam and the retroreflector can be compensated by the Orientation Compensation Algorithm (OCA). A block diagram of the OCA is shown in Fig. 4.12. First, the Gimbal will be positioned at the initial reference point, which corresponds to zero roll, pitch and yaw angles with respect to the laser beam. When there are changes in the pitch and yaw angles, the motors associated with the Gimbal unit will be commanded to zero the angles at each update. The main advantage of this approach is that with the ability to rotate the retroreflector so that it always face the direction of incoming laser beam, the limitation of small incident acceptance angle of the retroreflector has effectively been removed. This improves the measuring range of the retroreflector. The change of orientation from the initial reference can be determined from the encoders' readings. A flowchart for the OCA is provided in Fig. 4.13. Figs 4.14 to 4.19 show the implementation of the OCA for the correction of pitch and yaw angles.

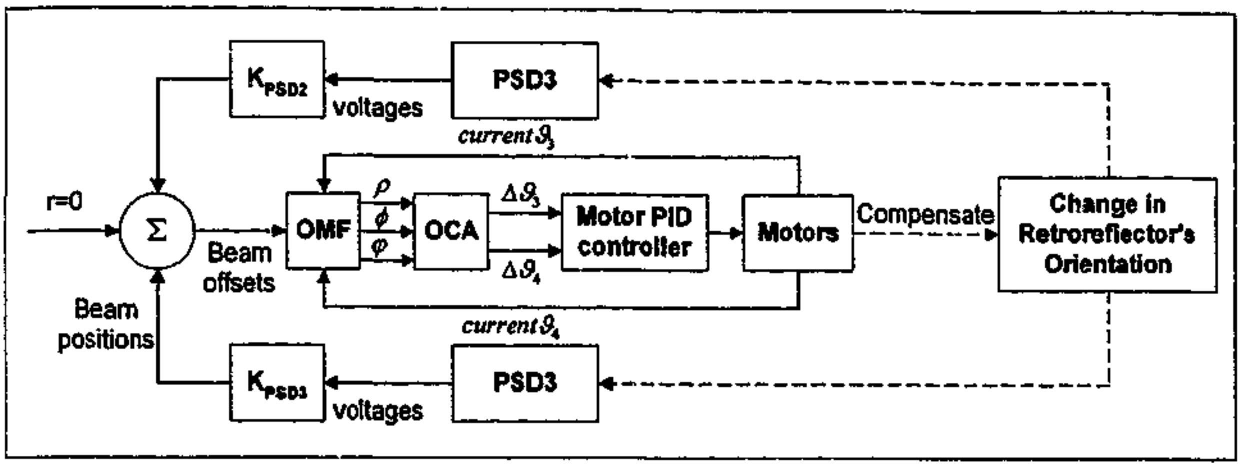


Figure 4.12: Orientation Compensation Algorithm control block diagram

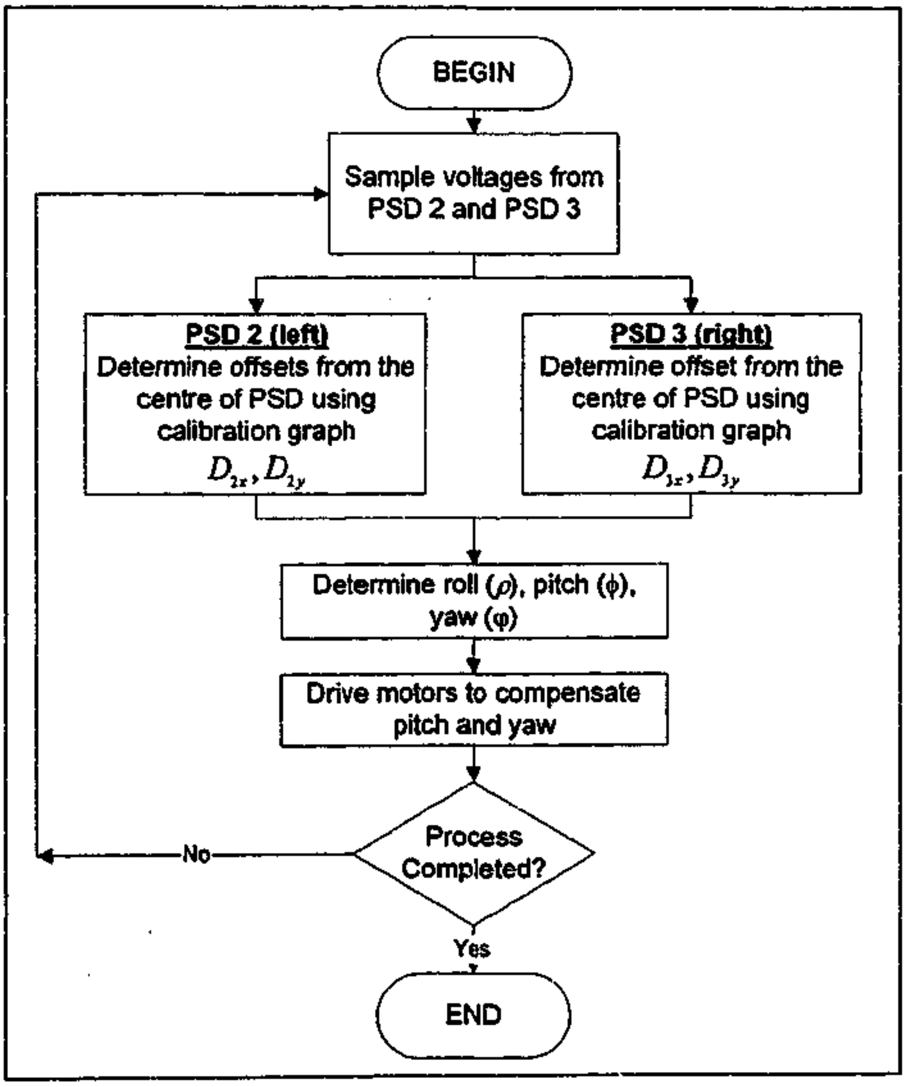

BEGIN
Sample voltages from PSD 2 and PSD 3
PSD 2 (left)
Determine offsets from the centre of PSD using calibration graph
D2x, D2y
PSD 3 (right)
Determine offset from the centre of PSD using calibration graph
D3x, D3y
Determine roll (ρ), pitch (ϕ), yaw (φ)
Drive motors to compensate pitch and yaw
Process Completed?
No
Yes
END

Figure 4.13: Flowchart for the orientation compensation algorithm

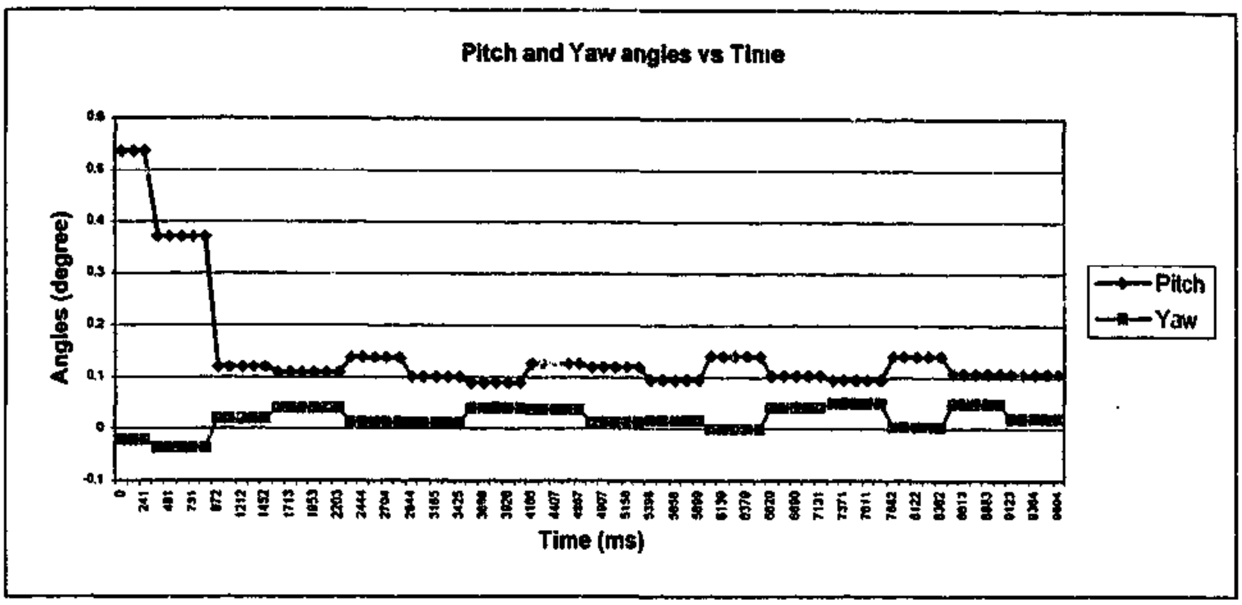


Figure 4.14: Single compensation of small pitch angle using OCA

Pitch angle steady state error = 0.10°

Yaw angle steady state error = 0.02°

Response time = 1 sec

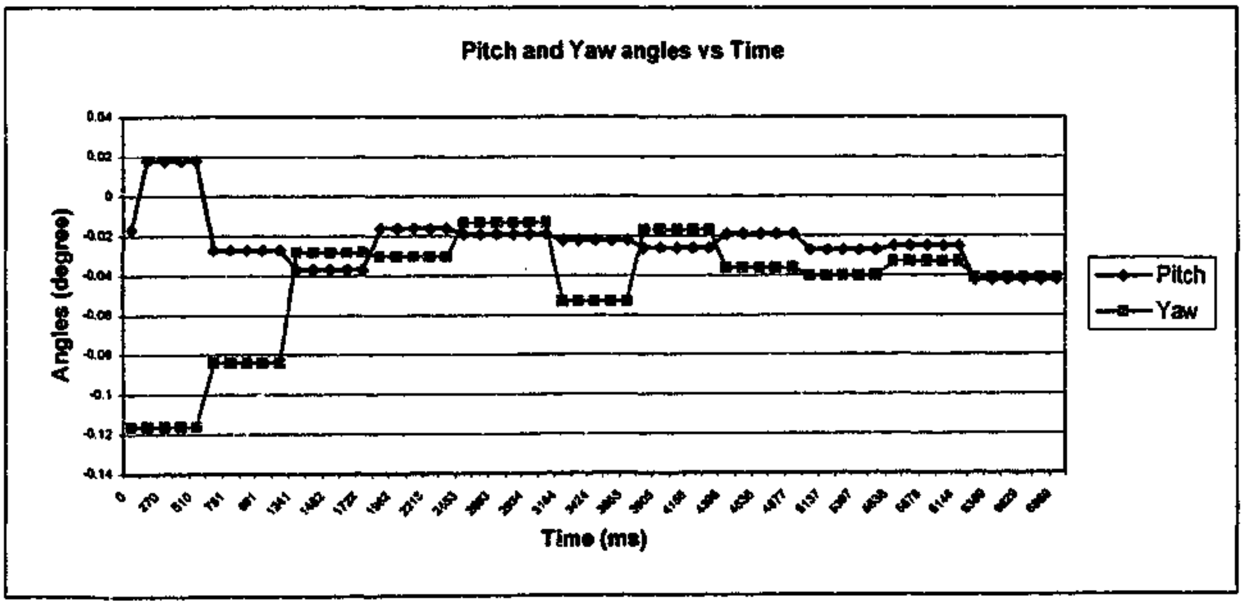


Figure 4.15: Single compensation of small yaw angle using OCA

Pitch angle steady state error = -0.04°

Yaw angle steady state error = -0.04°

Response time = 1.2 sec

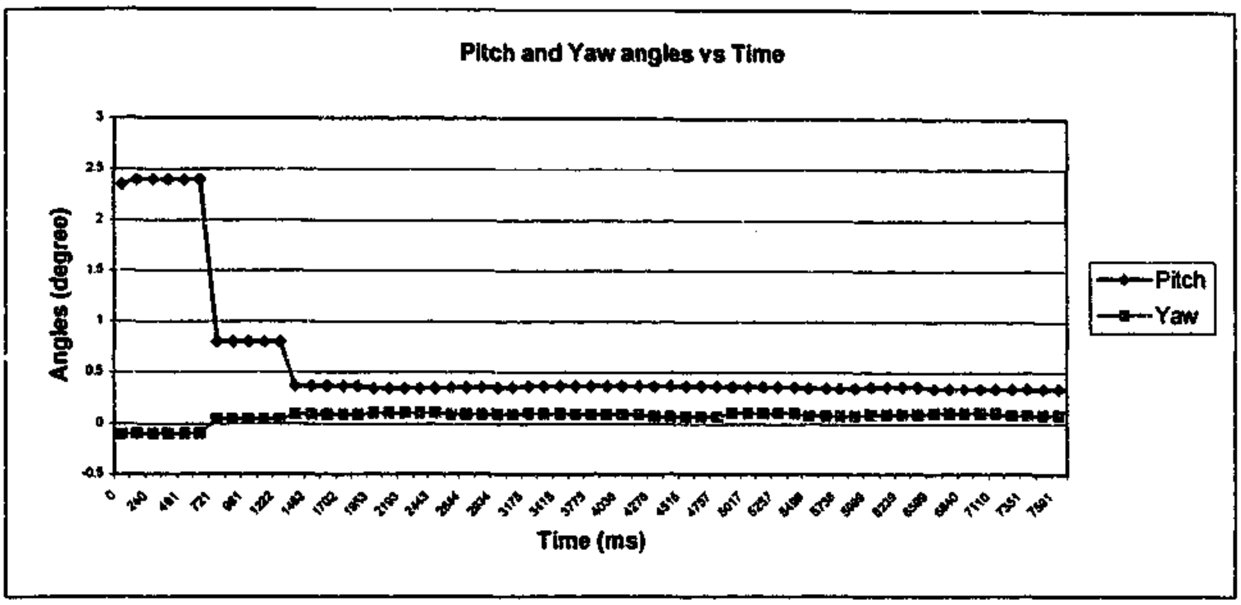


Figure 4.16: Single compensation of large pitch angle using OCA

Pitch angle steady state error = 0.34°

Yaw angle steady state error = 0.08°

Response time = 1.5 sec

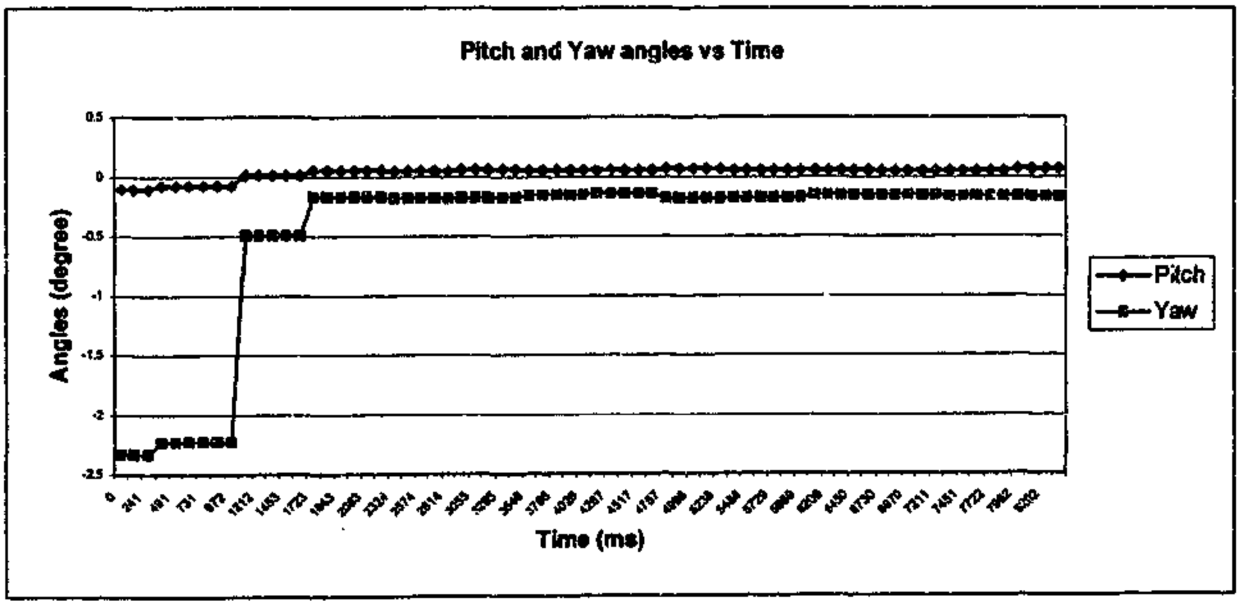


Figure 4.17: Single compensation of large yaw angle using OCA

Pitch angle steady state error = 0.06°

Yaw angle steady state error = -0.16°

Response time = 1.7 sec

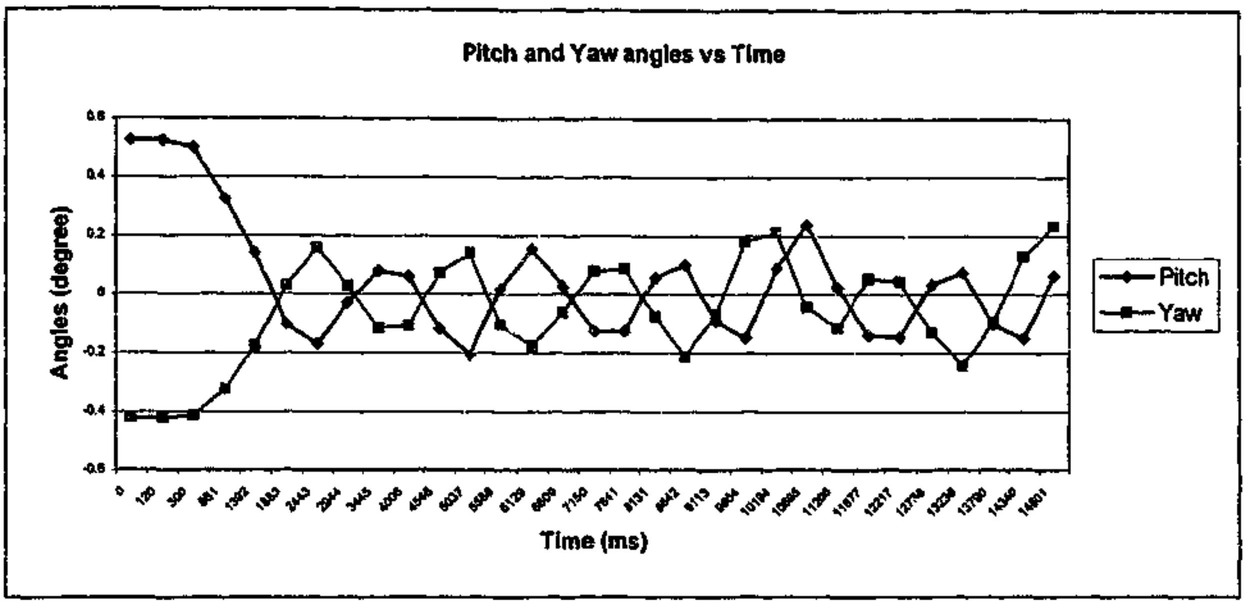


Figure 4.18: Continuous compensation of small pitch and yaw angles using OCA

Pitch angle steady state error = ±0.2°

Yaw angle steady state error = ±0.2°

Response time = 1.9 sec

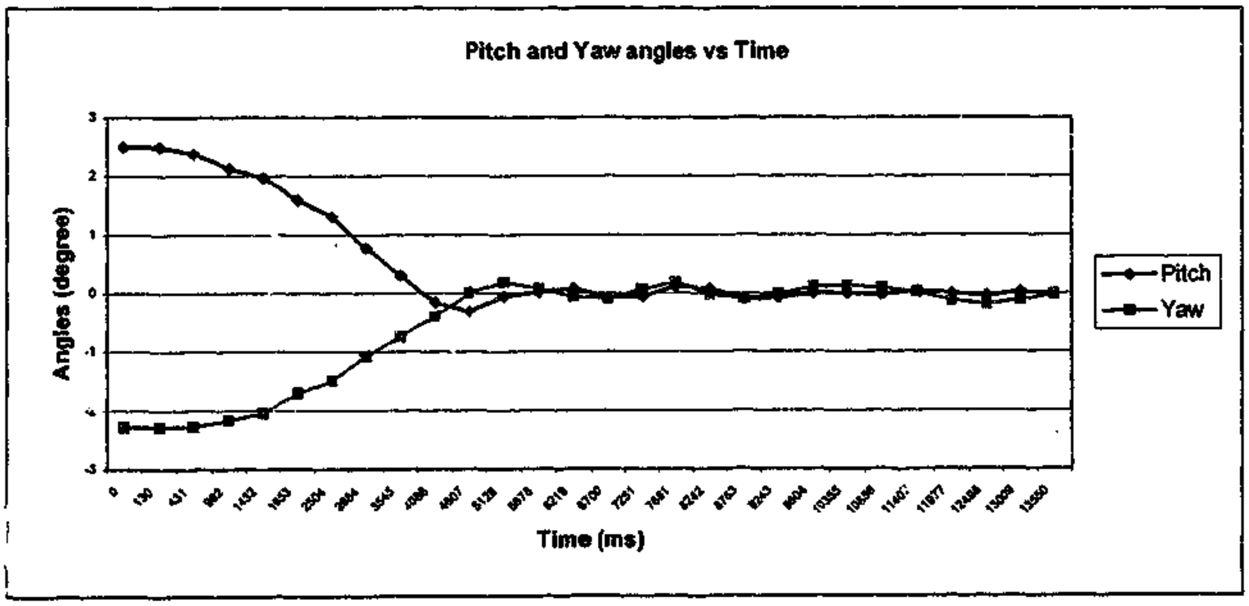


Figure 4.19: Continuous compensation of large pitch and yaw angles using OCA

Pitch angle steady state error = ±0.2°

Yaw angle steady state error = ±0.2°

Response time = 4 sec

Figs 4.14 and 4.15 show the single compensation of a small (<1°) pitch and yaw angles, respectively. The response time is about 1 second in both cases. Figs 4.16 and 4.17 illustrate the single compensation of large (>2°) pitch and yaw angles, respectively. The response time can be observed to be longer when compared with small angles compensation. This is due to the implementation of a constant velocity profile in both cases above. Further, the steady state error is higher compared to the error with small angles compensation. This is due to the growth of the error in the calculated angles with the increase in angular displacement of the retroreflector with respect to the laser beam as shown in Figs 4.8 and 4.9. Since the motors are commanded to move by the angles calculated, the error in the calculated angles results in inaccurate angular displacements commanded.

Figs 4.18 and 4.19 illustrate the implementation of continuous OCA for the compensation of pitch and yaw angles. The response time for small and large angles compensation is 2 and 4 seconds, respectively. The response is slower compared to single OCA due to the motors having to move through the motion profile to repeatedly compensate for the angles calculated. This results in the starting and stoping of the motors and thus a longer delay. Moreover, the continuous command received by the motor controller causes some interruption to the motors' motions. From Figs 4.18 and 4.19, it can be observed that the pitch and yaw angles are varying between ±0.2° at steady state. This fluctuation is the result of the noise in the PSDs. This noise led to the fluctuations of the calculated pitch and yaw angles of the retroreflector with respect to the laser beam and the motors are commanded to compensate for these changes continuously.

4.7 Integration with Experimental LISM apparatus

The orientation measurement methodology can be incorporated into the experimental LISM apparatus to provide for the position and orientation (pose) measurements. However, this is not the objective of this study. Further, the current set-up has the following limitations:

A) By using the Gimbal unit, more optics are added to the laser beam path. Each optic will reduce the beam optical power due to imperfect optical efficiencies. With the addition of a 70-30 beamsplitter, the optical power at PSD 1 within the experimental LISM apparatus is further reduced as shown in Fig 4.20. Photo-currents generated by the beam is 3.11 μAmp, which gives a Signal to Noise (S/N) ratio of 12.44. This ratio is half of the S/N ratio of 29.4 as shown in Section 3.9. Due to the lower S/N ratio, the target following performance is greatly affected.

B) The experimental LISM apparatus described in Chapter 3 is set-up to direct the laser beam to the centre of the retroreflector. To measure the roll ρ angle, the incident beam has to be directed off the centre of the retroreflector as shown in Fig. 4.10. This involves a major relocation of the laser head and the fibre optic pickup so that the reference and the measurement beams will still completely overlapped each other. The PSD in the LISM apparatus also has to be shifted so that when the reflected beam is at point R (Fig. 4.10), the tracking offset measured by the PSD will be zero.

C) Additional codes have to be added into the LISM's control program. The PSD object has to sample 12 channels of PSD data instead of 4. An orientation compensation algorithm (OCA) has to be added together with the Target Following Control Algorithm (TFCA) (Section 3.7) to the on-board program on the ACR2000 controller. 4 motors are being controlled as opposed to 2, and 4 encoders have to be sampled compared to 2. All these factors would reduce the sampling frequency of each of the other sub-systems and the rate of execution of the control program.

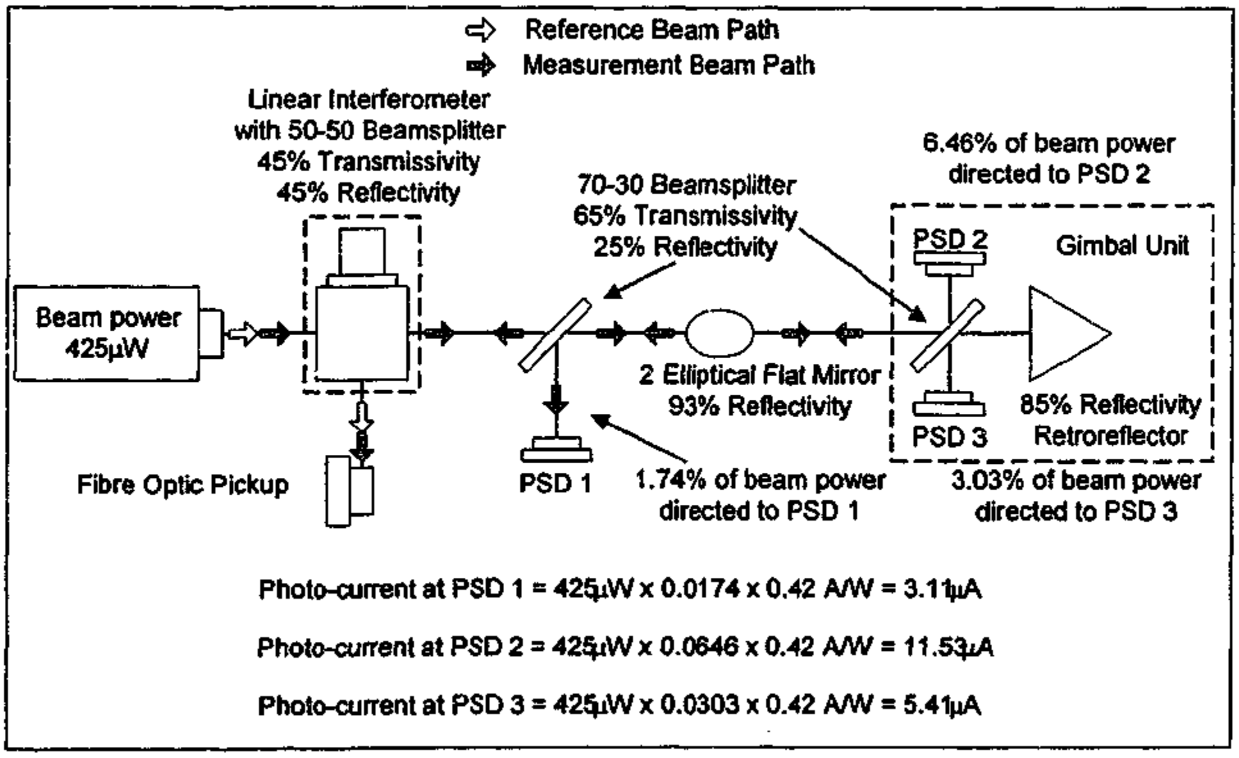


Photo-current at PSD 1 = 425μW x 0.0174 x 0.42 A/W = 3.11μA

Photo-current at PSD 2 = 425μW x 0.0646 x 0.42 A/W = 11.53μA

Photo-current at PSD 3 = 425μW x 0.0303 x 0.42 A/W = 5.41μA

Figure 4.20: Optics efficiency with Gimbal unit

The following improvements are recommended to allow for the pose measurements of the target:

A) The use of higher laser beam power and a PSD with a lower dark current so that higher S/N ratio can be obtained. The performance of the TFCA, OCA and pose measurements can be greatly improved. S/N ratio in the order of 100 is required.

B) A faster computer is required to increase the sampling frequency of all sub-systems and rate of execution of control programs.

C) Higher resolution, direct drive closed-loop motor systems can be used. This reduces the position loss, vibration, overshoot, gear backlash and response time of the motors, resulting in faster and smoother compensation of the pitch and yaw angles calculated.

D) Due to the reason that the full range of the PSDs is used for orientation measurement, PSDs with better linearity and uniformity can greatly improve the accuracy of angles calculated using OMF.

E) The Gimbal unit developed for this study is heavy. This whole assembly has to be rebuilt with strong but light material. The weight of the motors, PSDs, beamsplitter and retroreflector also has to be taken into account to reduce the weight.

4.8 Summary

In this chapter, the experimental investigation of the dual PSD-based orientation measurement methodology has been presented. Due to limitations presented, full integration of this methodology into the experimental LISM apparatus could not be performed to allow for the real time pose measurements of the target. However, from the experimental results, it can be observed that the proposed methodology can be used for the measurements of the retroreflector's orientation relative to the incident laser beam. Moreover, the Gimbal unit can be rotated based on the angles calculated. The retroreflector will therefore always face the direction of the incident laser beam, maintaining the line of sight for the incident laser beam. This approach effectively removed the limitation of small incident acceptance angle of the retroreflector. Next chapter present the kinematic model of the experimental LISM apparatus, which is essential for the calculation of the positions of the end-effector. Analysis of the uncertainties of the measurements acquired by the LISM apparatus and the Gimbal unit are also evaluated.

Chapter 5

Kinematic Model and Uncertainties Analysis of Experimental LISM Apparatus

5.1 Introduction

This chapter presents formation of the kinematic model of the experimental LISM apparatus. This model is important for the calculation of the target Cartesian positions using the angular displacements of the motors, the laser interferometer measurements, PSD measurements and the geometry of the apparatus. The accuracy of the LISM's measurements obtained is dependent on the accuracy of the parameters used in the model. However, no mechanical system is ever manufactured perfectly. For the experimental LISM apparatus, there will always be slight variations in the kinematic parameters or error due to the sensors and transducers used. These errors can be broadly classified into geometric and non-geometric errors. Geometric errors are caused by the inaccuracies in manufacturing and assembly of the components (such as mirror-positioning error, motors, laser and PSD misalignment). These errors can be compensated through calibration using a kinematic model of the LISM apparatus, thus improving the accuracy of the target position's measurement made by LISM. Non-geometric errors are caused by the inherent characteristics of the sensors and transducers, the environmental effects, the dynamic resonance, etc. Non-geometric errors comprise of tumbling motion, motor backlash, sensitivity of the material to temperature, encoder coupling, noise, etc. These errors occur randomly and are difficult, if not impossible, to determine. These errors will contribute to the uncertainties of the measurements acquired by the LISM apparatus.

5.2 Errors Analysis

Position measurements acquired by experimental LISM apparatus and orientation measurements acquired by the orientation measurement methodology rely heavily on the use of other sensors and transducers. Some of the possible sources of errors in the measurements include:

- unknown effects of the environmental conditions on the measurements;
- error in measurements acquisition from various instruments;

- resolution of various instruments;
- approximations and averaging steps (where applicable) made in the measurements process;
- inexact values of reference instruments (e.g. for calibration purposes).

Four different sub-systems were developed in this study. These sub-systems are the laser interferometer sub-system, beam-steering sub-system, position sensitive detector (PSD) sub-system, and finally the orientation measurement sub-system. The sources and magnitudes of errors associated with each sub-system are analysed in the following sections.

5.2.1 Laser Interferometer Sub-system

The overall accuracy of any laser interferometry apparatus is affected by a number of different factors including, [25, 59]:

- environmental factors;
- geometric and manufacturing errors;
- Instrument errors.

5.2.1.1 Environmental factors

The refractive index of the laser beam is affected by the variations in the environmental conditions such as ambient temperature, air pressure, and humidity. A variation in the index of refraction introduces variation in the wavelength. As the laser beam displacement is computed using the wavelength (Section 2.2), variation in wavelength during measurement introduces error in the displacement measured. An error of approximately ±1μm/m occurs in the index of the laser used due to each of the following environmental changes, [25]:

- 1° C change in the air temperature;
- 2.8mm Hg change in air pressure;
- 90% change in the relative humidity.

A uniform change in temperature also causes expansion or contraction of the interferometer components and introduces further error in the displacement measured. The linear interferometer used within this LISM apparatus has a temperature coefficient of less than 0.022 μm/°C.

5.2.1.2 Geometric and manufacturing errors

Misalignment of the optics introduces errors such as cosine error, Abbe' offset error and polarisation mixing error [25]. A cosine error is a measurement error caused by an angular misalignment between the laser beam and the axis of motion of the displacement being measured. The cosine error degrades the signal received by the receiver, and more importantly, reduces the accuracy of measurement because of not measuring the actual target displacement. Abbe' offset is the result of an offset between the measurement laser beam and the axis of motion of the target. Positioning the beam as close as possible to the axis of motion will reduce the Abbe' offset.

Polarisation mixing error is due to the misalignment of the laser head relative to the interferometer and the imperfections in the polarisation beamsplitter. This produces a leakage of undesired polarisation state into the two polarised frequencies. Polarisation mixing will introduce distortion in the interference signal, which will produce a non-linearity between the measured displacement and the actual displacement, and thus affect the accuracy of measurement. Fig. 5.1 shows a comparison between properly aligned polarising system and polarising mixing. With properly aligned polarisation, the measurement and reference beams each contain only one frequency, F_1 and F_2, respectively. For the polarisation mixing shown in Fig. 5.1, the measurement and reference beams contain not only the frequencies F_1 or F_2, respectively, but also a small portion of the each other's beam frequency. Polarisation mixing error is cyclic with a period of 360° occurring approximately every 158 nm (a quarter of the wavelength) of displacement for a double pass interferometer, [25]. This cyclic error is non-cumulative.

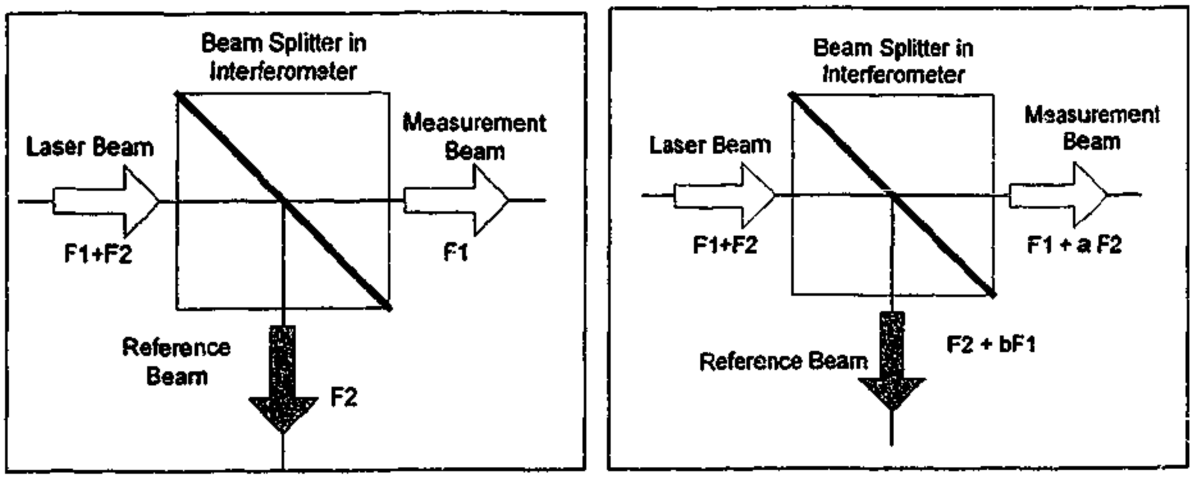


Figure 5.1: Comparison between proper polarisation alignment and polarisation mixing

5.2.1.3 Instrument errors

The laser interferometer used has a maximum electronic error of 1.3 counts. The contribution of this error to the uncertainty analysis is a product of the electronic accuracy and the optical resolution of the interferometer used. In this case, with each count equals to 2.74 nm, the electronic error is ±3.21 nm, [25].

Due to the imperfections of the optical components and their coatings, there will again be a leakage of undesired polarisation state. This will cause polarisation mixing error of the beams within the interferometer. For a linear interferometer, this error is ±0.8 nm, [25].

5.2.2 Beam Steering Sub-system

As shown in Section 3.5, this sub-system consists of an open-loop stepper motor with optical encoder and a closed loop servo motor with resolver feedback. The vertical axis controlled by the servo motor has a resolution of 0.00059° and an accuracy of 0.0125°. The horizontal axis is controlled by a stepper motor, and it has a maximum resolution of 0.0045° using a step rate of 80000 steps per revolution. This motor has an accuracy of 0.045°. The encoder used for the horizontal axis is a 10000-line encoder with a maximum resolution of 0.009°.

The stepper motor is responsible for rotating the attached mirror to direct the beam along the LISM's z-axis, whereas the servo motor is responsible for turning the attached mirror, directing the beam along the LISM's x-axis. Both axes of rotations have to intersect each other, and the point of intersection corresponds to the tracking reference point (Section 3.7) of the laser beam. However, imperfections in the construction process (e.g. the positioning of the mirrors, misalignment of the rotational axes, etc.), and in the assembly of the beam steering sub-system affect its precision. Other possible sources of errors are inherent hardware error, motor electronic design, heat generated from friction between the moving parts, shaft misalignment between the motors and the encoders, and shaft bending due to load and tumbling of motors [60].

5.2.3 PSD Sub-system

In Section 3.4, it was shown that the main sources of error include the linearity and the sensitivity of the PSD, the beam power incident on the sensor area, the amount of dark current, and the environmental noise [26, 41]. The maximum error is ±0.07 mm. This error was calculated from the deviation of the calibrated results from the known reference.

5.2.4 Orientation Measurement Sub-system

The main source of error for the dual PSD-based orientation measurement approach is the error inherent in the PSD. Furthermore, Eqs. 4.6 to 4.13 are used to describe the orientation of the target retroreflector with respect to the laser beam. These equations consist of the geometric parameters of the Gimbal unit. Imperfection in manufacturing and assembly of the Gimbal unit will degrade the accuracy of the measurements acquired.

5.2.5 Reflecting Target

The reflecting target used is the most commonly used air-path type retroreflector. Fig. 2.8 shows the make-up of this retroreflector. An ideal retroreflector has the attributes to ensure that a light beam incident onto any point on the retroreflector, regardless of its orientation, is reflected through 180°, and the reflected beam must travel back parallel to the incident beam. The centre of the retroreflector is always exactly centred between the incident and the reflected beams.

Possible errors in a retroreflector are associated with the construction of the retroreflector. For air-path type reflector, the mirrors are joined together usually with adhesive. There could be an imperfection in joining the mirrors, and the three mirrors used might not intersect each other at a single point. This causes an offset error so that the centre of the retroreflector is not exactly centred between the incident and the reflected beams. This contributes to the error in the position measurement in LISM. Further, there may exist an error associated with the orthogonality of the mirrors. Therefore, the beam may not be reflected through an angle of 180°, rather with a slight angular deviation and there could be a change in the path of the beam. The angular deviation is specified as 0.0014° (5 arc sec) in this case as stated in [27]. Moreover, as the beam offset recorded by the PSD is not the real offset due to the lateral motion of the retroreflector, there will be an additional error in the beam offset measured. Fig. 5.2 shows the error due to the imperfection in the construction of the retroreflector.

5.2.6 Measurement Platform Geometry and Environmental Effect

All the sub-systems including the laser interferometer and the other optics are located on an optical breadboard. The flatness of the breadboard will affect the accuracy of the LISM by contributing to the cosine error. Furthermore, the misalignment between optics contributes further to the inaccuracy of the results. Environmental conditions also affect the properties of materials used in the LISM apparatus and thus its geometry.

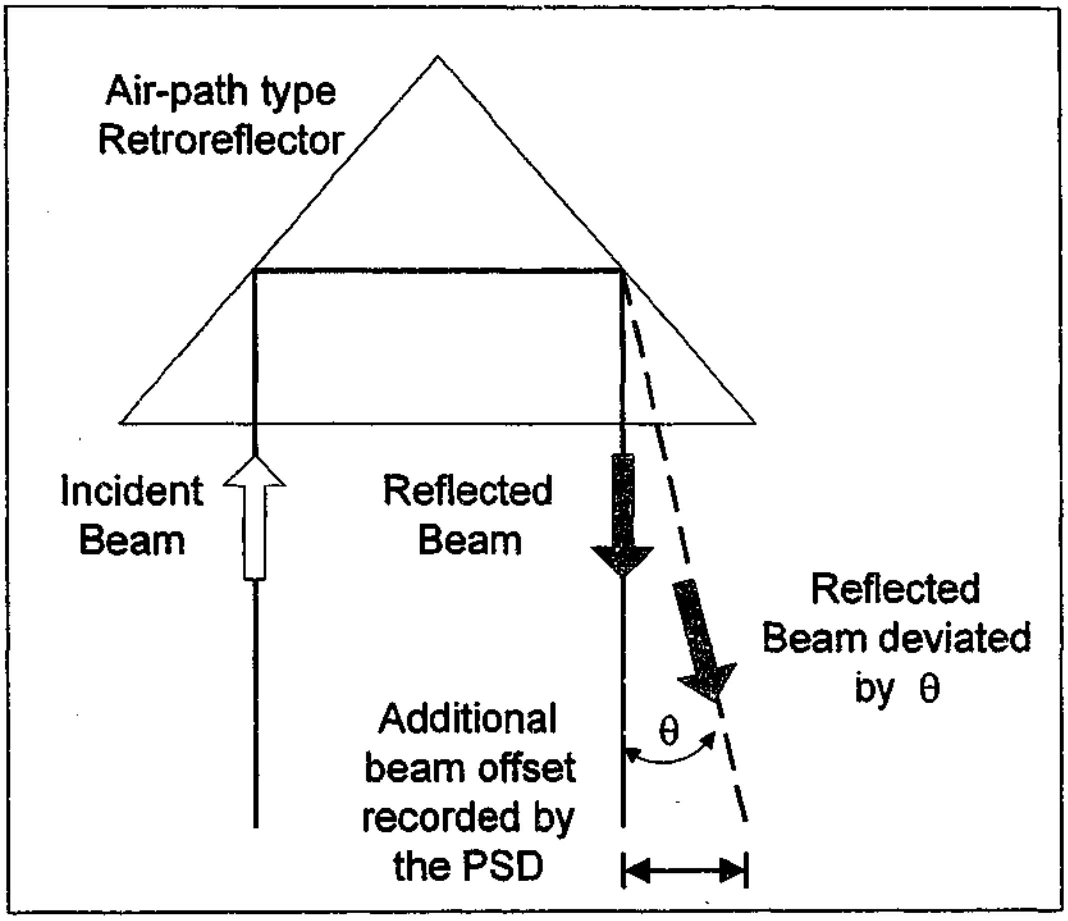


Figure 5.2: Angular deviation of beam caused by imperfection of retroreflector

The following sections present the kinematic model and calibration process to reduce geometric errors. The uncertainty analysis is presented next to determine the overall measurement uncertainty of the LISM apparatus.

5.3 Kinematic Model and Calibration Methodology

Kinematic model provides a mathematical description of the path of the laser beam to the retroreflector on the robot end-effector and makes use of the coordinate systems (CS) shown in Fig. 5.3. A co-ordinate frame is placed on each of the mirrors with the xy-plane describing the mirror surface and the z-axis directed to the blind side of the mirror. In the current configuration, the description of the mirrors relative to the reference co-ordinate frame are given as follows [61]:

$$M_1 = [Transl(x_1, y_1, z_1)][A_{mirr}(\alpha_1, \beta_1, z_{m1})] \tag{5.1}$$

$$\begin{aligned} M_2 = {} & [Transl(x_2, y_2, z_2)][Rotz(\vartheta_{c1})][Transl(dx_3, dy_3, dz_3)] \\ & [Rotx(\vartheta_{c2})][A_{mirr}(\alpha_2, \beta_2, z_{m2})] \end{aligned} \tag{5.2}$$

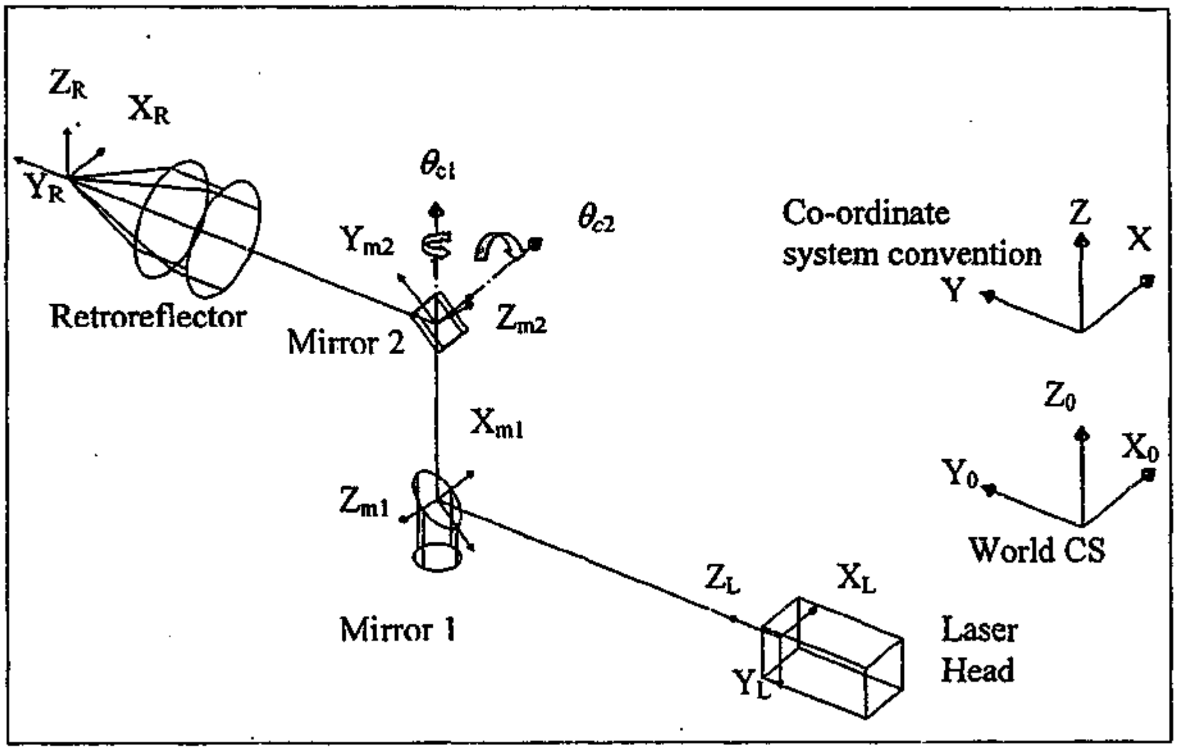


Figure 5.3: Kinematic model of the experimental LISM apparatus

where $Rotz(\vartheta_{c1})$ and $Rotx(\vartheta_{c2})$ are transformation due to rotations of the vertical and horizontal motors, M_1 is the co-ordinate frame on mirror 1 relative to the reference co-ordinate frame, M_2 is the co-ordinate frame on mirror 2 relative to the reference co-ordinate frame. $Trans(dx_3,dy_3,dz_3)$ is the transformation matrix from the vertical rotational axis to the horizontal rotational axis. This is to account for the misalignment of the motors rotational axes.

$A_{mirr}(\alpha,\beta,z)$ is the transformation matrix to place the xy-plane of the mirror co-ordinate frame on the mirroring surface and the z-axis towards the blind side of the mirror [22]. On a perfectly plane mirror surface, the rotation about z-axis and translation along the xy-plane will not alter the reflected beam. Therefore,

$$A_{mirr}(\alpha_i,\beta_i,z_{mi}) = [Rotx(\alpha_i)][Roty(\beta_i)][Transl(0,0,z_{mi})] \qquad (i = 1, 2) \qquad (5.3)$$

Another co-ordinate frame is placed on the source of the laser beam with the z-axis aligned with the beam. This frame is translated to each mirror in turn and reflected according to the physical positions of each mirror described by the co-ordinate frame M_1 and M_2. If L_N denotes the laser frame, the reflected laser frame L_{N+1} with its origin on the mirror plane can be described as follows:

$$L_{N+1} = [L_N][A_{reflection}(M_N, L_N] \quad (\mathrm{N} = 0, 1) \tag{5.4}$$

$$L_0 = Transl(a_{XL}, a_{YL}, a_{ZL})Rot(\alpha_L, \beta_L, \gamma_L) \tag{5.5}$$

where L_0 represents the transformation from the reference co-ordinate frame to the first laser frame.

$A_{reflection}$ is a function of the mirror's and the current laser's co-ordinate frames [22]. After reflection from the last mirror, the beam is directed towards the target retroreflector. This is a simple translation along the beam and can be described as follows:

$$L_R = [L_2][Transl(a_{XR}, a_{YR}, a_{ZR})][Rot(\alpha_R, \beta_R, \gamma_R)] \tag{5.6}$$

where a_{XR} and a_{YR} are deviations from the centre of the retroreflector when the laser beam does not hit the centre of the retroreflector. Both can be determined from the target lateral offset calculated from Eq. 3.15 and 3.16, as follows:

$$a_{XR} = \frac{\Delta d_1}{\cos \vartheta_{c1}} \tag{5.7}$$

$$a_{YR} = \frac{-\Delta d_2}{\cos 2\vartheta_{c2}} \tag{5.8}$$

where Δd_1 and Δd_2 are target lateral offsets calculated from Eq. 3.15 and 3.16, respectively. a_{ZR} is the displacement of the beam from the last mirror to the retroreflector and can be calculated from the following equation:

$$a_{ZR} = l_{radial} - \Delta d_1 \tan \vartheta_{c1} - \Delta d_2 \tan 2\vartheta_{c2} \tag{5.9}$$

where l_{radial} is the radial distance of the laser beam.

To account for the geometric errors, every transformation parameter p in the kinematic model of the LISM apparatus consists of the ideal value given in the design specifications plus an error value Δp, which represents the geometric error associated with that parameter. This provides the following equation:

$$p = p_{ideal} + \Delta p \tag{5.10}$$

The kinematic model consists of 21 parameters with corresponding geometric errors. These errors can be compensated through calibration by comparing the position determined using the model with those obtained from a known reference. The difference in position can be related to the system parameters by J, the calibration matrix, which represents a differential change in position of the modelled transformation ${}^{0}L_{R}$ with respect to differential change in parameter p.

$$\Delta e = J\,\Delta p \qquad (5.11)$$

where Δe is the differential change in position between the measured and the calculated value using the modelled transformation ${}^{0}L_{R}$, Δp is the differential change in parameters p.

The generalised inverse of J in Eq. 5.11 can be found and by using linear least squares method, and thus Δp can be found by the following formulation, [11, 62]:

$$\Delta p = J^{+}\Delta x \qquad (5.12)$$

where J^{+} is the generalised inverse or commonly known as the Moore-Penrose inverse of the calibration matrix.

The calibration was carried out using 150 reference points. Eq. 5.12 was solved using a program prepared in MATLAB in conjunction with the optimisation toolbox (refer to Appendix C for the complete MATLAB program). Table 5.1 tabulates the parameters before and after calibration obtained by the use of this methodology.

Figs. 5.4 and 5.5 compare the errors of the position measurements obtained using the ideal model and the calibrated model, respectively. From Fig. 5.5, the accuracy of position measurements made by LISM apparatus is ±0.5mm within a radial distance of 900mm. Accuracy degrades further with radial distance higher than 900mm. The overall position error has been reduced significantly (by 89%) when compared to position measurements calculated using the ideal model. However, the calibrated position error is high due to several reasons. Firstly, the repeatability of the motors is low, especially the stepper motor due to open-loop configuration. There can also be undetected position loss that introduces error in the angular displacements detected by the encoders, thus results in inaccurate position measurement. Secondly, imperfections in the roundness of the motor axis introduce tumbling of the motors [60]. Tumbling of the motors affects each parameter in the model. However, this is not modelled in this study. Thirdly, the redundancy of the parameters is not being studied. Redundancy of the kinematic parameters may affect the reliability of the parameter estimated [22]. Furthermore, the measurements obtained from each sub-system have high uncertainty (as presented in next section). This introduces uncertainty in position measurements, which affects the accuracy of the parameters estimated.

Table 5.1: Kinematic parameters before and after calibration

	initial	calibrated		initial	calibrated		initial	calibrated
a_{XL} (mm)	0.00	0.00	a_{X1} (mm)	0.00	0.00	a_{X2} (mm)	0.00	-0.40
a_{YL} (mm)	0.00	0.00	a_{Y1} (mm)	465.13	465.13	a_{Y2} (mm)	465.13	465.72
a_{ZL} (mm)	162.00	162.00	a_{Z1} (mm)	162.00	162.00	a_{Z2} (mm)	548.37	534.51
α_L (deg)	-90.00	-90.12	α_1 (deg)	-135.00	-135.12	da_{X3} (mm)	0.00	0.00
β_L (deg)	0.00	0.00	β_1 (deg)	0.00	0.14	da_{Y3} (mm)	0.00	22.07
γ_L (deg)	0.00	-0.15	z_{m1} (mm)	0.00	0.00	da_{Z3} (mm)	0.00	0.00
						α_2 (deg)	45.00	44.86
						β_2 (deg)	0.00	0.17
						z_{m2} (mm)	0.00	25.67

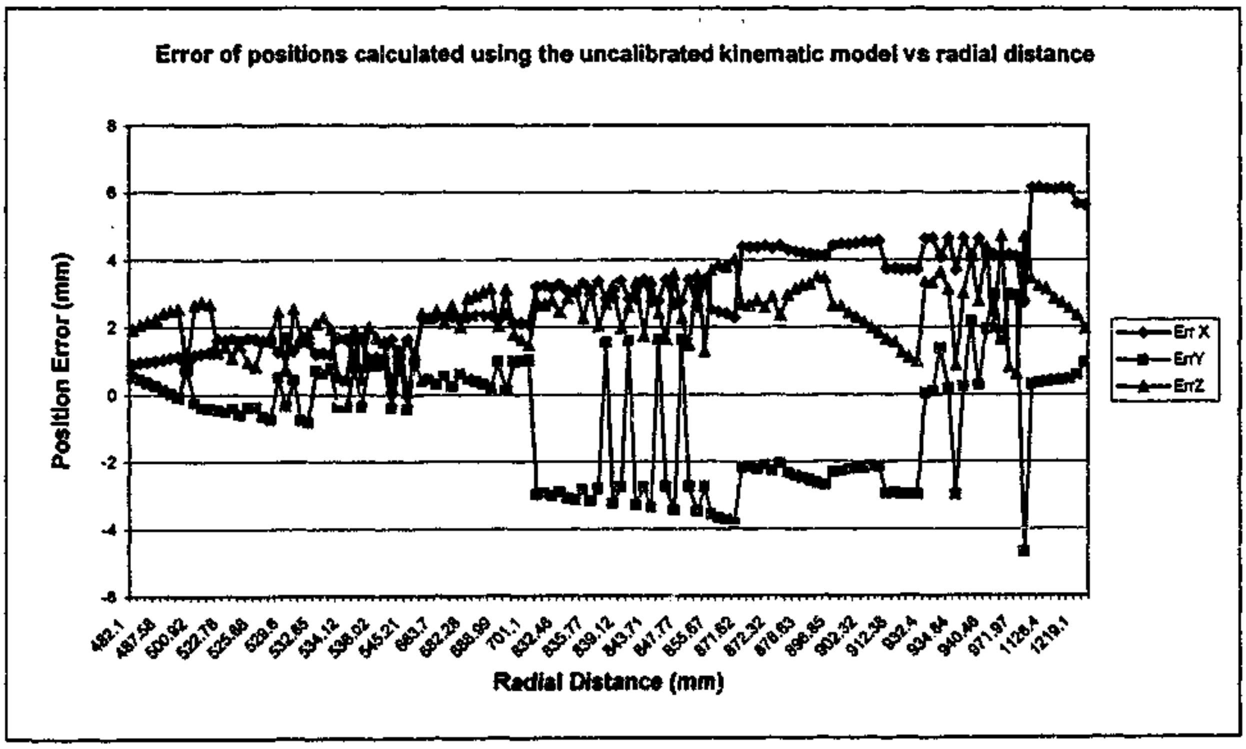


Figure 5.4: Error of position measured using LISM apparatus calculated using the uncalibrated kinematic model with respect to radial distance from the last mirror

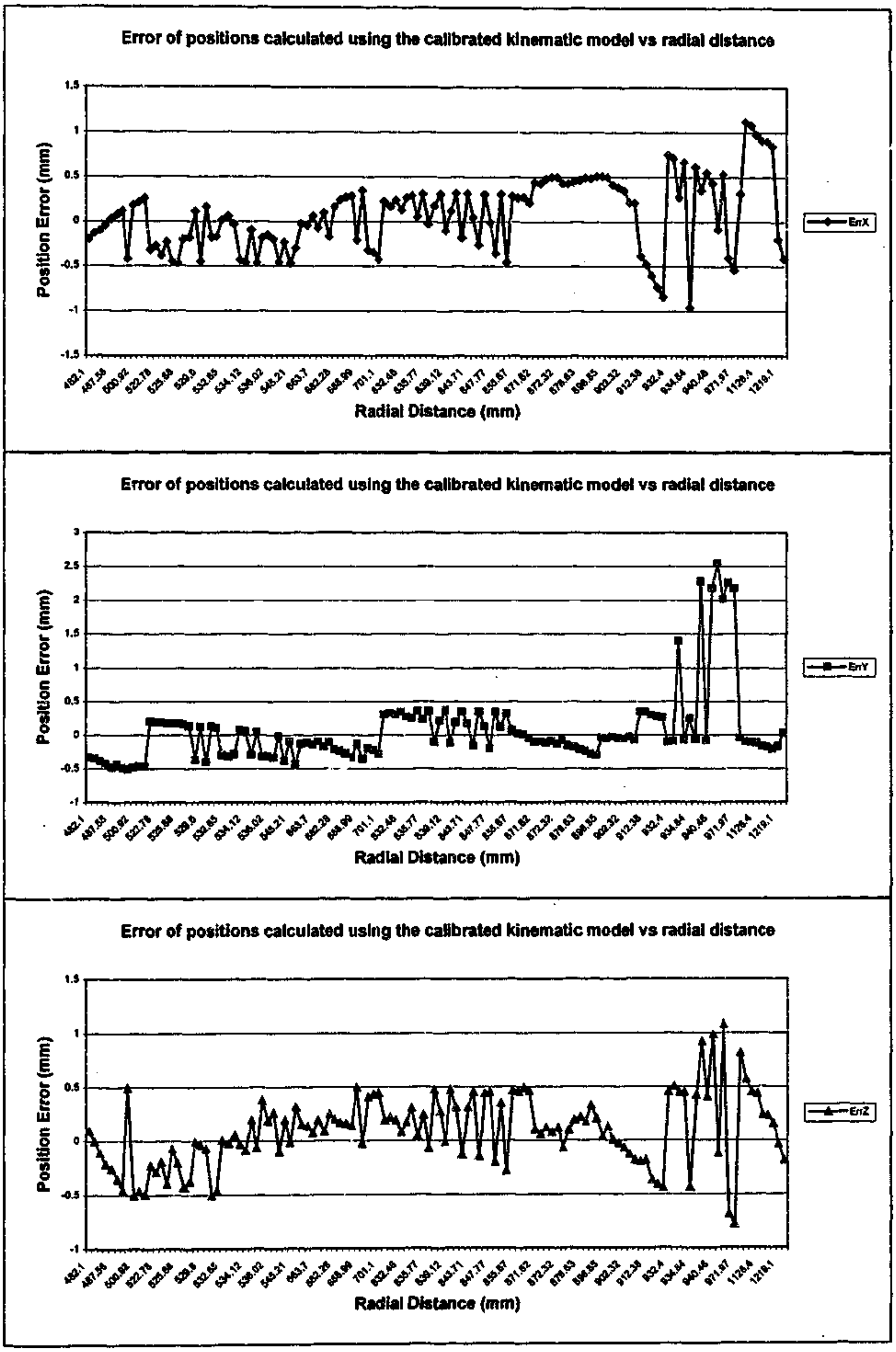


Figure 5.5: Error of position measured using LISM apparatus calculated using the calibrated kinematic model with respect to radial distance from the last mirror

5.4 Measurement Uncertainties

This section describes the uncertainties associated with LISM apparatus. An uncertainty analysis approach is introduced and the overall measurement uncertainty of the LISM apparatus and the Gimbal unit is presented [61].

5.4.1 Expression and Analysis of Uncertainty

The objective of any measurement technique is to determine the value of the measurand (i.e. a particular quantity to be measured at a particular condition). In general, the result of a measurement technique is only an estimate of the true value of the measurand. The result is only complete when stated with the uncertainty of that estimate. The uncertainty, therefore, indicates the dispersion of the values of the measurand, [63]. The total uncertainty comprises uncertainties of many components, which can be obtained from either the results of a series of measurements or experience/other information. The former is termed Type A Uncertainty, and the latter Type B Uncertainty, [63]. In order to develop a formulation for the uncertainties, it is important to establish the formulation for correlated and uncorrelated inputs. For a measurement *m*, whose results depend on uncorrelated input estimates $x_1, x_2, ...x_N$, the standard uncertainty of the measurement is obtained by appropriately combining the standard uncertainties of these input estimates. The combined standard uncertainty of the estimate *m* denoted by $u_c(m)$ is calculated from the following equations, [63]:

$$m = f(x_1, x_2, ..., x_N) \tag{5.13}$$

$$u_c^2(m) = \sum_{i=1}^{N} \left[\frac{\partial f}{\partial x_i} \right]^2 u^2(x_i) \tag{5.14}$$

where *f* is the function of *m* in terms of input estimates $x_1, x_2, ...x_N$, $u(x_i)$ is a standard uncertainty of inputs which may be evaluated either from Type A Uncertainty or from Type B Uncertainty, $u_c^2(m)$ is known as the combined variance.

When the input estimates are correlated, the combined variance $u_c^2(m)$ associated with the results of a measurement is determined by the following equation:

$$u_c^2(m) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j} u(x_i, x_j)$$

$$= \sum_{i=1}^{N} \left[\frac{\partial f}{\partial x_i} \right]^2 u^2(x_i) + 2 \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j} u(x_i, x_j) \tag{5.15}$$

where $u(x_i, x_j)$ denotes the estimated covariance associated with x_i and x_j.

An analysis is necessary to identify the contribution of each source of errors from the sub-systems to the overall measurement uncertainty of the whole system. From this analysis, the major source of uncertainty can be determined. Depending on the accuracy required, appropriate adjustments in the set-up of the LISM apparatus can be made to reduce the uncertainty, and thus to improve the accuracy and repeatability of the measurements.

5.4.2 Methodology for the Estimation of Uncertainties

The uncertainties in position measurement and orientation measurement are estimated in the following examinations, with the assumption that the input estimates are uncorrelated.

5.4.2.1 Uncertainties in Position Measurement

The approach to uncertainty calculation is based on assuming that all the geometrical errors are compensated using technique shown in Section 5.3. With this assumption, the position of the retroreflector can be determined by translating the co-ordinate frame of the laser beam to each mirror in turn and rotating the frame about a particular axis by appropriate angles. This is shown in Fig. 5.6.

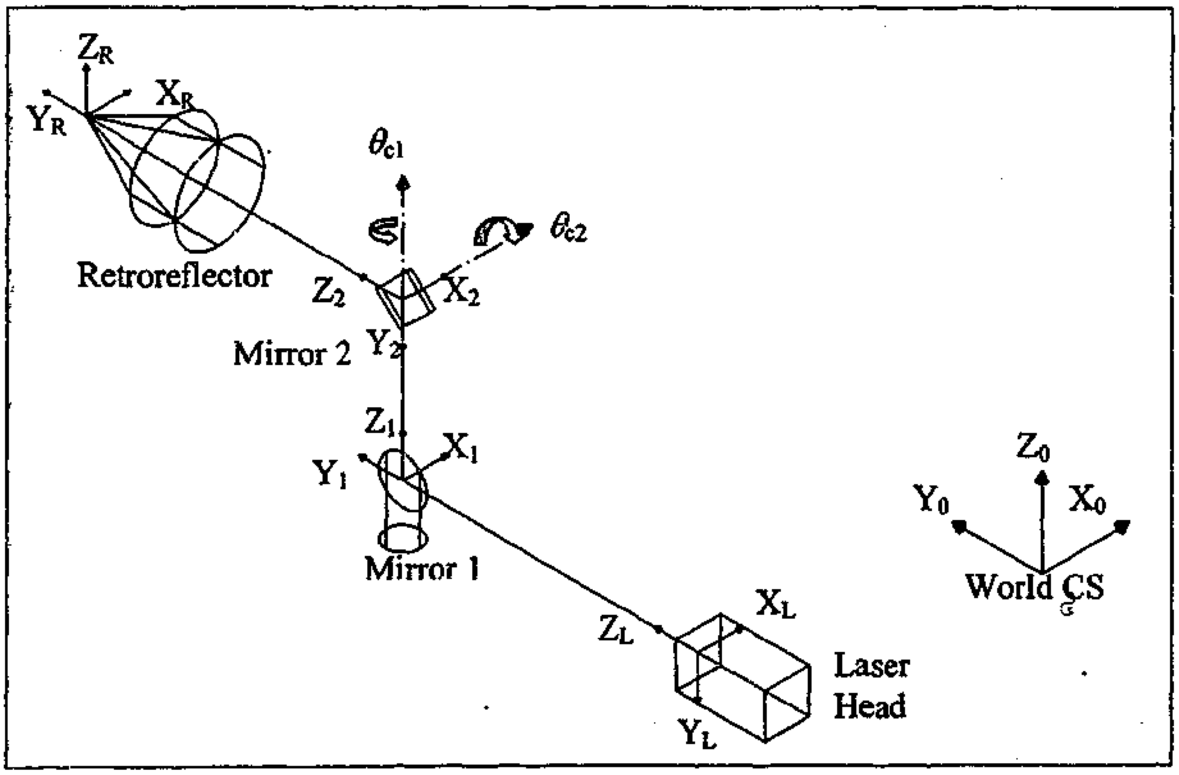


Figure 5.6: Kinematic model of the experimental LISM apparatus for uncertainties calculation

With this method, the position of the retroreflector with respect to the world reference can be written as:

$$ {}^{0}A_R = A_0 A_1 A_2 A_3 \qquad (5.16) $$

where $A_0 = [transl(a_{XL}, a_{YL}, a_{ZL})][Rotx(-90°)]$

$A_1 = [transl(z_L; a_{Z1})][Rotx(90°)]$

$A_2 = [transl(z_1; a_{Z2})][Rotz(\vartheta_{c1})][Rotx(-90° + 2\vartheta_{c2})]$

$A_R = [transl(z_2; a_{ZR})][transl(x_2; a_{XR})][transl(y_2; a_{YR})]$

The angular displacement θ_{c2} is multiplied by 2 as the laser beam is rotated by twice the angular displacement of motor 2. From the ${}^{0}A_R$ matrix given by Eq. 5.16, the x, y, and z Cartesian positions of the retroreflector are:

$$ x = a_{XR}\cos(\vartheta_{c1}) - a_{YR}\sin(\vartheta_{c1})\sin(2\vartheta_{c2}) - a_{ZR}\sin(\vartheta_{c1})\cos(2\vartheta_{c2}) + a_{XL} \qquad (5.17) $$

$$ y = a_{XR}\sin(\vartheta_{c1}) + a_{YR}\cos(\vartheta_{c1})\sin(2\vartheta_{c2}) + a_{ZR}\cos(\vartheta_{c1})\cos(2\vartheta_{c2}) + a_{Z1} + a_{YL} \qquad (5.18) $$

$$ z = -a_{YR}\cos(2\vartheta_{c2}) + a_{ZR}\sin(2\vartheta_{c2}) + a_{Z2} + a_{ZL} \qquad (5.19) $$

By expressing the standard uncertainty of measurements calculated using Eqs 5.17 to 5.19, the variance of the retroreflector's position can be calculated with the following equations:

$$ \begin{aligned} u^2(x) = & \left[\frac{\delta x}{\delta \vartheta_{c1}}\right]^2 u^2(\vartheta_{c1}) + \left[\frac{\delta x}{\delta \vartheta_{c2}}\right]^2 u^2(\vartheta_{c2}) + \left[\frac{\delta x}{\delta \Delta d_1}\right]^2 u^2(\Delta d_1) + \\ & \left[\frac{\delta x}{\delta \Delta d_2}\right]^2 u^2(\Delta d_2) + \left[\frac{\delta x}{\delta l_{radial}}\right]^2 u^2(l_{radial}) \\ = & \left[\begin{array}{l} 2\Delta d_1 \dfrac{\sin(\vartheta_{c1})}{\cos^2(\vartheta_{c1})}\cos(2\vartheta_{c2}) + \Delta d_2\cos(\vartheta_{c1})\dfrac{\sin(2\vartheta_{c2})}{\cos(2\vartheta_{c2})} \\ + \Delta d_2\cos(q_1)\sin(2q_2) - l_{radial}\cos(q_1)\cos(2q_2) \end{array}\right]^2 u^2(\vartheta_{c1}) + \\ & \left[\begin{array}{l} -2\Delta d_1 \dfrac{\sin^2(\vartheta_{c1})}{\cos(\vartheta_{c1})}\sin(2\vartheta_{c2}) + 2\Delta d_2 \dfrac{\sin(\vartheta_{c1})}{\cos^2(2\vartheta_{c2})} \\ + 2\Delta d_2\sin(\vartheta_{c1})\cos(2\vartheta_{c2}) + 2l_{radial}\sin(\vartheta_{c1})\sin(2\vartheta_{c2}) \end{array}\right]^2 u^2(\vartheta_{c2}) + \qquad (5.20) \\ & \left[1 + \frac{\sin^2(\vartheta_{c1})}{\cos(\vartheta_{c1})}\cos(2\vartheta_{c2})\right]^2 u^2(\Delta d_1) + \\ & \left[\sin(\vartheta_{c1})\frac{\sin(2\vartheta_{c2})}{\cos(2\vartheta_{c2})} + \sin(\vartheta_{c1})\sin(2\vartheta_{c2})\right]^2 u^2(\Delta d_2) + \\ & [-\sin(\vartheta_{c1})\cos(2\vartheta_{c2})]^2 u^2(l_{radial}) \end{aligned} $$

$$
\begin{aligned}
u^2(y) = {} & \left[\frac{\delta y}{\delta \vartheta_{c1}}\right]^2 u^2(\vartheta_{c1}) + \left[\frac{\delta y}{\delta \vartheta_{c2}}\right]^2 u^2(\vartheta_{c2}) + \left[\frac{\delta y}{\delta \Delta d_1}\right]^2 u^2(\Delta d_1) + \\
& \left[\frac{\delta y}{\delta \Delta d_2}\right]^2 u^2(\Delta d_2) + \left[\frac{\delta y}{\delta l_{radial}}\right]^2 u^2(l_{radial}) \\
= {} & \left[\begin{array}{l} \dfrac{\Delta d_1}{\cos^2(\vartheta_{c1})} - \Delta d_1 \cos(\vartheta_{c1})\cos(2\vartheta_{c2}) \\ + \Delta d_2 \sin(\vartheta_{c1}) \dfrac{\sin(2\vartheta_{c2})}{\cos(2\vartheta_{c2})} + \Delta d_2 \sin(q_1)\sin(2q_2) \\ - l_{radial} \sin(q_1)\cos(2q_2) \\ \end{array}\right]^2 u^2(\vartheta_{c1}) + \\
& \left[\begin{array}{l} 2\Delta d_1 \sin(\vartheta_{c1})\sin(2\vartheta_{c2}) - 2\Delta d_2 \dfrac{\cos(\vartheta_{c1})}{\cos^2(2\vartheta_{c2})} \\ - 2\Delta d_2 \cos(\vartheta_{c1})\cos(2\vartheta_{c1}) - 2l_{radial} \cos(\vartheta_{c1})\sin(2\vartheta_{c2}) \end{array}\right]^2 u^2(\vartheta_{c2}) + \qquad (5.21) \\
& \left[\frac{\sin(\vartheta_{c1})}{\cos(\vartheta_{c1})} - \sin(\vartheta_{c1})\cos(2\vartheta_{c2})\right]^2 u^2(\Delta d_1) + \\
& \left[-\cos(\vartheta_{c1}) \frac{\sin(2\vartheta_{c2})}{\cos(2\vartheta_{c2})} - \cos(\vartheta_{c1})\sin(2\vartheta_{c2})\right]^2 u^2(\Delta d_2) + \\
& [\cos(\vartheta_{c1})\cos(2\vartheta_{c2})]^2 u^2(l_{radial})
\end{aligned}
$$

$$
\begin{aligned}
u^2(z) = {} & \left[\frac{\delta z}{\delta \vartheta_{c1}}\right]^2 u^2(\vartheta_{c1}) + \left[\frac{\delta z}{\delta \vartheta_{c2}}\right]^2 u^2(\vartheta_{c2}) + \left[\frac{\delta z}{\delta \Delta d_1}\right]^2 u^2(\Delta d_1) + \\
& \left[\frac{\delta z}{\delta \Delta d_2}\right]^2 u^2(\Delta d_2) + \left[\frac{\delta z}{\delta l_{radial}}\right]^2 u^2(l_{radial}) \\
= {} & \left[\frac{-\Delta d_1}{\cos^2(\vartheta_{c1})} \sin(2\vartheta_{c2})\right]^2 u^2(\vartheta_{c1}) + \qquad (5.22) \\
& \left[\begin{array}{l} -2\Delta d_1 \dfrac{\sin(\vartheta_{c1})}{\cos(\vartheta_{c1})} \cos(2\vartheta_{c2}) - 2\Delta d_2 \dfrac{\sin(2\vartheta_{c2})}{\cos^2(2\vartheta_{c2})} \\ -2\Delta d_2 \sin(2\vartheta_{c2}) + 2l_{radial} \cos(2\vartheta_{c2}) \end{array}\right]^2 u^2(\vartheta_{c2}) + \\
& \left[-\frac{\sin(\vartheta_{c1})}{\cos(\vartheta_{c1})} \sin(2\vartheta_{c2})\right]^2 u^2(\Delta d_1) + \left[1 - \frac{\sin^2(2\vartheta_{\vartheta 2})}{\cos(2\vartheta_{c2})}\right]^2 u^2(\Delta d_2) \\
& [\sin(2\vartheta_{c2})]^2 u^2(l_{radial})
\end{aligned}
$$

where $u(\theta_{c1})$ and $u(\theta_{c2})$ are uncertainties of the angular displacements of the motors, $u(\Delta d_1)$ and $u(\Delta d_2)$ are uncertainties of the beam offset measured by the PSD that take into account the errors caused by the PSD and the retroreflector errors, $u(l_{radial})$ is the uncertainty of the laser interferometer.

The standard uncertainty of x, y, and z measurements can be determined by the square root of Eqs. 5.20 to 5.22 as follows:

$$u(i) = \sqrt{u^2(i)} \qquad (i = x, y, z) \qquad (5.23)$$

It must be noted that the uncertainties of positions determined using this approach are dependent on the instantaneous angular displacement of the motors and the displacement of the laser beam. Tables 5.2 to 5.4 tabulate the uncertainty obtained using Eq. 5.23 with various angular displacements of the motors. The results were calculated based on the assumption that the laser beam is hitting the centre of the retroreflector and the target is located 600mm away from the last mirror.

From Table 5.2 to 5.4, it can be observed that the coefficients for $u(\theta_{c1})$ and $u(\theta_{c2})$ are the dominant variables in the overall uncertainties of the position measurements. With the present motor accuracy, the maximum uncertainty is 0.51mm for x, 0.69mm for y, 0.95mm for z. To improve the uncertainty, motors and PSD with higher accuracy are required. With the utilisation of motors with accuracy of 0.003° and PSD with accuracy of 0.01mm, the maximum uncertainty can be reduced to 0.04mm for x, 0.05mm for y, 0.06mm for z. This is shown in Table 5.5 to 5.7.

Table 5.2: Uncertainty of retroreflector x co-ordinate at 4 different motors configurations

	$\delta x/\delta \Delta d1$	$\delta x/\delta \Delta d2$	$\delta x/\delta l_{radial}$	$\delta x/\delta \theta_{c1}$	$\delta x/\delta \theta_{c2}$
$\theta_{c1} = 0^\circ$	1	0	0	-600	0
$\theta_{c2} = 0^\circ$					
l_{radial} = 600 mm	$(\delta x/\delta \Delta d1)^2$	$(\delta x/\delta \Delta d2)^2$	$(\delta x/\delta l_{radial})^2$	$(\delta x/\delta \theta_{c1})^2$	$(\delta x/\delta \theta_{c2})^2$
$\Delta d1 = 0$	1	0	0	360000	0
$\Delta d2 = 0$					
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.75965E-08	6.1685E-07
u(x) (mm)	0.164726				
	$\delta x/\delta \Delta d1$	$\delta x/\delta \Delta d2$	$\delta x/\delta l_{radial}$	$\delta x/\delta \theta_{c1}$	$\delta x/\delta \theta_{c2}$
$\theta_{c1} = 45^\circ$	1.5	1.207107	-0.5	-300	600
$\theta_{c2} = 24.5^\circ$					
l_{radial} = 600 mm	$(\delta x/\delta \Delta d1)^2$	$(\delta x/\delta \Delta d2)^2$	$(\delta x/\delta l_{radial})^2$	$(\delta x/\delta \theta_{c1})^2$	$(\delta x/\delta \theta_{c2})^2$
$\Delta d1 = 0$	2.25	1.457107	0.25	90000	360000
$\Delta d2 = 0$					
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.75965E-08	6.1685E-07
u(x) (mm)	0.513246				
	$\delta x/\delta \Delta d1$	$\delta x/\delta \Delta d2$	$\delta x/\delta l_{radial}$	$\delta x/\delta \theta_{c1}$	$\delta x/\delta \theta_{c2}$
$\theta_{c1} = 45^\circ$	1	0	0	-424.2640687	0
$\theta_{c2} = 0^\circ$					
l_{radial} = 600 mm	$(\delta x/\delta \Delta d1)^2$	$(\delta x/\delta \Delta d2)^2$	$(\delta x/\delta l_{radial})^2$	$(\delta x/\delta \theta_{c1})^2$	$(\delta x/\delta \theta_{c2})^2$
$\Delta d1 = 0$	1	0	0	180000	0
$\Delta d2 = 0$					
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.75965E-08	6.1685E-07
u(x) (mm)	0.136262				
	$\delta x/\delta \Delta d1$	$\delta x/\delta \Delta d2$	$\delta x/\delta l_{radial}$	$\delta x/\delta \theta_{c1}$	$\delta x/\delta \theta_{c2}$
$\theta_{c1} = 0^\circ$	1.707107	0	-0.707106781	-424.2640687	0
$\theta_{c2} = 24.5^\circ$					
l_{radial} = 600 mm	$(\delta x/\delta \Delta d1)^2$	$(\delta x/\delta \Delta d2)^2$	$(\delta x/\delta l_{radial})^2$	$(\delta x/\delta \theta_{c1})^2$	$(\delta x/\delta \theta_{c2})^2$
$\Delta d1 = 0$	2.914214	0	0.5	180000	0
$\Delta d2 = 0$					
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.75965E-08	6.1685E-07
u(x) (mm)	0.194191				

Table 5.3: Uncertainty of retroreflector y co-ordinate at 4 different motors configurations

	$\delta y/\delta\Delta d1$	$\delta y/\delta\Delta d2$	$\delta y/\delta l_{radial}$	$\delta y/\delta\theta_{c1}$	$\delta y/\delta\theta_{c2}$
θ_{c1} = 0° θ_{c2} = 0°	0	0	1	0	0
l_{radial} = 600 mm	$(\delta y/\delta\Delta d1)^2$	$(\delta y/\delta\Delta d2)^2$	$(\delta y/\delta l_{radial})^2$	$(\delta y/\delta\theta_{c1})^2$	$(\delta y/\delta\theta_{c2})^2$
$\Delta d1$ = 0 $\Delta d2$ = 0	0	0	1	0	0
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.76E-08	6.17E-07
u(y) (mm)	0.001				
	$\delta y/\delta\Delta d1$	$\delta y/\delta\Delta d2$	$\delta y/\delta l_{radial}$	$\delta y/\delta\theta_{c1}$	$\delta y/\delta\theta_{c2}$
θ_{c1} = 45° θ_{c2} = 45°	0.5	-1.20711	0.5	-300	-600
l_{radial} = 600 mm	$(\delta y/\delta\Delta d1)^2$	$(\delta y/\delta\Delta d2)^2$	$(\delta y/\delta l_{radial})^2$	$(\delta y/\delta\theta_{c1})^2$	$(\delta y/\delta\theta_{c2})^2$
$\Delta d1$ = 0 $\Delta d2$ = 0	0.25	1.457107	0.25	90000	360000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.76E-08	6.17E-07
u(y) (mm)	0.493377				
	$\delta y/\delta\Delta d1$	$\delta y/\delta\Delta d2$	$\delta y/\delta l_{radial}$	$\delta y/\delta\theta_{c1}$	$\delta y/\delta\theta_{c2}$
θ_{c1} = 45° θ_{c2} = 0°	0	-1.70711	0.707106781	0	-848.528
l_{radial} = 600 mm	$(\delta y/\delta\Delta d1)^2$	$(\delta y/\delta\Delta d2)^2$	$(\delta y/\delta l_{radial})^2$	$(\delta y/\delta\theta_{c1})^2$	$(\delta y/\delta\theta_{c2})^2$
$\Delta d1$ = 0 $\Delta d2$ = 0	0	2.914214	0.5	0	720000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.76E-08	6.17E-07
u(y) (mm)	0.68795				
	$\delta y/\delta\Delta d1$	$\delta y/\delta\Delta d2$	$\delta y/\delta l_{radial}$	$\delta y/\delta\theta_{c1}$	$\delta y/\delta\theta_{c2}$
θ_{c1} = 0° θ_{c2} = 24.5°	0.292893	0	0.707106781	-424.264	0
l_{radial} = 600 mm	$(\delta y/\delta\Delta d1)^2$	$(\delta y/\delta\Delta d2)^2$	$(\delta y/\delta l_{radial})^2$	$(\delta y/\delta\theta_{c1})^2$	$(\delta y/\delta\theta_{c2})^2$
$\Delta d1$ = 0 $\Delta d2$ = 0	0.085786	0	0.5	180000	0
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.76E-08	6.17E-07
u(y) (mm)	0.097086				

Table 5.4: Uncertainty of retroreflector z co-ordinate at 4 different motors configurations

	$\delta z/\delta \Delta d1$	$\delta z/\delta \Delta d2$	$\delta z/\delta l_{radial}$	$\delta z/\delta \theta_{c1}$	$\delta z/\delta \theta_{c2}$
$\theta_{c1} = 0°$ $\theta_{c2} = 0°$	0	1	0	0	1200
$l_{radial} = 600$ mm	$(\delta z/\delta \Delta d1)^2$	$(\delta z/\delta \Delta d2)^2$	$(\delta z/\delta l_{radial})^2$	$(\delta z/\delta \theta_{c1})^2$	$(\delta z/\delta \theta_{c2})^2$
$\Delta d1 = 0$ $\Delta d2 = 0$	0	1	0	0	1440000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.75965E-08	6.17E-07
u(z) (mm)	**0.947768**				

	$\delta z/\delta \Delta d1$	$\delta z/\delta \Delta d2$	$\delta z/\delta l_{radial}$	$\delta z/\delta \theta_{c1}$	$\delta z/\delta \theta_{c2}$
$\theta_{c1} = 45°$ $\theta_{c2} = 24.5°$	-0.707107	0.292893	0.707106781	0	848.5281
$l_{radial} = 600$ mm	$(\delta z/\delta \Delta d1)^2$	$(\delta z/\delta \Delta d2)^2$	$(\delta z/\delta l_{radial})^2$	$(\delta z/\delta \theta_{c1})^2$	$(\delta z/\delta \theta_{c2})^2$
$\Delta d1 = 0$ $\Delta d2 = 0$	0.5	0.085786	0.5	0	720000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.75965E-08	6.17E-07
u(z) (mm)	**0.670813**				

	$\delta z/\delta \Delta d1$	$\delta z/\delta \Delta d2$	$\delta z/\delta l_{radial}$	$\delta z/\delta \theta_{c1}$	$\delta z/\delta \theta_{c2}$
$\theta_{c1} = 45°$ $\theta_{c2} = 0°$	0	0.292893	0.707106781	0	848.5281
$l_{radial} = 600$ mm	$(\delta z/\delta \Delta d1)^2$	$(\delta z/\delta \Delta d2)^2$	$(\delta z/\delta l_{radial})^2$	$(\delta z/\delta \theta_{c1})^2$	$(\delta z/\delta \theta_{c2})^2$
$\Delta d1 = 0$ $\Delta d2 = 0$	0	0.085786	0.5	0	720000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.75965E-08	6.17E-07
u(z) (mm)	**0.667076**				

	$\delta z/\delta \Delta d1$	$\delta z/\delta \Delta d2$	$\delta z/\delta l_{radial}$	$\delta z/\delta \theta_{c1}$	$\delta z/\delta \theta_{c2}$
$\theta_{c1} = 0°$ $\theta_{c2} = 24.5°$	0	1	0	0	1200
$l_{radial} = 600$ mm	$(\delta z/\delta \Delta d1)^2$	$(\delta z/\delta \Delta d2)^2$	$(\delta z/\delta l_{radial})^2$	$(\delta z/\delta \theta_{c1})^2$	$(\delta z/\delta \theta_{c2})^2$
$\Delta d1 = 0$ $\Delta d2 = 0$	0	1	0	0	1440000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.01	0.01	0.000001	4.75965E-08	6.17E-07
u(z) (mm)	**0.947768**				

Table 5.5: Uncertainty of retroreflector x co-ordinate at 4 different motors configurations with higher instruments' accuracy

	$\delta x/\delta\Delta d1$	$\delta x/\delta\Delta d2$	$\delta x/\delta l_{radial}$	$\delta x/\delta\theta_{c1}$	$\delta x/\delta\theta_{c2}$
θ_{c1} = 0° θ_{c2} = 0°	1	0	0	-600	0
l_{radial} = 600 mm	$(\delta x/\delta\Delta d1)^2$	$(\delta x/\delta\Delta d2)^2$	$(\delta x/\delta l_{radial})^2$	$(\delta x/\delta\theta_{c1})^2$	$(\delta x/\delta\theta_{c2})^2$
Δd1 = 0 Δd2 = 0	1	0	0	360000	0
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74156E-09	2.74156E-09
u(x) (mm)	**0.032969**				
	$\delta x/\delta\Delta d1$	$\delta x/\delta\Delta d2$	$\delta x/\delta l_{radial}$	$\delta x/\delta\theta_{c1}$	$\delta x/\delta\theta_{c2}$
θ_{c1} = 45° θ_{c2} = 24.5°	1.5	1.207107	-0.5	-300	600
l_{radial} = 600 mm	$(\delta x/\delta\Delta d1)^2$	$(\delta x/\delta\Delta d2)^2$	$(\delta x/\delta l_{radial})^2$	$(\delta x/\delta\theta_{c1})^2$	$(\delta x/\delta\theta_{c2})^2$
Δd1 = 0 Δd2 = 0	2.25	1.457107	0.25	90000	360000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74156E-09	2.74156E-09
u(x) (mm)	**0.040058**				
	$\delta x/\delta\Delta d1$	$\delta x/\delta\Delta d2$	$\delta x/\delta l_{radial}$	$\delta x/\delta\theta_{c1}$	$\delta x/\delta\theta_{c2}$
θ_{c1} = 45° θ_{c2} = 0°	1	0	0	-424.2640687	0
l_{radial} = 600 mm	$(\delta x/\delta\Delta d1)^2$	$(\delta x/\delta\Delta d2)^2$	$(\delta x/\delta l_{radial})^2$	$(\delta x/\delta\theta_{c1})^2$	$(\delta x/\delta\theta_{c2})^2$
Δd1 = 0 Δd2 = 0	1	0	0	180000	0
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74156E-09	2.74156E-09
u(x) (mm)	**0.024361**				
	$\delta x/\delta\Delta d1$	$\delta x/\delta\Delta d2$	$\delta x/\delta l_{radial}$	$\delta x/\delta\theta_{c1}$	$\delta x/\delta\theta_{c2}$
θ_{c1} = 0° θ_{c2} = 24.5°	1.707107	0	-0.707106781	-424.2640687	0
l_{radial} = 600 mm	$(\delta x/\delta\Delta d1)^2$	$(\delta x/\delta\Delta d2)^2$	$(\delta x/\delta l_{radial})^2$	$(\delta x/\delta\theta_{c1})^2$	$(\delta x/\delta\theta_{c2})^2$
Δd1 = 0 Δd2 = 0	2.914214	0	0.5	180000	0
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74156E-09	2.74156E-09
u(x) (mm)	**0.028025**				

Table 5.6: Uncertainty of retroreflector y co-ordinate at 4 different motors configurations with higher instruments' accuracy

	$\delta y/\delta\Delta d1$	$\delta y/\delta\Delta d2$	$\delta y/\delta l_{radial}$	$\delta y/\delta\theta_{c1}$	$\delta y/\delta\theta_{c2}$
$\theta_{c1} = 0^\circ$	0	0	1	0	0
$\theta_{c2} = 0^\circ$					
l_{radial} = 600 mm	$(\delta y/\delta\Delta d1)^2$	$(\delta y/\delta\Delta d2)^2$	$(\delta y/\delta l_{radial})^2$	$(\delta y/\delta\theta_{c1})^2$	$(\delta y/\delta\theta_{c2})^2$
$\Delta d1 = 0$	0	0	1	0	0
$\Delta d2 = 0$					
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74E-09	2.74E-09
u(y) (mm)	**0.001**				
	$\delta y/\delta\Delta d1$	$\delta y/\delta\Delta d2$	$\delta y/\delta l_{radial}$	$\delta y/\delta\theta_{c1}$	$\delta y/\delta\theta_{c2}$
$\theta_{c1} = 45^\circ$	0.5	-1.20711	0.5	-300	-600
$\theta_{c2} = 24.5^\circ$					
l_{radial} = 600 mm	$(\delta y/\delta\Delta d1)^2$	$(\delta y/\delta\Delta d2)^2$	$(\delta y/\delta l_{radial})^2$	$(\delta y/\delta\theta_{c1})^2$	$(\delta y/\delta\theta_{c2})^2$
$\Delta d1 = 0$	0.25	1.457107	0.25	90000	360000
$\Delta d2 = 0$					
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74E-09	2.74E-09
u(y) (mm)	**0.037479**				
	$\delta y/\delta\Delta d1$	$\delta y/\delta\Delta d2$	$\delta y/\delta l_{radial}$	$\delta y/\delta\theta_{c1}$	$\delta y/\delta\theta_{c2}$
$\theta_{c1} = 45^\circ$	0	-1.70711	0.707106781	0	-848.528
$\theta_{c2} = 0^\circ$					
l_{radial} = 600 mm	$(\delta y/\delta\Delta d1)^2$	$(\delta y/\delta\Delta d2)^2$	$(\delta y/\delta l_{radial})^2$	$(\delta y/\delta\theta_{c1})^2$	$(\delta y/\delta\theta_{c2})^2$
$\Delta d1 = 0$	0	2.914214	0.5	0	720000
$\Delta d2 = 0$					
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74E-09	2.74E-09
u(y) (mm)	**0.047601**				
	$\delta y/\delta\Delta d1$	$\delta y/\delta\Delta d2$	$\delta y/\delta l_{radial}$	$\delta y/\delta\theta_{c1}$	$\delta y/\delta\theta_{c2}$
$\theta_{c1} = 0^\circ$	0.292893	0	0.707106781	-424.264	0
$\theta_{c2} = 24.5^\circ$					
l_{radial} = 600 mm	$(\delta y/\delta\Delta d1)^2$	$(\delta y/\delta\Delta d2)^2$	$(\delta y/\delta l_{radial})^2$	$(\delta y/\delta\theta_{c1})^2$	$(\delta y/\delta\theta_{c2})^2$
$\Delta d1 = 0$	0.085786	0	0.5	180000	0
$\Delta d2 = 0$					
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74E-09	2.74E-09
u(y) (mm)	**0.022418**				

Table 5.7: Uncertainty of retroreflector z co-ordinate at 4 different motors configurations with higher instruments' accuracy

	$\delta z/\delta \Delta d1$	$\delta z/\delta \Delta d2$	$\delta z/\delta l_{radial}$	$\delta z/\delta \theta_{c1}$	$\delta z/\delta \theta_{c2}$
$\theta_{c1} = 0^\circ$ $\theta_{c2} = 0^\circ$	0	1	0	0	1200
$l_{radial} = 600$ mm	$(\delta z/\delta \Delta d1)^2$	$(\delta z/\delta \Delta d2)^2$	$(\delta z/\delta l_{radial})^2$	$(\delta z/\delta \theta_{c1})^2$	$(\delta z/\delta \theta_{c2})^2$
$\Delta d1 = 0$ $\Delta d2 = 0$	0	1	0	0	1440000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74156E-09	2.74E-09
u(z) (mm)	0.063623				
	$\delta z/\delta \Delta d1$	$\delta z/\delta \Delta d2$	$\delta z/\delta l_{radial}$	$\delta z/\delta \theta_{c1}$	$\delta z/\delta \theta_{c2}$
$\theta_{c1} = 45^\circ$ $\theta_{c2} = 24.5^\circ$	-0.707107	0.292893	0.707106781	0	848.5281
$l_{radial} = 600$ mm	$(\delta z/\delta \Delta d1)^2$	$(\delta z/\delta \Delta d2)^2$	$(\delta z/\delta l_{radial})^2$	$(\delta z/\delta \theta_{c1})^2$	$(\delta z/\delta \theta_{c2})^2$
$\Delta d1 = 0$ $\Delta d2 = 0$	0.5	0.085786	0.5	0	720000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74156E-09	2.74E-09
u(z) (mm)	0.045089				
	$\delta z/\delta \Delta d1$	$\delta z/\delta \Delta d2$	$\delta z/\delta l_{radial}$	$\delta z/\delta \theta_{c1}$	$\delta z/\delta \theta_{c2}$
$\theta_{c1} = 45^\circ$ $\theta_{c2} = 0^\circ$	0	0.292893	0.707106781	0	848.5281
$l_{radial} = 600$ mm	$(\delta z/\delta \Delta d1)^2$	$(\delta z/\delta \Delta d2)^2$	$(\delta z/\delta l_{radial})^2$	$(\delta z/\delta \theta_{c1})^2$	$(\delta z/\delta \theta_{c2})^2$
$\Delta d1 = 0$ $\Delta d2 = 0$	0	0.085786	0.5	0	720000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74156E-09	2.74E-09
u(z) (mm)	0.044531				
	$\delta z/\delta \Delta d1$	$\delta z/\delta \Delta d2$	$\delta z/\delta l_{radial}$	$\delta z/\delta \theta_{c1}$	$\delta z/\delta \theta_{c2}$
$\theta_{c1} = 0^\circ$ $\theta_{c2} = 24.5^\circ$	0	1	0	0	1200
$l_{radial} = 600$ mm	$(\delta z/\delta \Delta d1)^2$	$(\delta z/\delta \Delta d2)^2$	$(\delta z/\delta l_{radial})^2$	$(\delta z/\delta \theta_{c1})^2$	$(\delta z/\delta \theta_{c2})^2$
$\Delta d1 = 0$ $\Delta d2 = 0$	0	1	0	0	1440000
	$u^2(\Delta d1)$	$u^2(\Delta d2)$	$u^2(l_{radial})$	$u^2(\theta_{c1})$	$u^2(\theta_{c2})$
	0.0001	0.0001	0.000001	2.74156E-09	2.74E-09
u(z) (mm)	0.063623				

5.4.2.2 Uncertainties in Orientation Measurement

For the dual PSD-based orientation measurement approach, the uncertainty can be estimated by expressing the uncertainty of the Orientation Measurement Formulation (OMF) obtained using Eqs. 4.6 to 4.13 as follows [61].

From Eq. 4.6, let $n_1 = \tan\varphi = \dfrac{D_{3x} - D_{2x}}{2l_1 - l_2 + l_3}$. The variance of n_1 can be calculated using the following equation:

$$
\begin{aligned}
u^2(n_1) &= \left[\frac{\delta n_1}{\delta D_{2x}}\right]^2 u^2(D_{2x}) + \left[\frac{\delta n_1}{\delta D_{3x}}\right]^2 u^2(D_{3x}) + \left[\frac{\delta n_1}{\delta l_1}\right]^2 u^2(l_1) + \\
&\quad \left[\frac{\delta n_1}{\delta l_2}\right]^2 u^2(l_2) + \left[\frac{\delta n_1}{\delta l_3}\right]^2 u^2(l_3) \\
&= \left[\frac{-1}{2l_1 - l_2 + l_3}\right]^2 u^2(D_{2x}) + \left[\frac{1}{2l_1 - l_2 + l_3}\right]^2 u^2(D_{3x}) + \\
&\quad \left[\frac{-2(D_{3x} - D_{2x})}{(2l_1 - l_2 + l_3)^2}\right]^2 u^2(l_1) + \left[\frac{(D_{3x} - D_{2x})}{(2l_1 - l_2 + l_3)^2}\right]^2 u^2(l_2) + \\
&\quad \left[\frac{-(D_{3x} - D_{2x})}{(2l_1 - l_2 + l_3)^2}\right]^2 u^2(l_3)
\end{aligned}
\tag{5.24}
$$

From Eq. 4.7, let $n_2 = \tan\phi = \dfrac{D_{2y} + D_{3y}}{2l_1 - l_2 + l_3}\cos\varphi$. The variance of n_2 can be calculated using the following equation:

$$
\begin{aligned}
u^2(n_2) &= \left[\frac{\delta n_2}{\delta D_{2y}}\right]^2 u^2(D_{2y}) + \left[\frac{\delta n_2}{\delta D_{3y}}\right]^2 u^2(D_{3y}) + \left[\frac{\delta n_2}{\delta l_1}\right]^2 u^2(l_1) + \\
&\quad \left[\frac{\delta n_2}{\delta l_2}\right]^2 u^2(l_2) + \left[\frac{\delta n_2}{\delta l_3}\right]^2 u^2(l_3) + \left[\frac{\delta n_2}{\delta \varphi}\right]^2 u^2(\varphi) \\
&= \left[\frac{\cos\varphi}{2l_1 - l_2 + l_3}\right]^2 u^2(D_{2y}) + \left[\frac{\cos\varphi}{2l_1 - l_2 + l_3}\right]^2 u^2(D_{3y}) + \\
&\quad \left[\frac{-2(D_{2y} + D_{3y})\cos\varphi}{(2l_1 - l_2 + l_3)^2}\right]^2 u^2(l_1) + \left[\frac{(D_{2y} + D_{3y})\cos\varphi}{(2l_1 - l_2 + l_3)^2}\right]^2 u^2(l_2) + \\
&\quad \left[\frac{-(D_{2y} + D_{3y})\cos\varphi}{(2l_1 - l_2 + l_3)^2}\right]^2 u^2(l_3) + \left[\frac{-(D_{2y} + D_{3y})\sin\varphi}{2l_1 - l_2 + l_3}\right]^2 u^2(\varphi)
\end{aligned}
\tag{5.25}
$$

Similiarly, from Eq. 4.13, let $n_3 = \tan\rho = \frac{B_{2z}}{B_{2x}}$. The variance of n_3 can be calculated using the following equation:

$$
\begin{aligned}
u^2(n_3) = {} & \left[\frac{\delta n_3}{\delta D_{2x}}\right]^2 u^2(D_{2x}) + \left[\frac{\delta n_3}{\delta D_{2y}}\right]^2 u^2(D_{2y}) + \left[\frac{\delta n_3}{\delta l_1}\right]^2 u^2(l_1) + \\
& \left[\frac{\delta n_3}{\delta l_2}\right]^2 u^2(l_2) + \left[\frac{\delta n_3}{\delta \phi}\right]^2 u^2(\phi) + \left[\frac{\delta n_3}{\delta \varphi}\right]^2 u^2(\varphi) \\
= {} & \left[\frac{AD_{2x}}{B^2}\right]^2 u^2(D_{2x}) + \left[\frac{B\cos\phi - A\sin\varphi\sin\phi}{B^2}\right]^2 u^2(D_{2y}) + \\
& \left[\frac{B\dfrac{\sin\phi}{\cos\phi} - A(-\sin\varphi + \dfrac{\sin^2\phi}{\cos\phi}\tan\varphi)}{B^2}\right]^2 u^2(l_1) + \\
& \left[\frac{-B\dfrac{\sin\phi}{\cos\phi} - A(-\sin\varphi - \dfrac{\sin^2\phi}{\cos\phi}\tan\varphi)}{B^2}\right]^2 u^2(l_2) + \\
& \left[\frac{\begin{array}{l} B[-D_{2y}\sin\phi - (l_2 - l_1)\dfrac{\cos\phi}{\cos\varphi}] - A[D_{2y}\sin\varphi\cos\phi - \\ (l_2 - l_1)\tan\varphi\dfrac{2\sin\phi}{\cos^2\phi}] \end{array}}{B^2}\right]^2 u^2(\phi) + \\
& \left[\frac{\begin{array}{l} B[-(l_2 - l_1)\dfrac{\sin\phi\sin\varphi}{\cos^2\varphi}] - \\ A[D_{2x}\sin\varphi + (l_2 - l_1)\cos\varphi + D_{2y}\sin\phi\cos\varphi - \\ \quad (l_2 - l_1)\dfrac{\sin^2\phi}{\cos\phi\cos^2\varphi}] \end{array}}{B^2}\right]^2 u^2(\varphi)
\end{aligned}
\tag{5.26}
$$

$$A = D_{2y}\cos\phi - (l_2 - l_1)\frac{\sin\phi}{\cos\varphi} \tag{5.27}$$

$$B = -D_{2x}\cos\varphi + (l_2 - l_1)\sin\varphi + D_{2y}\sin\varphi\sin\phi - (l_2 - l_1)\frac{\sin^2\phi}{\cos\phi}\tan\varphi \tag{5.28}$$

The uncertainty of the angles can be obtained from the arctan of the square root of the variances given by Eqs. 5.24 to 5.26:

$$u(Angle) = \tan^{-1}[u(n_i)] \qquad (i = 1, 2, 3) \qquad (5.29)$$

where *Angle* represents φ, ϕ, and ρ.

It must be noted that the uncertainty in φ, which is the rotation of the Gimbal unit about the z-axis in Fig. 4.1, is dependent only on the geometry of the gimbal and the PSD measurements. However, the uncertainty in ϕ, which is the rotation of the gimbal about the x-axis in Fig. 4.1, is dependent on φ as well as the uncertainty in φ. Uncertainty in ρ, which is the rotation of the gimbal about the y-axis in Fig. 4.1, is dependent on the instantaneous values and uncertainties of both φ and ϕ. The uncertainty of the angles calculated based on $\varphi = 0$, $\phi = 0$ and a beam offset of 2mm is tabulated in Table 5.8.

From Table 5.8, it can be observed that the coefficients of $u(D_{2x})$, $u(D_{2y})$, $u(D_{3x})$, $u(D_{3y})$ are the dominant parameters in the overall uncertainties of the orientation measurements. These are measurements made by the PSDs, and thus they are equivalent to the uncertainty of the PSD measurements. The uncertainty is 0.07° for both yaw and pitch angles, and 5.71° for roll angle. Due to the high uncertainty associated with the calculation of roll angle, OMF is not effective in acquiring roll angle in the current set-up. Uncertainty for orientation measurement can be reduced by using PSD with a better accuracy. Furthermore, the uncertainty of roll angles was observed to be dependent on the beam offset as described in Section 4.5. Thus, uncertainty for the roll angle can be further improved by using a larger beam offset. Alignment tools to establish a beam offset of 12.8mm have been provided by Zygo. With the utilisation of PSD with an accuracy of 0.01mm and a beam offset of 12.8mm (D_{2x} = 6.4mm, D_{3x} = 6.4mm), the uncertainty can be reduced to 0.007° for both yaw and pitch angles and 0.090° for roll angle. This is shown in Table 5.9.

Table 5.8: Uncertainty of roll, pitch and yaw angles

	$\delta n_1/\delta D_{2x}$	$\delta n_1/\delta D_{3x}$	$\delta n_1/\delta l_1$	$\delta n_1/\delta l_2$	$\delta n_1/\delta l_3$	
D_{2x} = 1 mm D_{3x} = 1 mm	-8.33E-03	8.33E-03	0	0	0	
l_1 = 60 mm	$(\delta n_1/\delta D_{2x})^2$	$(\delta n_1/\delta D_{3x})^2$	$(\delta n_1/\delta l_1)^2$	$(\delta n_1/\delta l_2)^2$	$(\delta n_1/\delta l_3)^2$	
l_2 = 60 mm l_3 = 60 mm	6.94E-05	6.94E-05	0	0	0	
	$u^2(D_{2x})$	$u^2(D_{3x})$	$u^2(l_1)$	$u^2(l_2)$	$u^2(l_3)$	
	0.01	0.01	0.01	0.01	0.01	
$u(n_1)$	0.0011785					
$u(\varphi)$ (degree)	0.0675237					
	$\delta n_2/\delta D_{2y}$	$\delta n_2/\delta D_{3y}$	$\delta n_2/\delta l_1$	$\delta n_2/\delta l_2$	$\delta n_2/\delta l_3$	$\delta n_2/\delta \varphi$
D_{2y} = 0 D_{3y} = 0	8.33E-03	8.33E-03	0	0	0	0
l_1 = 60 mm	$(\delta n_2/\delta D_{2x})^2$	$(\delta n_2/\delta D_{3x})^2$	$(\delta n_2/\delta l_1)^2$	$(\delta n_2/\delta l_2)^2$	$(\delta n_2/\delta l_3)^2$	$(\delta n_2/\delta \varphi)^2$
l_2 = 60 mm l_3 = 60 mm	6.94E-05	6.94E-05	0	0	0	0
φ = 0	$u^2(D_{2y})$	$u^2(D_{3y})$	$u^2(l_1)$	$u^2(l_2)$	$u^2(l_3)$	$u^2(\varphi)$
	0.01	0.01	0.01	0.01	0.01	1.44E-06
$u(n_2)$	0.0011785					
$u(\phi)$ (degree)	0.0675237					
	$\delta n_3/\delta D_{2x}$	$\delta n_3/\delta D_{2y}$	$\delta n_3/\delta l_1$	$\delta n_3/\delta l_2$	$\delta n_3/\delta \phi$	$\delta n_3/\delta \varphi$
D_{2x} = 1 mm D_{2y} = 0	0	-1	0	0	0	0
l_1 = 60 mm	$(\delta n_1/\delta D_{2x})^2$	$(\delta n_1/\delta D_{2y})^2$	$(\delta n_1/\delta l_1)^2$	$(\delta n_1/\delta l_2)^2$	$(\delta n_1/\delta \phi)^2$	$(\delta n_3/\delta \varphi)^2$
l_2 = 60 mm l_3 = 60 mm	0	1	0	0	0	0
φ = 0	$u^2(D_{2x})$	$u^2(D_{3x})$	$u^2(l_1)$	$u^2(l_2)$	$u^2(\phi)$	$u^2(\varphi)$
ϕ = 0 A = 0 B = -1 mm	0.01	0.01	0.01	0.01	1.44E-06	1.44E-06
$u(n_3)$	0.1					
$u(\rho)$ (degree)	5.7105931					

Table 5.9: Uncertainty of roll, pitch and yaw angles with higher instruments' accuracy and larger beam offset

	$\delta n_1/\delta D_{2x}$	$\delta n_1/\delta D_{3x}$	$\delta n_1/\delta l_1$	$\delta n_1/\delta l_2$	$\delta n_1/\delta l_3$	
$D_{2x} = 6.4$ mm $D_{3x} = 6.4$ mm	-8.33E-03	8.33E-03	0	0	0	
$l_1 = 60$ mm	$(\delta n_1/\delta D_{2x})^2$	$(\delta n_1/\delta D_{3x})^2$	$(\delta n_1/\delta l_1)^2$	$(\delta n_1/\delta l_2)^2$	$(\delta n_1/\delta l_3)^2$	
$l_2 = 60$ mm $l_3 = 60$ mm	6.94E-05	6.94E-05	0	0	0	
	$u^2(D_{2x})$	$u^2(D_{3x})$	$u^2(l_1)$	$u^2(l_2)$	$u^2(l_3)$	
	0.0001	0.0001	0.0001	0.0001	0.0001	
$u(n_1)$	**0.0001179**					
$u(\varphi)$ (degree)	**0.0067524**					
	$\delta n_2/\delta D_{2y}$	$\delta n_2/\delta D_{3y}$	$\delta n_2/\delta l_1$	$\delta n_2/\delta l_2$	$\delta n_2/\delta l_3$	$\delta n_2/\delta \varphi$
$D_{2y} = 0$ $D_{3y} = 0$	8.33E-03	8.33E-03	0	0	0	0
$l_1 = 60$ mm	$(\delta n_2/\delta D_{2x})^2$	$(\delta n_2/\delta D_{3x})^2$	$(\delta n_2/\delta l_1)^2$	$(\delta n_2/\delta l_2)^2$	$(\delta n_2/\delta l_3)^2$	$(\delta n_2/\delta \varphi)^2$
$l_2 = 60$ mm $l_3 = 60$ mm	6.94E-05	6.94E-05	0	0	0	0
$\varphi = 0$	$u^2(D_{2y})$	$u^2(D_{3y})$	$u^2(l_1)$	$u^2(l_2)$	$u^2(l_3)$	$u^2(\varphi)$
	0.0001	0.0001	0.0001	0.0001	0.0001	1.44E-08
$u(n_2)$	**0.0001179**					
$u(\phi)$ (degree)	**0.0067524**					
	$\delta n_3/\delta D_{2x}$	$\delta n_3/\delta D_{2y}$	$\delta n_3/\delta l_1$	$\delta n_3/\delta l_2$	$\delta n_3/\delta \phi$	$\delta n_3/\delta \varphi$
$D_{2x} = 6.4$ mm $D_{2y} = 0$	0	-0.15625	0	0	0	0
$l_1 = 60$ mm	$(\delta n_1/\delta D_{2x})^2$	$(\delta n_1/\delta D_{2y})^2$	$(\delta n_1/\delta l_1)^2$	$(\delta n_1/\delta l_2)^2$	$(\delta n_1/\delta \phi)^2$	$(\delta n_3/\delta \varphi)^2$
$l_2 = 60$ mm $l_3 = 60$ mm	0	0.024414	0	0	0	0
$\varphi = 0$	$u^2(D_{2x})$	$u^2(D_{3x})$	$u^2(l_1)$	$u^2(l_2)$	$u^2(\phi)$	$u^2(\varphi)$
$\phi = 0$ $A = 0$ $B = -6.4$ mm	0.0001	0.0001	0.0001	0.0001	1.44E-08	1.44E-08
$u(n_3)$	**0.0015625**					
$u(\rho)$ (degree)	**0.0895246**					

5.5 Summary

In this chapter, the sources of errors associated with the LISM apparatus and the Gimbal unit have been identified. The establishment and calibration of the kinematic model of the LISM apparatus has been presented. An approach to analyse the overall uncertainties of the measurement made by the LISM apparatus and the Gimbal unit has also been presented. It is desirable that position or orientation measurements obtained have a very high accuracy, at least better than the accuracy of the robot manipulator to be measured in order to carry out calibration of the robot manipulator. This requires that the uncertainty of the measurements made by the LISM apparatus due to geometric and non-geometric or any other errors to be less than the accuracy desired. This knowledge of uncertainty and the analysis method allow the correct selection of individual sub-system devices for the LISM apparatus and the Gimbal unit in order to improve the measurement accuracy.

Next chapter presents the investigation of issues, requirements, and factors associated with the establishment of a methodology for closed-loop control of robot manipulators using the LISM apparatus.

Chapter 6

Investigation of Closed-loop Control of Robot Manipulators using LISM

6.1 Introduction

Robot manipulators have been used in many manufacturing industries. Due to an increase in complexity and higher quality requirements in production operations, a more accurate control of robot's end-effector is required. Various robot control architectures have been developed to allow the end-effector's position to be controlled so that a desired position can be reached or a desired trajectory can be followed with high precision and stability. However, there are limitations on the existing architectures such as the computational power requirement, the uncertainty of the robot structure, the incompatibility between joint-space and Cartesian space, mechanical hysteresis and friction, external disturbance, etc. [1, 4, 6, 23, 64, 65, 66, 67, 68]. In this chapter, a closed-loop control of the robot manipulator using the Laser Interferometry-based Sensing and Measuring (LISM) apparatus is proposed. This is a control methodology where the LISM apparatus is used to perform dynamic measurements of the robot's end-effector during robot manipulation to perform compensation or error adjustment on-line. The control algorithm for this methodology, together with the measurement and analysis techniques are described in the following sections. Experimental implementation of the methodology on an industrial robot manipulator is also presented. The results are analysed to examine the effectiveness of the methodology. This feedback of the actual Cartesian position improves the robot path accuracy and the robot approach toward a desired point without the need of complex dynamic model of the robot manipulator [69, 70, 71].

6.2 Principle of Robot Closed-loop Control Using LISM

A LISM Closed-Loop Control (CLC) is a control methodology where the LISM apparatus is used as a position sensor to measure the Cartesian position of the robot's end-effector. The LISM apparatus maintains tracking of the robot's end-effector while it is moving along a predefined path. During the robot motion, there may be wobbling and bending of the robot arm along the path of motion, which causes the position of the end-effector to deviate from the desired path. The LISM apparatus is implemented as a processing unit to determine the position error. Feedback of the position error from the LISM apparatus is used to provide a position compensation specification to the robot manipulator. This position error can be converted to compensation joint angles by the robot controller and fed into the robot's joint space controller as shown in Fig. 6.1. This feedback loop ensures a higher positioning accuracy as the error of the robot end-effector in Cartesian space can be detected using the LISM apparatus. Further, this methodology ensures that the deviation of the end-effector's travel is within a desired error-limit and thus improves the path accuracy of the robot end -effector.

6.3 Control Methodology

A flowchart of the control algorithm is provided in Fig. 6.2. Two different methods of commanding the robot have been investigated. The first method is based on the command of robot positions. The second method is by controlling the robot velocity based on the calculated position errors. In both methodologies, the control software first prompts the operator for the desired trajectory of the robot. This trajectory must consist of the starting point, the end point, and the other desired control points that the end-effector must reach. Straight line motion segments are used to join the start point to the first control point, from each control point to the next, and finally from the last control point to the end point. Fig. 6.3 shows the schematic description of the generated path. A number of via points, used for the verification of robot position along each motion segment, are generated and plotted along each motion segment. Fig. 6.4 shows the same generated motion segments with the inclusion of the target points (start, end, control and via points).

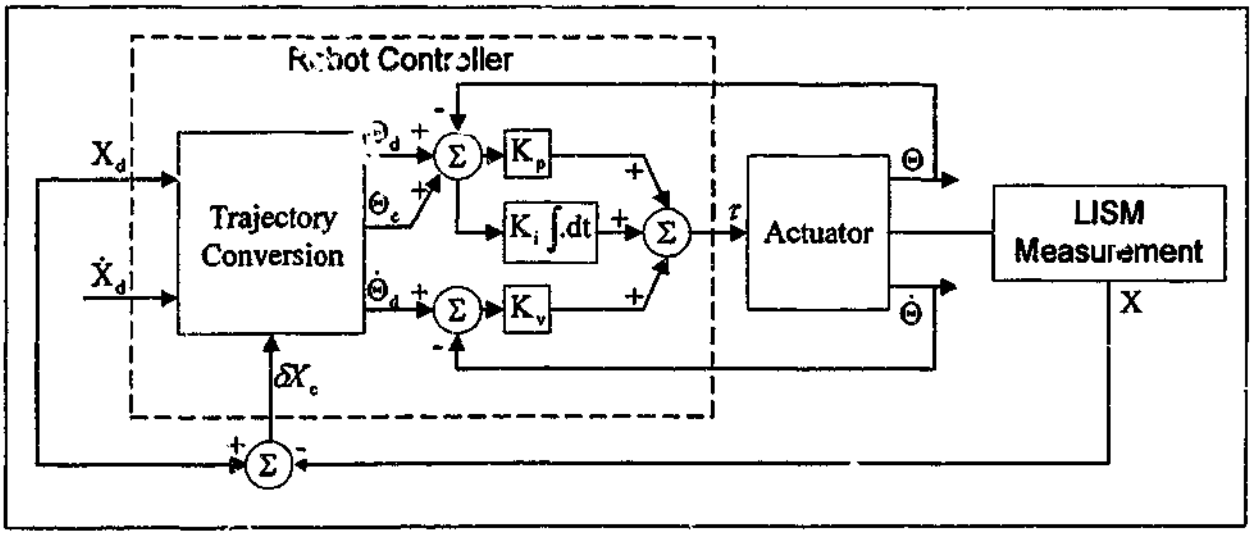


Figure 6.1: Block diagram of closed-loop control using LISM feedback

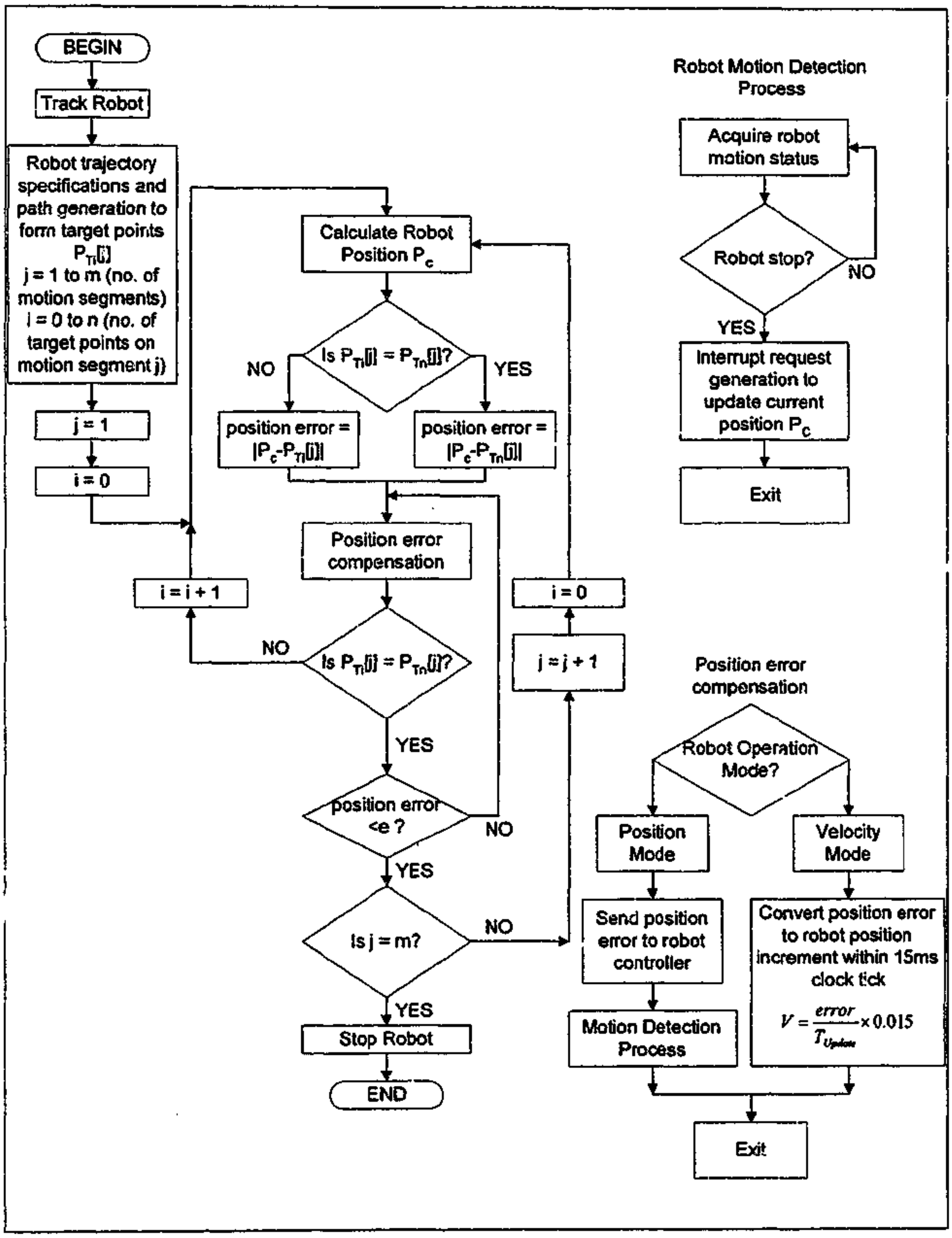

BEGIN
Track Robot
Robot trajectory specifications and path generation to form target points $P_{Ti}[j]$
j = 1 to m (no. of motion segments)
i = 0 to n (no. of target points on motion segment j)
j = 1
i = 0
Calculate Robot Position P_c
Is $P_{Ti}[j] = P_{Tn}[j]$?
NO
YES
position error = $|P_c - P_{Ti}[j]|$
position error = $|P_c - P_{Tn}[j]|$
Position error compensation
i = i + 1
i = 0
NO
Is $P_{Ti}[j] = P_{Tn}[j]$?
j = j + 1
YES
position error <e ?
NO
YES
Is j = m?
NO
YES
Stop Robot
END
Robot Motion Detection Process
Acquire robot motion status
Robot stop?
NO
YES
Interrupt request generation to update current position P_c
Exit
Position error compensation
Robot Operation Mode?
Position Mode
Velocity Mode
Send position error to robot controller
Convert position error to robot position increment within 15ms clock tick
$V = \frac{error}{T_{Update}} \times 0.015$
Motion Detection Process
Exit

Figure 6.2: Flowchart of closed-loop control algorithm using LISM

]

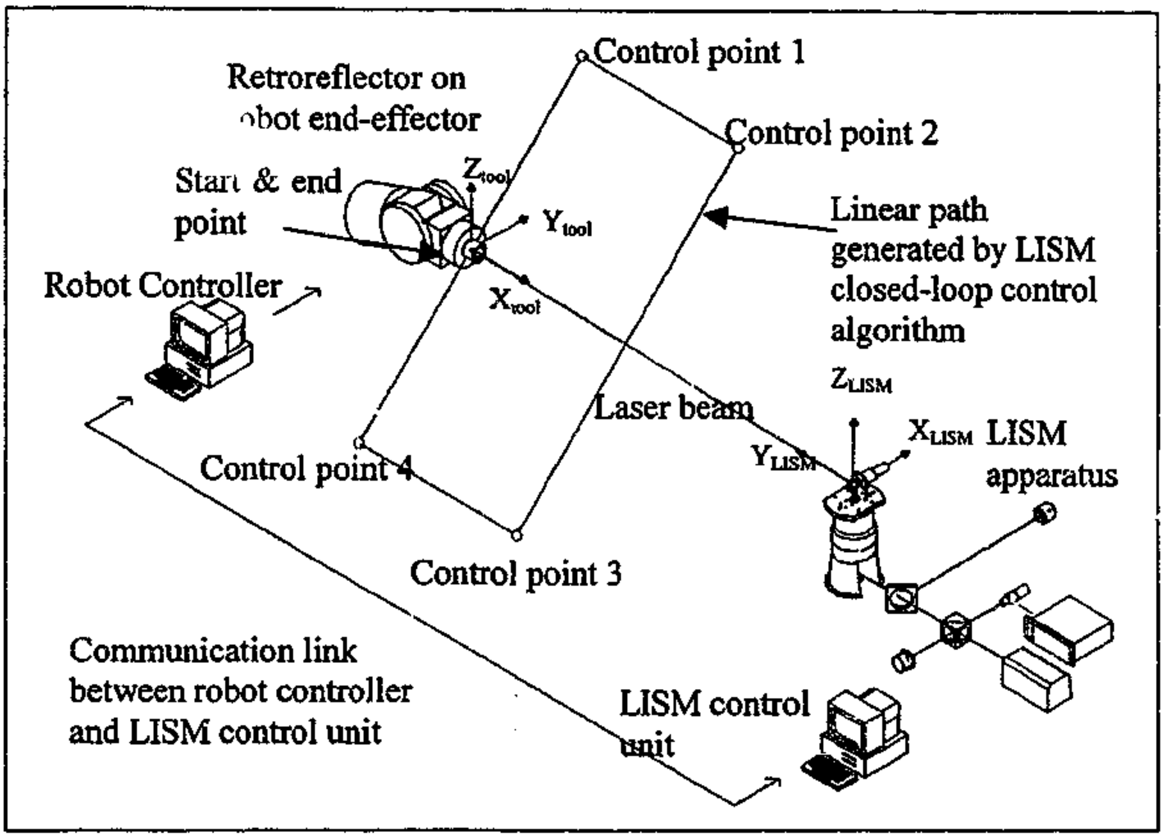


Figure 6.3: Schematic description of linear path generated by LISM closed-loop control algorithm

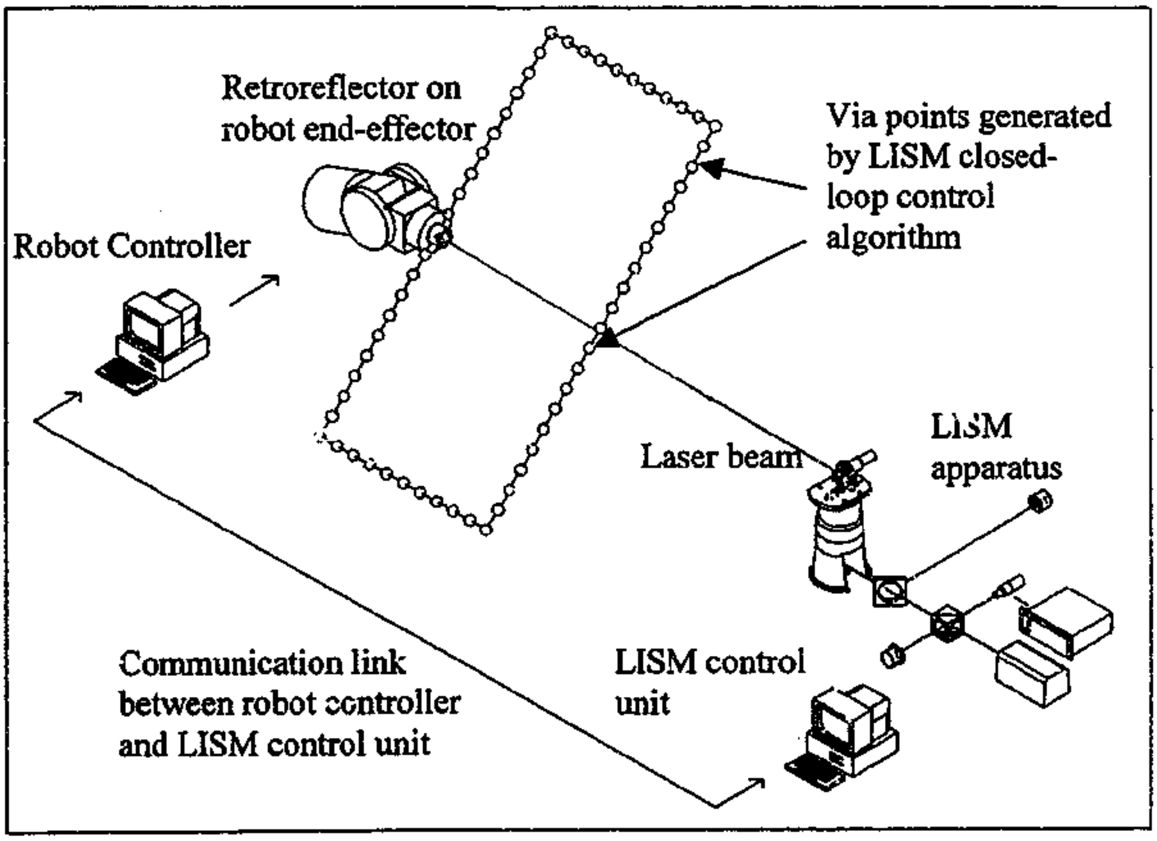


Figure 6.4: Schematic description of via points generated by LISM closed-loop control algorithm

6.3.1 Position Control Mode

In position control mode, the via points are generated based on frequent a robot position verification is required. Due to the incapability of changing the position of most industrial robots on the fly, the robot has to stop at each target point in position control mode. The motion status of the robot manipulator is being monitored continuously by the motion status detection process as shown in Fig. 6.2. When the robot has stopped by reaching a via point, the current position of the robot measured by the LISM apparatus is compared to the next via point and the resultant position increment is transmitted to the robot. By using more via points along a motion segment, the path accuracy of the end-effector can be further improved.

In the case where a control point has been reached, the position error is compared to the predefined error limit. If the error is more than the predefined error limit, this error is again compensated by commanding the robot to move by the calculated increment. If the position error is less than the predefined error limit, the robot is commanded to move to the next via point on the next motion segment. In the case of reaching an end point, the error is compensated continuously until the end-effector of the robot reaches a position where the error is less than the predefined error limit. The predefined error limit is specified based on the precision required at each control point or end point. This reduces steady state error due to the inaccuracy of the robot controller in positioning the end-effector of the robot. Further, the position error of the robot is minimised before the robot is commanded to the next control point. By using this methodology, the end-effector of the robot is commanded to all the target points increment by increment along the predefined path.

6.3.2 Velocity Control Mode

In velocity control mode, the via points are generated based on the average velocity commanded and update interval. Instead of stopping at each via point as described in position control mode, these via points are used for checking the position of the robot at every update interval. At every update interval, the current position of the robot measured by the LISM apparatus is compared to the next via point to calculate the position increment. The velocity of the robot is updated based on the resultant position increment. With lower average velocity, more via points are generated and the path accuracy of the end-effector can be further improved.

Similarly, when a control point has been reached, the position error is compared to the predefined error limit. If the error is more than the predefined error limit, this error is again compensated by commanding the robot to move by the calculated velocity. The robot is commanded to move to the next via point on the next motion segment only when the error is less than the predefined error limit. In the case of reaching the end point, the error is compensated continuously until the end-effector of the robot reaches a position where the error is less than the predefined error limit. By controlling the velocity of the robot, the position error can be compensated on the fly. The end-effector of the robot is therefore commanded through all of the target points continuously along the predefined path.

6.4 Experimental Set-up

Experiments have been performed to establish the behaviour of the control algorithm for closed-loop control of robot manipulators. The experimental set-up, shown in Fig. 6.5, consists of the mentioned experimental LISM apparatus and a Motoman robot manipulator. A special feature of the Motoman robot controller is that the velocity of the robot can be controlled using the TurboLink interface [72]. This is carried out using the real-time service function, which synchronises with the real-time clock of the controller to read incremental position data from an application program. The interval of the real-time clock is 15 ms by default. The robot moves by the commanded increment at every clock tick. Zero increment is used if it is not updated by the user, thus stopping the robot. This robot has a resolution of 20μm and a repeatability of 0.4mm [73].

During the experiments, a robot tool co-ordinate frame was first established so that subsequent movements of the end-effector were recorded with respect to the origin of this co-ordinate frame (refer to Appendix D for the procedures of transforming LISM measurements into positions with respect to the robot tool co-ordinate frame). The co-ordinate frames are shown in Fig. 6.3.

In the first experiment, the position control mode was used. The end-effector was commanded to an end point by moving along a single axis. The Cartesian co-ordinates of the end-effector at every target points were recorded using the LISM apparatus. Two different sets of results were recorded for motion along each axis, one with the method of CLC and the other without. The experiment was repeated using a higher number of target points along the path.

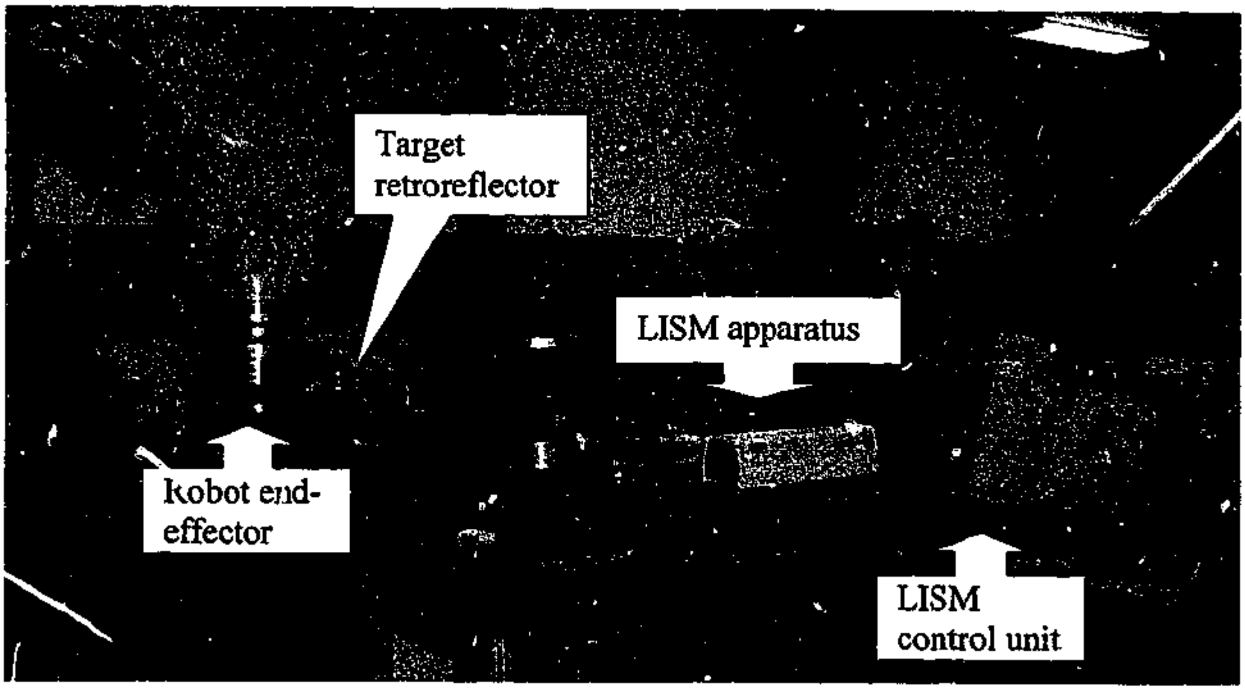


Figure 6.5: Experiment set-up with LISM apparatus and robot manipulator

The second experiment was performed to investigate the velocity control mode in this methodology. Again, the end-effector was commanded to an end point by moving along a single axis at a desired velocity. The experiment was repeated using a lower velocity. The Cartesian co-ordinates of the end-effector at each update interval were recorded using the LISM apparatus. The performance of the position and velocity control modes were compared in order to identify the effectiveness and behaviour of both control modes.

6.5 Experimental Results and Discussions

Figs. 6.6 to 6.8 show the error between the measured and target robot positions without the LISM closed-loop control, moving along the robot x-, y- and z-axes, respectively. From these figures, it can be observed that there are initial position errors. This suggests that the starting position commanded by the robot controller differs from the desired starting point. The possible source of this error is the inaccurate kinematic parameters used in the kinematic model of the robot manipulator. Moreover, there were non-linear effects such as static and dynamic friction, hysteresis, and non-rigid behaviour of the robot manipulator [68, 74]. When the robot was in motion, there were changes in the position error, which represents a deviation of the end-effector away from the desired path. More significant changes can be observed in the x and z co-ordinates. A possible reason for this phenomenon could be the wobbling and bending of the robot arm due to gravitational force. Table 6.1 indicates that the maximum RMS position error for the end-effector along the commanded path is 0.7mm along the x- and y-axes, and 1.6mm along the z-axis. The maximum steady state error is ±0.8mm along the x- and y-axes, and ±1.8mm along the z-axis. Due to the fact that the robot controller does not observe the steady state error, this error will not be eliminated.

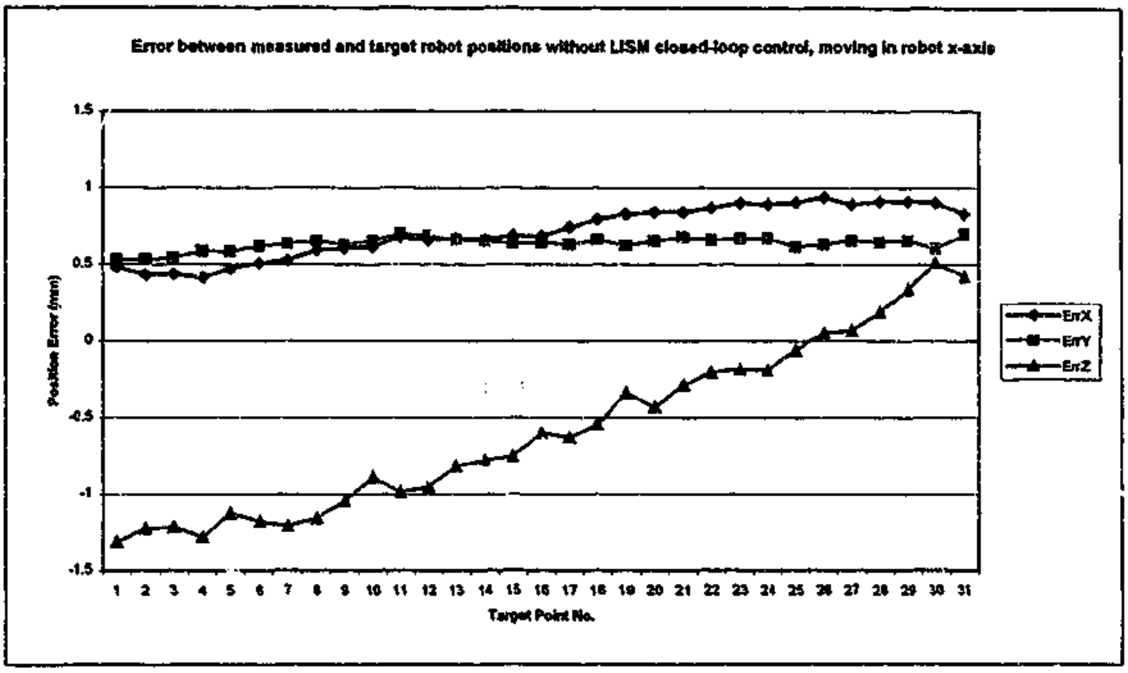


Figure 6.6: Error between measured and target robot positions without LISM closed-loop control while moving in robot x-axis

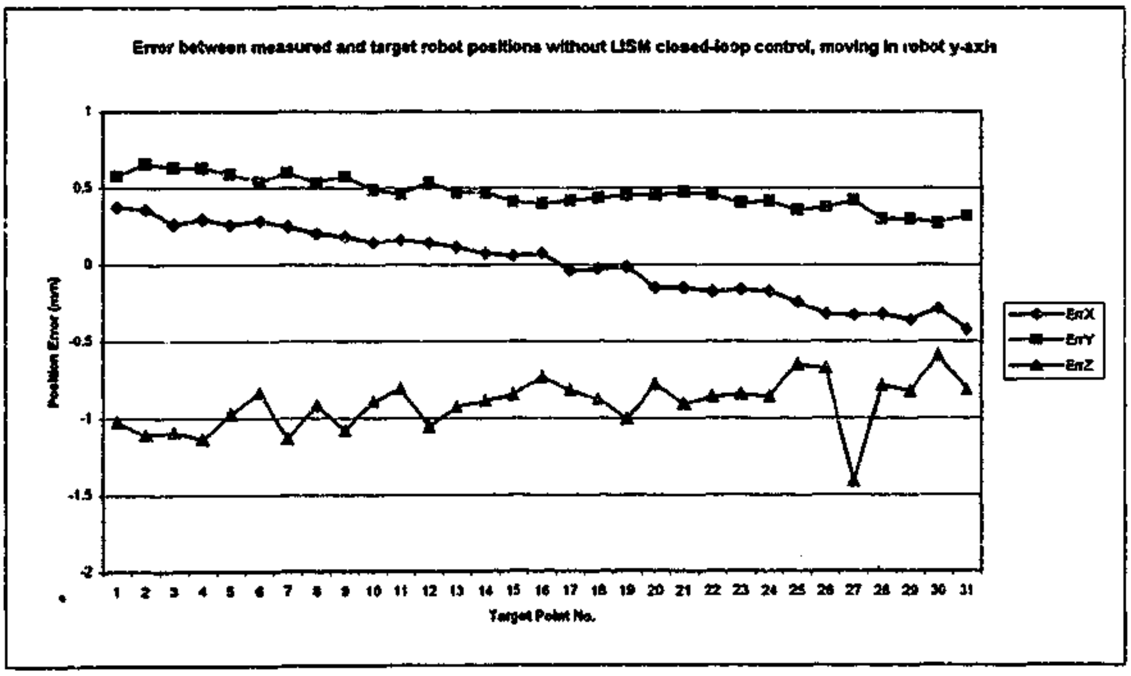


Figure 6.7: Error between measured and target robot positions without LISM closed-loop control while moving in robot y-axis

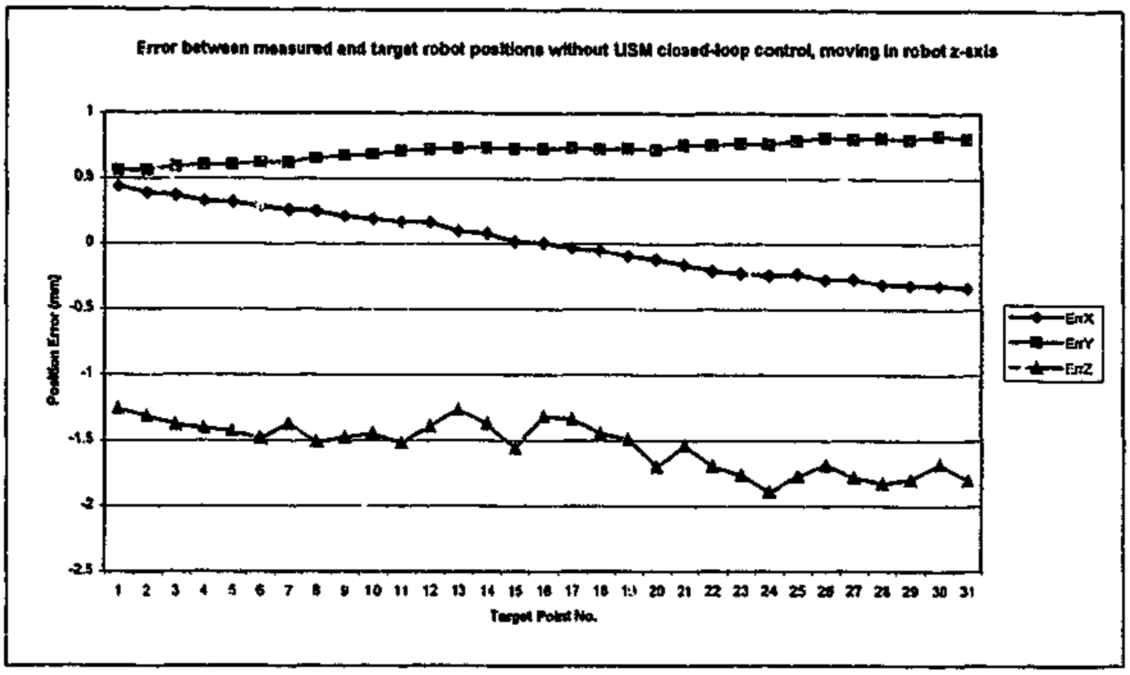


Figure 6.8: Error between measured and target robot positions without LISM closed-loop control while moving in robot z-axis

Table 6 1: Comparison of RMS and steady state position error with and without the implementation of position mode closed-loop control

	Distance between two via points: 10mm					
	With position mode CLC RMS Error			Without CLC RMS Error		
Move along	X	Y	Z	X	Y	Z
X	0.045	0.044	0.109	0.733	0.634	0.795
Y	0.054	0.040	0.179	0.235	0.479	0.924
Z	0.035	0.020	0.121	0.245	0.717	1.556
	With position mode CLC Steady state Error			Without CLC Steady state Error		
Move along	X	Y	Z	X	Y	Z
X	0.039	-0.042	-0.015	0.828	0.695	0.420
Y	-0.030	0.013	0.030	-0.422	0.320	-0.815
Z	0.003	0.017	0.041	-0.338	0.804	-1.805
	Distance between two via points: 5mm					
	With position mode CLC RMS Error			Without CLC RMS Error		
Move along	X	Y	Z	X	Y	Z
X	0.053	0.045	0.113	0.733	0.634	0.795
Y	0.042	0.054	0.168	0.235	0.479	0.924
Z	0.036	0.020	0.113	0.245	0.717	1.556
	With position mode CLC Steady state Error			Without CLC Steady state Error		
Move along	X	Y	Z	X	Y	Z
X	-0.052	0.045	-0.047	0.828	0.695	0.420
Y	-0.012	-0.006	-0.011	-0.422	0.320	-0.815
Z	-0.081	-0.023	-0.049	-0.338	0.804	-1.805

With the implementation of position mode closed-loop control (CLC), the initial position error has been compensated by first commanding the robot to the desired starting position as shown in Figs 6.9 to 6.11. There were still fluctuations in the position errors mainly due to the inaccurate kinematic parameters used in the kinematic model of the robot manipulators. The position compensation increment provided by the CLC algorithm was not being acted upon by the robot controller. In addition, there were coupling effect among the moving joints, which causes the coupling among the motion directions in Cartesian space. Furthermore, the error may be the results of non-linear effects such as friction, stiction, and elastic deformations of the robot arm. The magnitude of fluctuation is higher along z-axis due to the gravitational force. It must be noted that these errors were being detected by the control algorithm through the dynamic measurements of the robot end-effector. Therefore, compensation can be performed, resulting in a lower RMS position error of 0.05mm along the x-axis, 0.04mm along the y-axis and 0.18mm along the z-axis (Table 6.1). It must be noted that the RMS position errors in x, y, z directions have been reduced by 93 %, 94 %, 89 %, respectively. Likewise, the maximum steady state error has been reduced to ±0.04mm along the x-, y- and z-axes.

When the distance between the generated via points was reduced, the path and position accuracy of the end-effector did not improve as expected (see Figs. 6.12 to 6.14). This is again due to the bending of the robot arm and inaccurate kinematic model of the robot manipulator. Further, with smaller position increments, non-linear effect such as static friction is more dominant on the motion of the robot. Moreover, the RMS position error achieved in the previous section was close to the repeatability, and the steady state error was comparable to the resolution of the robot manipulator. Consequently, further improvement in path and position accuracy could not be attained. Table 6.1 indicates maximum RMS position error of 0.05mm along the x- and y-axes, and 0.17mm along the z-axis. The maximum steady state error is ±0.08mm along the x-axis, ±0.05mm along the y- and z-axes. It must be noted that the steady state position errors in x, y, z directions have been reduced by 90 %, 94 %, 97 %, respectively.

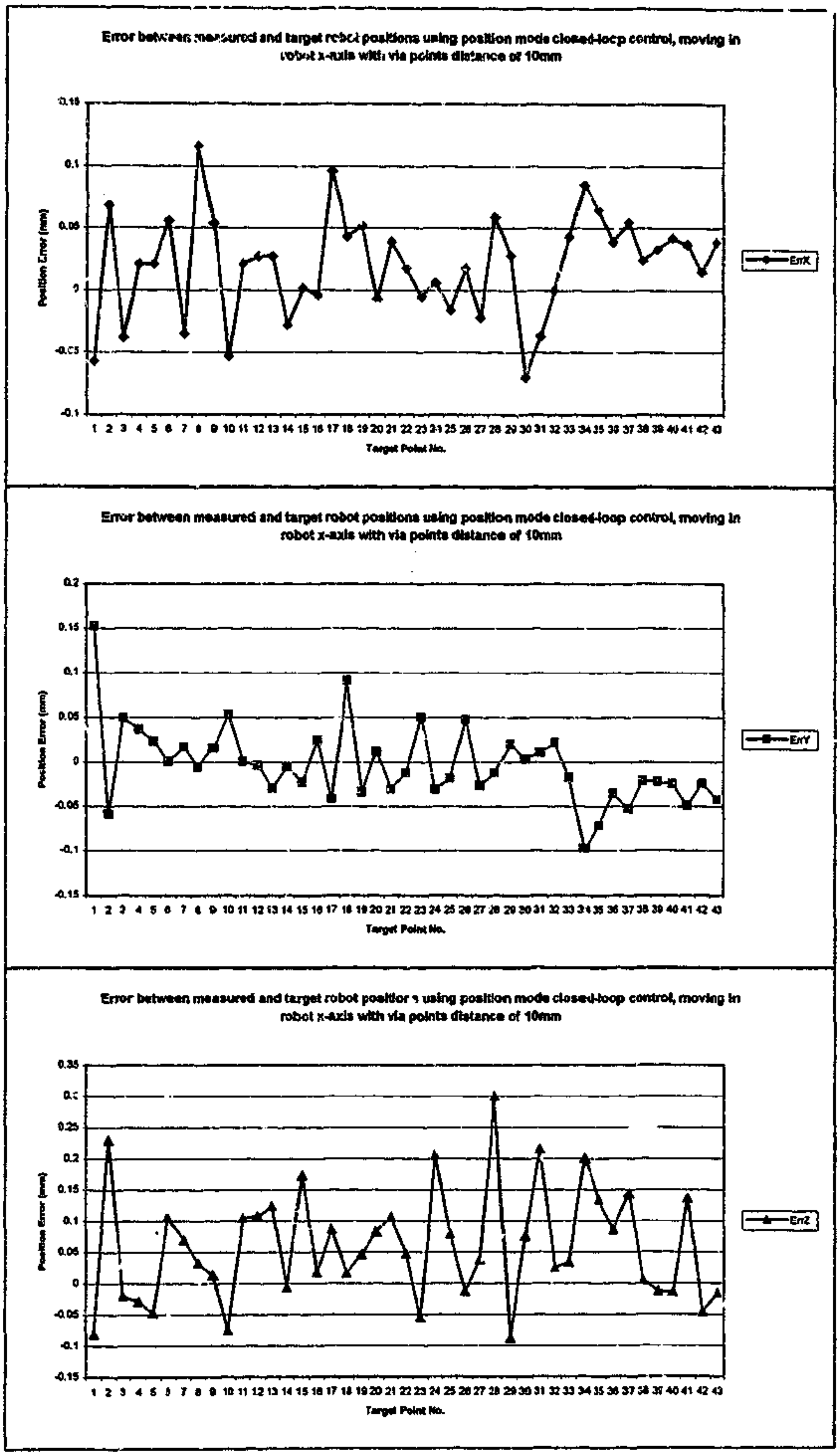
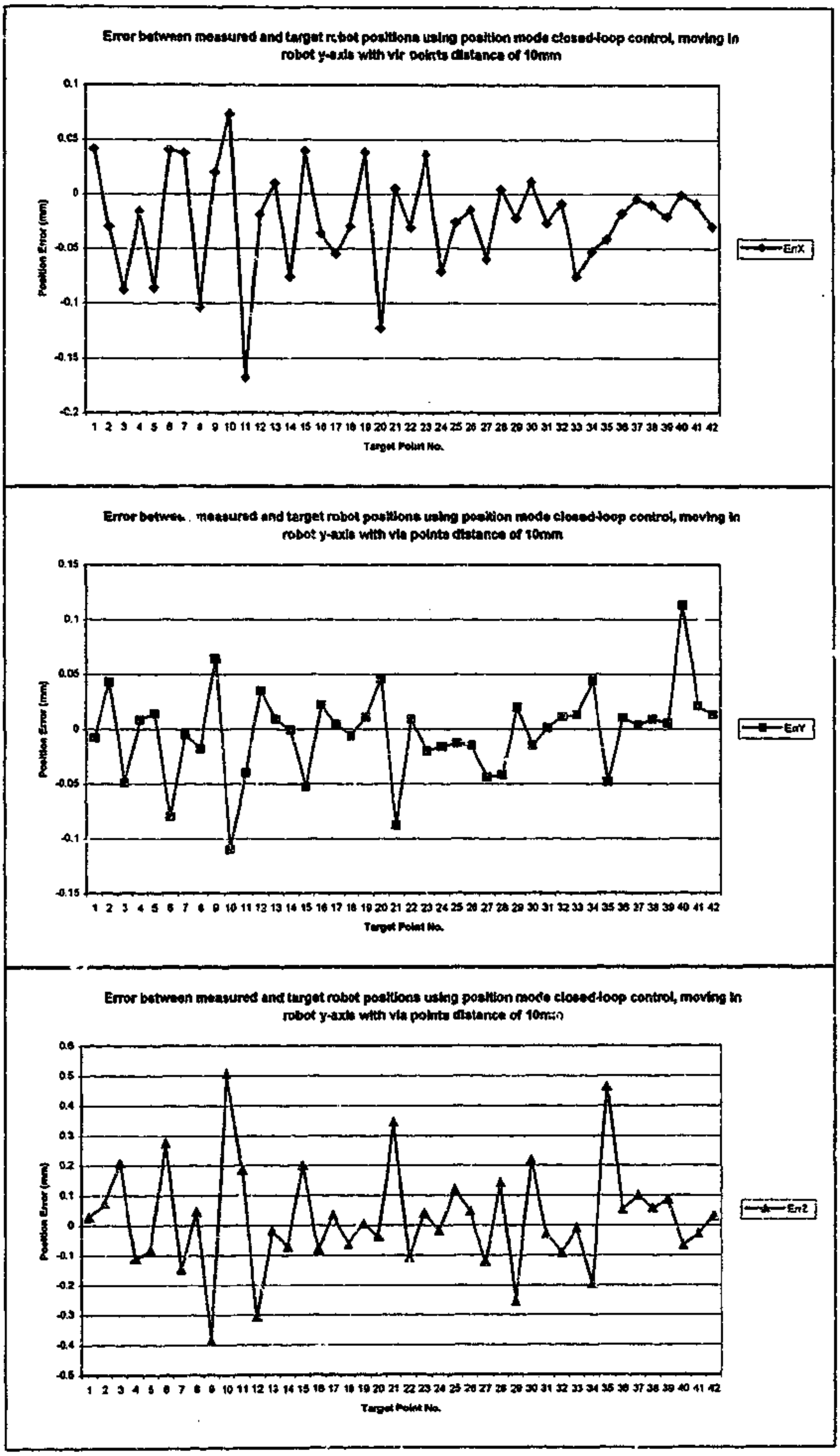


Figure 6.9: Error between measured and target robot positions using position mode closed-loop control while moving in robot x-axis with via points distance of 10mm


Error between measured and target robot positions using position mode closed-loop control, moving in robot y-axis with via points distance of 10mm

Error between measured and target robot positions using position mode closed-loop control, moving in robot y-axis with via points distance of 10mm

Error between measured and target robot positions using position mode closed-loop control, moving in robot y-axis with via points distance of 10mm

Figure 6.10: Error between measured and target robot positions using position mode closed-loop control while moving in robot y-axis with via points distance of 10mm

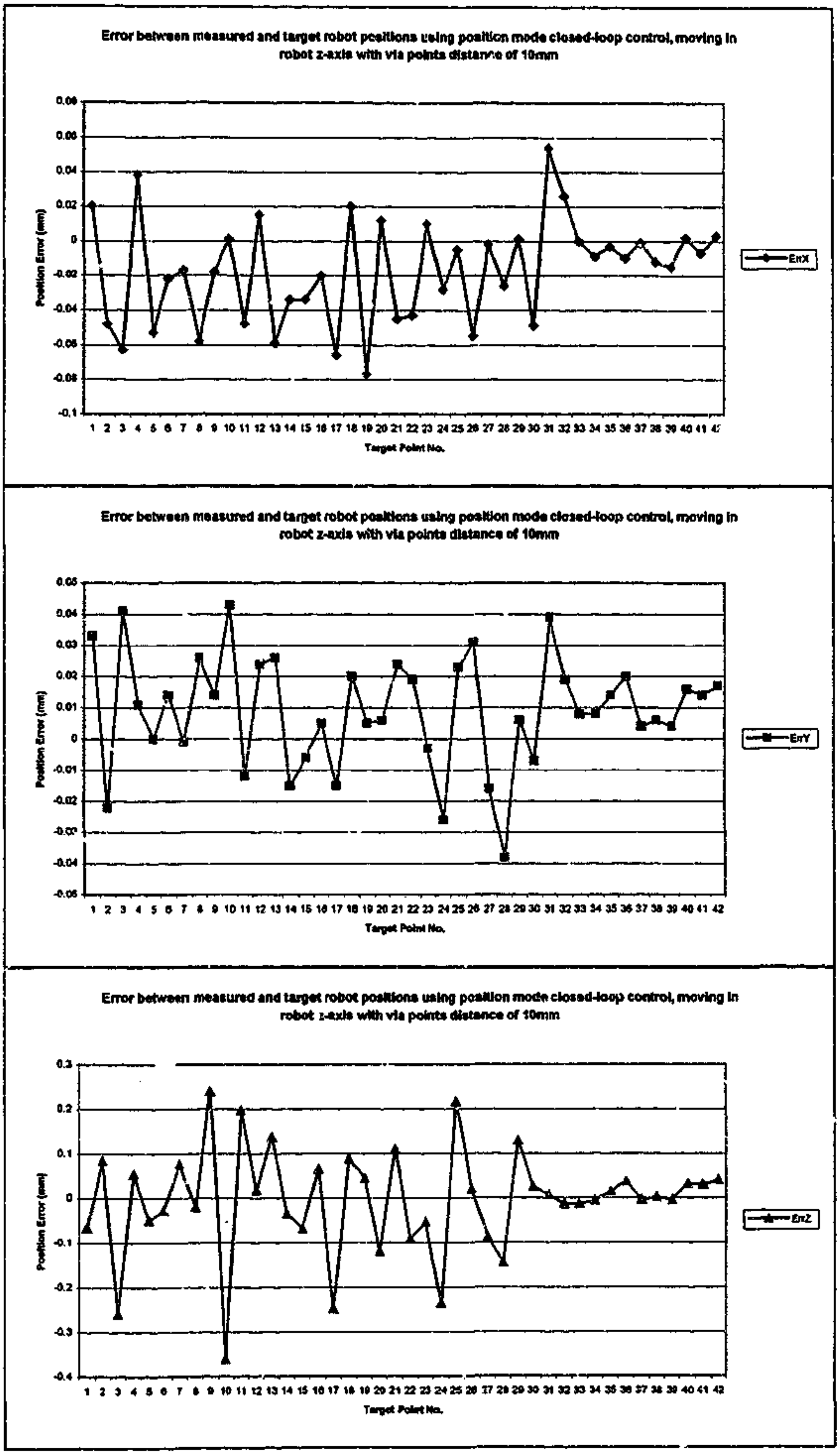


Figure 6.11: Error between measured and target robot positions using position mode closed-loop control while moving in robot z-axis with via points distance of 10mm

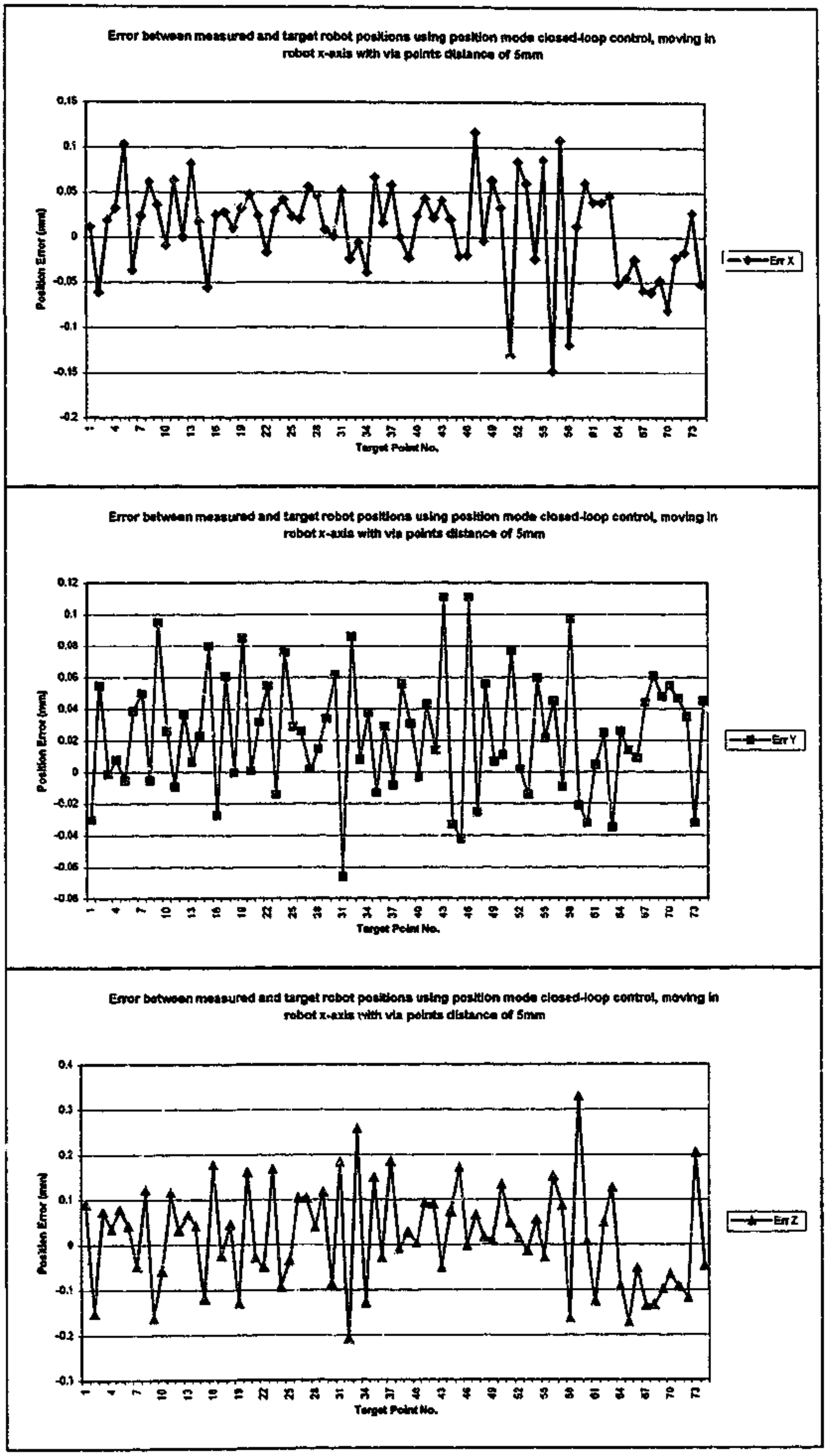


Figure 6.12: Error between measured and target robot positions using position mode closed-loop control while moving in robot x-axis with via points distance of 5mm

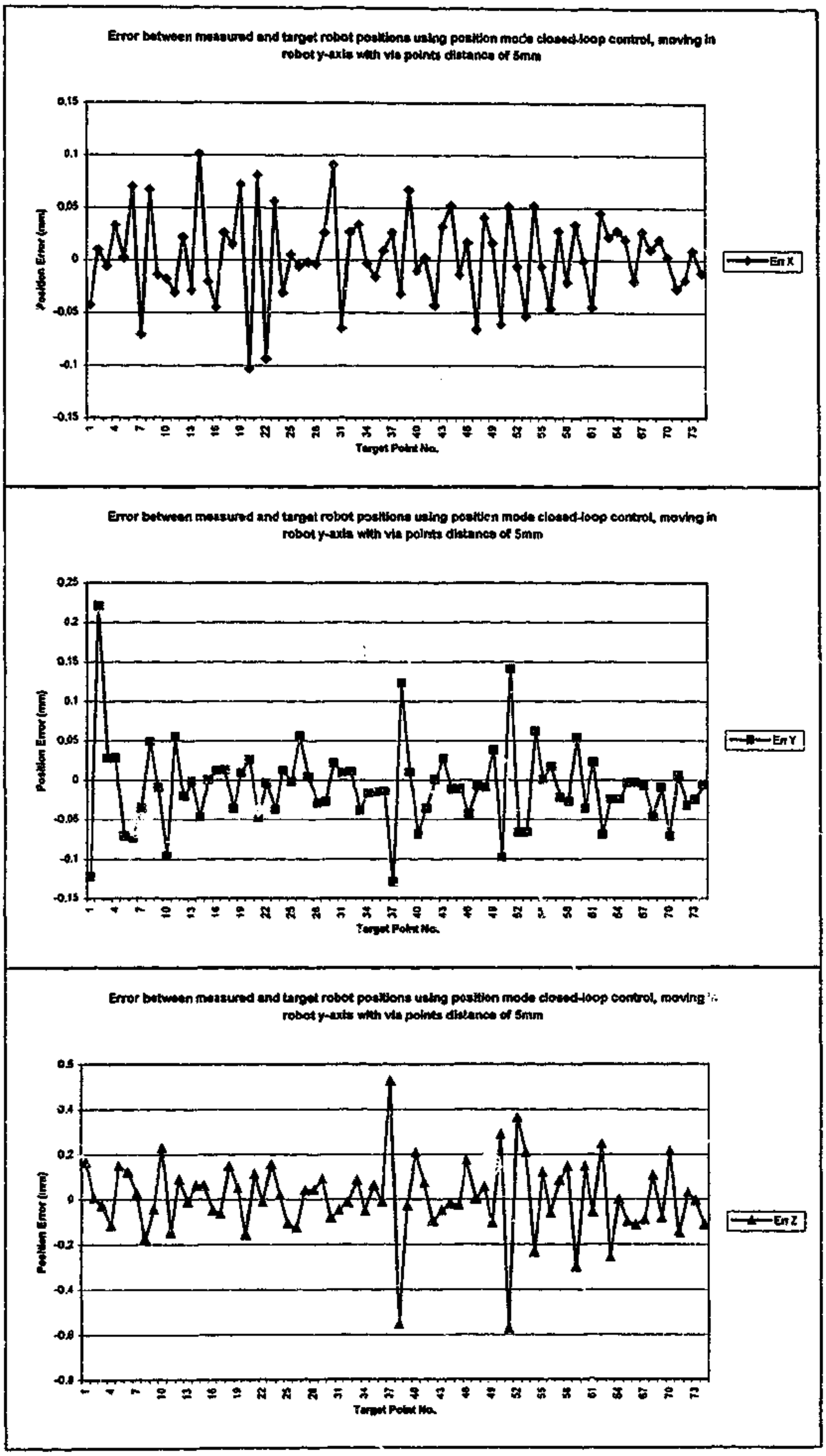


Figure 6.13: Error between measured and target robot positions using position mode closed-loop control while moving in robot y-axis with via points distance of 5mm

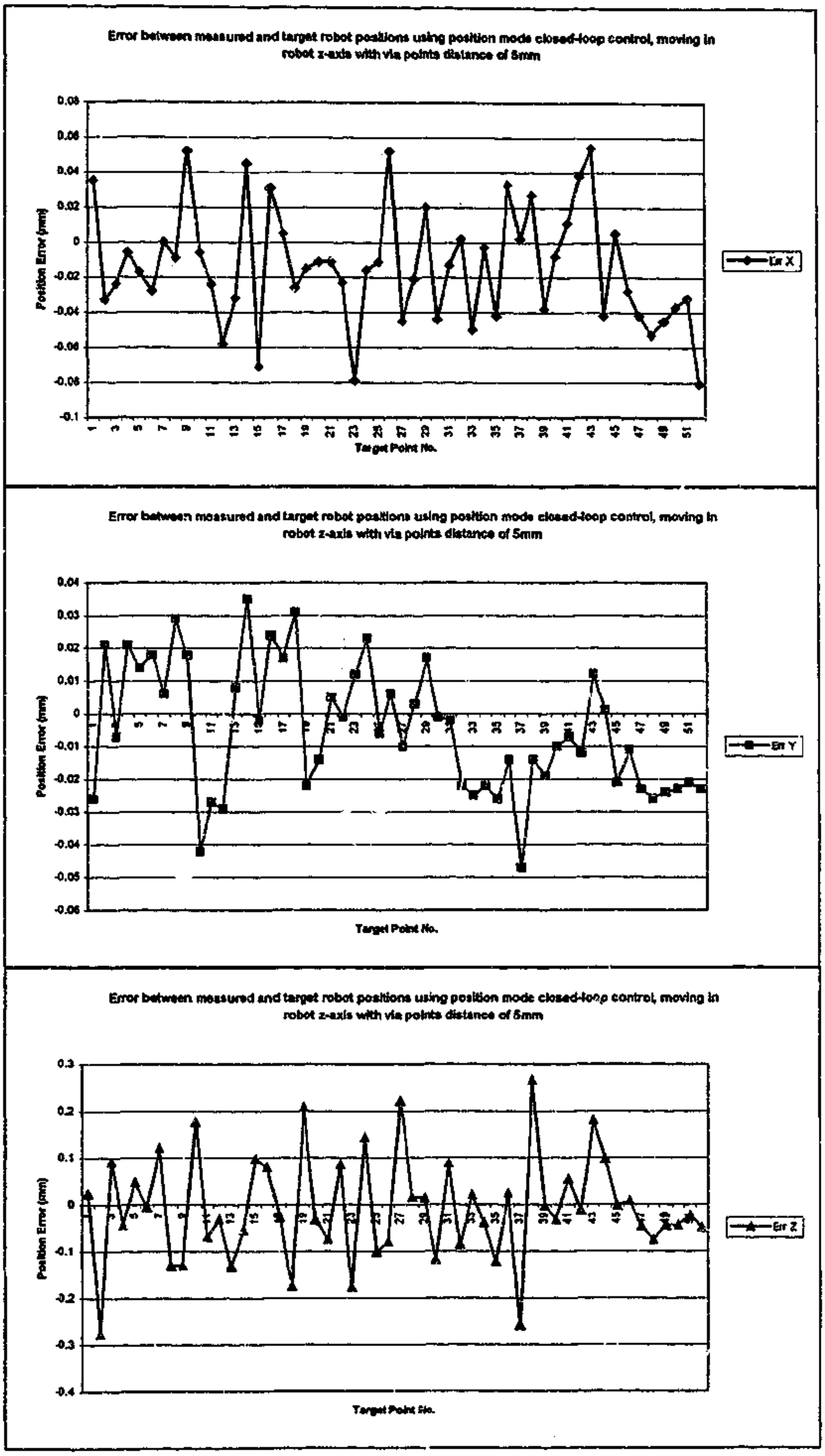


Figure 6.14: Error between measured and target robot positions using position mode closed-loop control while moving in robot z-axis with via points distance of 5mm

Figs. 6.15 to 6.20 show the results for velocity mode CLC. The RMS and steady state position error is tabulated in Table 6.2. From this table, it can be observed that the RMS position error was very high for the axis that the robot was commanded to move along (i.e. higher RMS position error along x-axis when the robot was commanded to move along the robot x-axis). This is because when a command was being sent to the robot using the TurboLink interface, there was a delay of 300ms before the command was being acted upon by the robot. In order to synchronise with the real time clock of the robot controller, an update interval of 15ms was used for the velocity mode closed-loop control algorithm. At every 15ms, the subsequent via point was used to calculate the position error. The robot was still approaching the first via point while the position error was computed using the 20th via point due to this delay. This results in higher RMS position error. A time-delay control algorithm is required to reduce the error [75]. However, for robot motion along a single axis, the high RMS position error (greyed diagonal elements in Table 6.2) for the co-ordinate that the robot was commanded to move along does not cause any path deviation. The end-effector was observed to be following the desired path closely at the commanded velocity, as shown in Figs. 6.15, 6.17 and 6.19. The deviation of the robot from the desired path was dominated by the position error in the other two directions, mainly due to the 300 ms delay and the coupling among the motion directions. From Table 6.2, it can be observed that the maximum RMS position error is higher when compared to position mode guidance. Similarly, these errors were being detected and compensated by the control algorithm, resulting in improved RMS position error when compared to commanding the robot without closed-loop control algorithm. With the omission of the diagonal elements, Table 6.2 indicates maximum RMS position error of 0.1mm along the x-axis, and 0.3mm along the y- and z-axes. By comparing with the maximum RMS position errors without CLC (Fig. 6.7 to 6.8), the maximum RMS position errors have been reduced by 86 %, 57 %, 81 % in the x, y, z directions, respectively. The maximum steady state error is ±0.07mm along the x- and y-axes, and ±0.05mm along the z-axis.

Figs. 6.21 to 6.26 show the results for velocity mode CLC using a lower commanded velocity. From Table 6.2, it can be observed the RMS position error was also very high for the co-ordinate that the robot was commanded to move along due to the same reason as described above. However, it can be observed that the RMS position error has been reduced in the other non-moving co-ordinates when compared to higher commanded velocity. This was due to the higher number of via points generated, and thus position compensation was carried out more frequently. The path accuracy of the end-effector was further improved. Table 6.2 indicates a maximum RMS position error of 0.08mm along the x-axis, and 0.15mm along the y-axis and 0.27mm along the z-axis. The maximum steady state error is ±0.07mm along the x-axis, ±0.05mm along the y-axis, and ±0.02mm along the z-axis.

Table 6.2: Comparison of RMS and steady state position error with and without the implementation of velocity mode closed-loop control

	Average velocity 5mm/s					
	With velocity mode CLC RMS Error			Without CLC RMS Error		
Move along	X	Y	Z	X	Y	Z
X	[illegible]	0.294	0.249	0.733	0.634	0.795
Y	0.096	[illegible]	0.327	0.235	0.479	0.924
Z	0.045	0.110	[illegible]	0.245	0.717	1.556
	With velocity mode CLC Steady state Error			Without CLC Steady state Error		
Move along	X	Y	Z	X	Y	Z
X	-0.033	0.071	0.016	0.828	0.695	0.420
Y	0.045	0.011	-0.043	-0.422	0.320	-0.815
Z	0.066	-0.015	0.026	-0.338	0.804	-1.805
	Average velocity 2mm/s					
	With velocity mode CLC RMS Error			Without CLC RMS Error		
Move along	X	Y	Z	X	Y	Z
X	[illegible]	0.152	0.253	0.733	0.634	0.795
Y	0.078	[illegible]	0.265	0.235	0.479	0.924
Z	0.075	0.080	[illegible]	0.245	0.717	1.556
	With velocity mode CLC Steady state Error			Without CLC Steady state Error		
Move along	X	Y	Z	X	Y	Z
X	-0.060	0.031	0.014	0.828	0.695	0.420
Y	0.073	0.041	0.015	-0.422	0.320	-0.815
Z	0.068	-0.026	-0.014	-0.338	0.804	-1.805

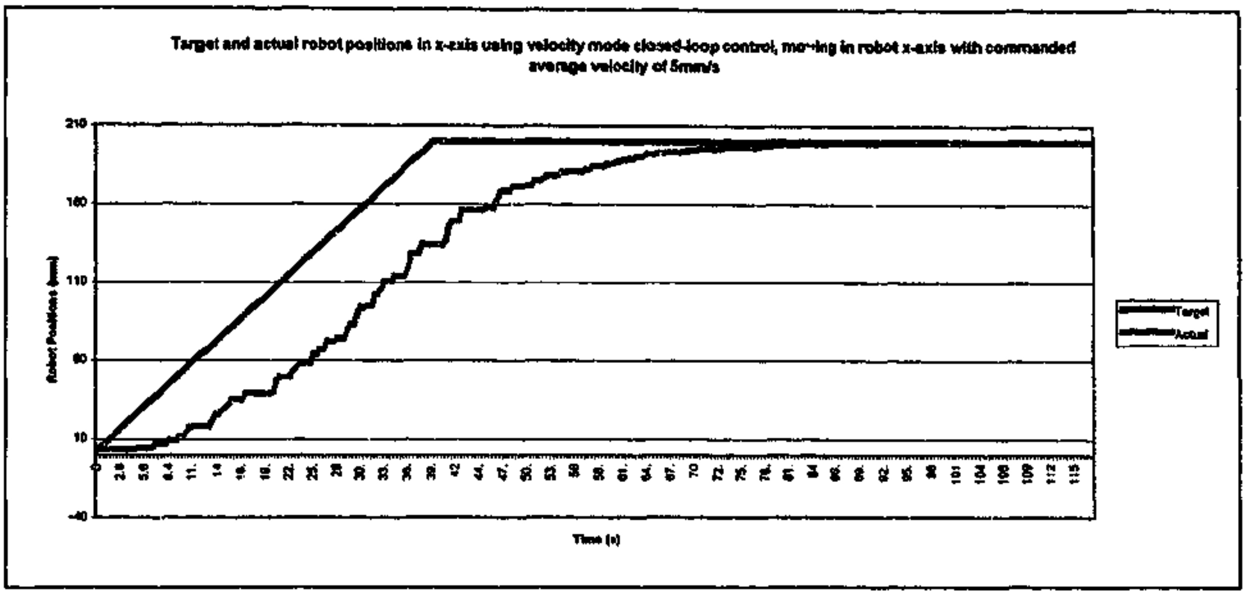


Figure 6.15: Target and actual robot positions in x-axis using velocity mode closed-loop control while moving in robot x-axis with commanded velocity of 5mm/s

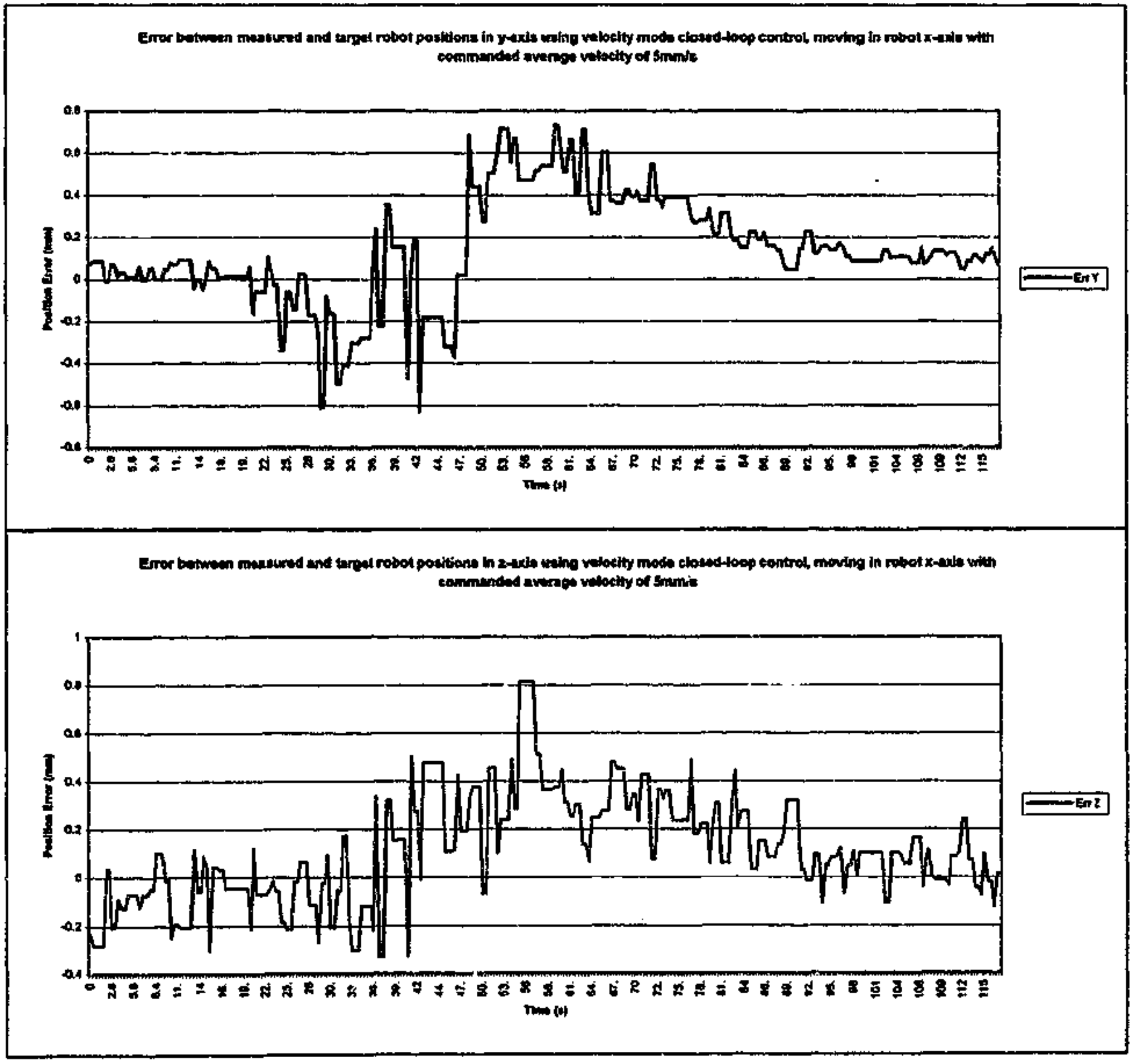


Figure 6.16: Error between measured and target robot positions using velocity mode closed-loop control while moving in robot x-axis with commanded velocity of 5mm/s

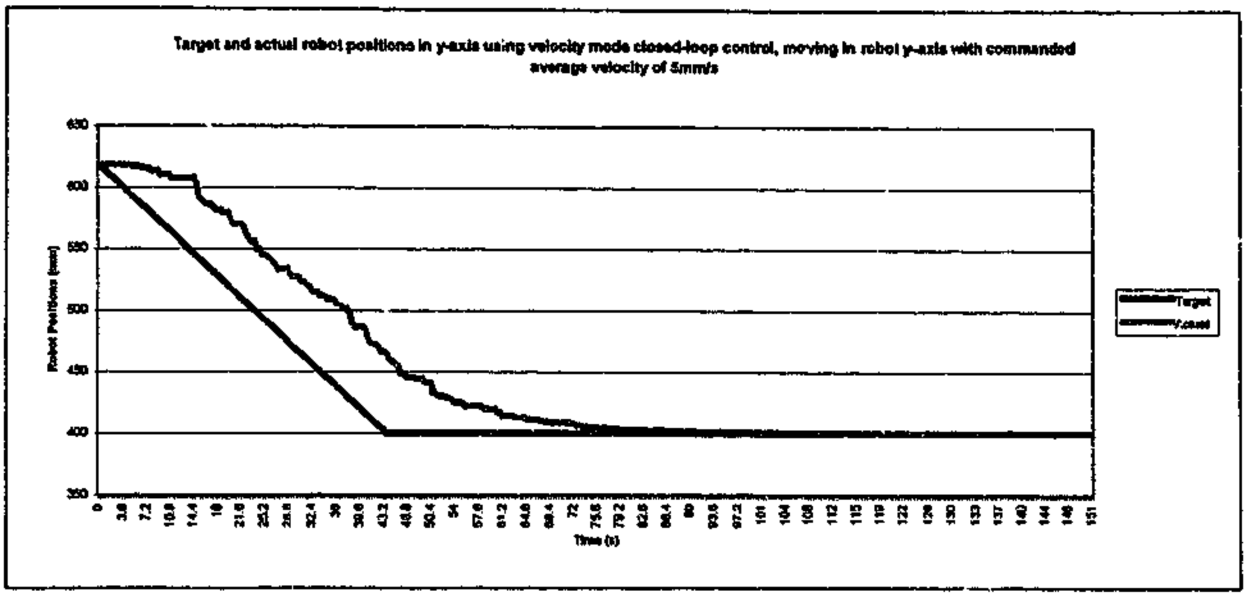


Figure 6.17: Target and actual robot positions in y-axis using velocity mode closed-loop control while moving in robot y-axis with commanded velocity of 5mm/s

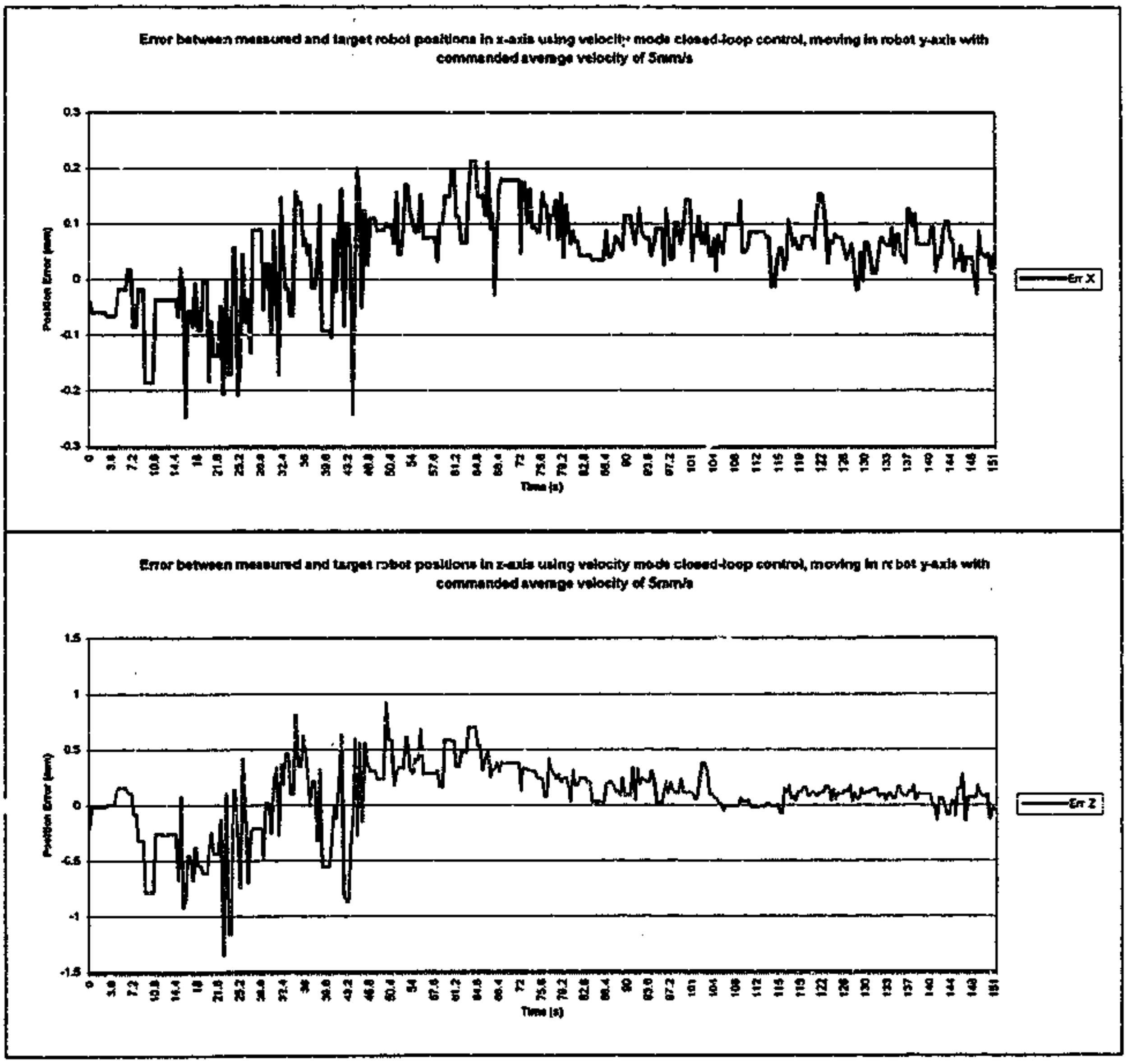


Figure 6.18: Error between measured and target robot positions using velocity mode closed-loop control while moving in robot y-axis with commanded velocity of 5mm/s

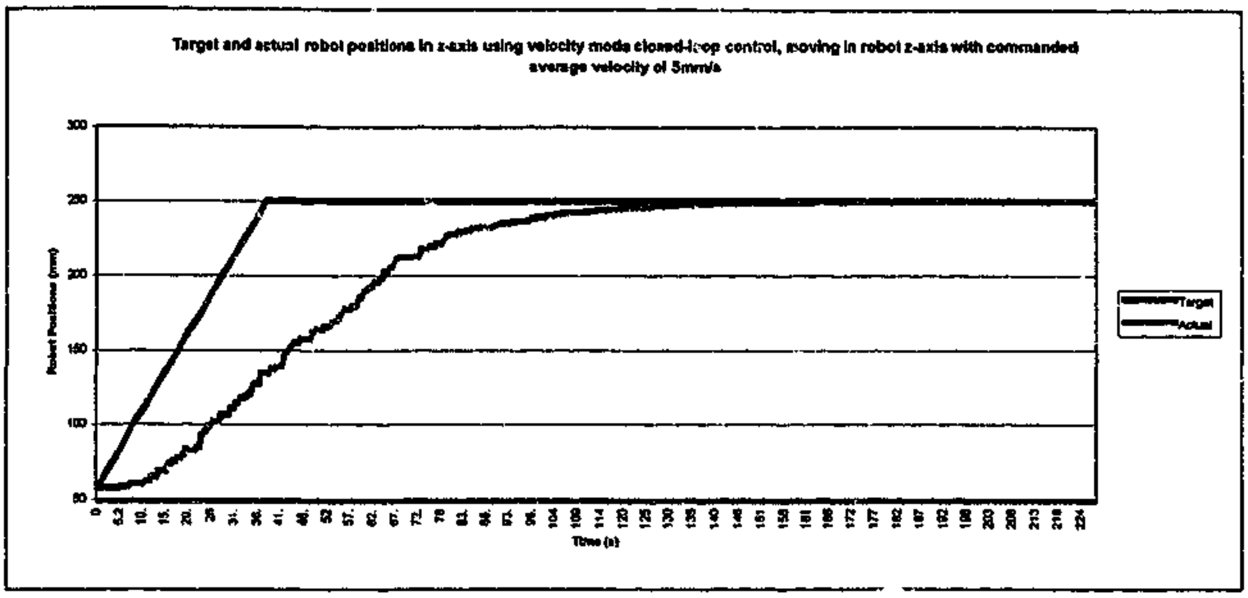


Figure 6.19: Target and actual robot positions in z-axis using velocity mode closed-loop control while moving in robot z-axis with commanded velocity of 5mm/s

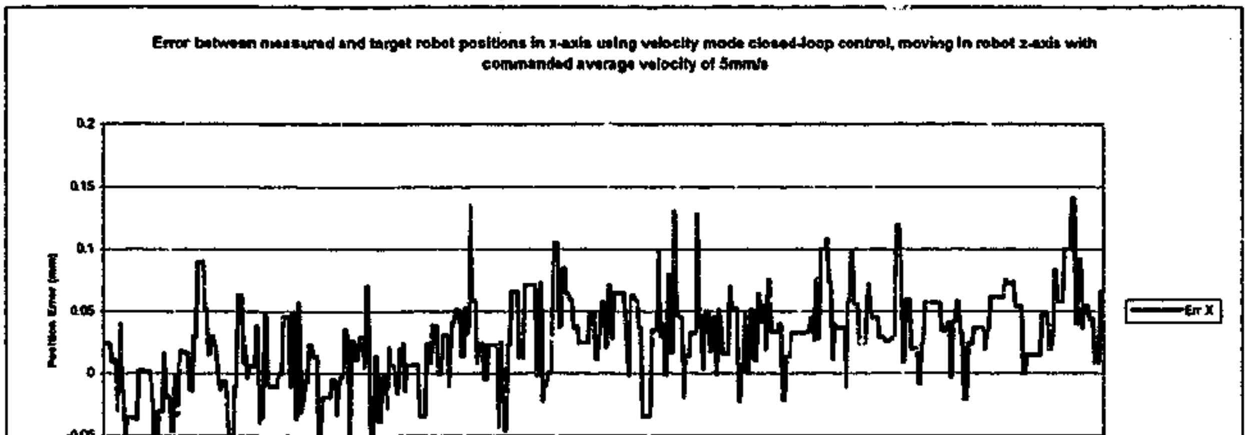


Figure 6.20: Error between measured and target robot positions using velocity mode closed-loop control while moving in robot z-axis with commanded velocity of 5mm/s

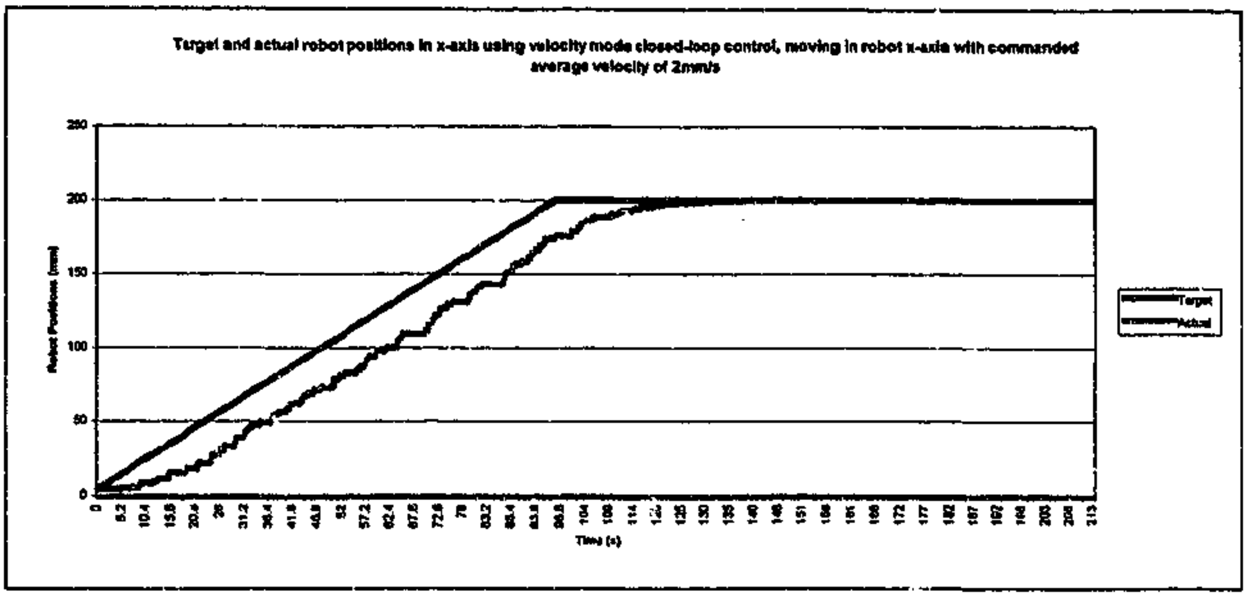


Figure 6.21: Target and actual robot positions in x-axis using velocity mode closed-loop control while moving in robot x-axis with commanded velocity of 2mm/s

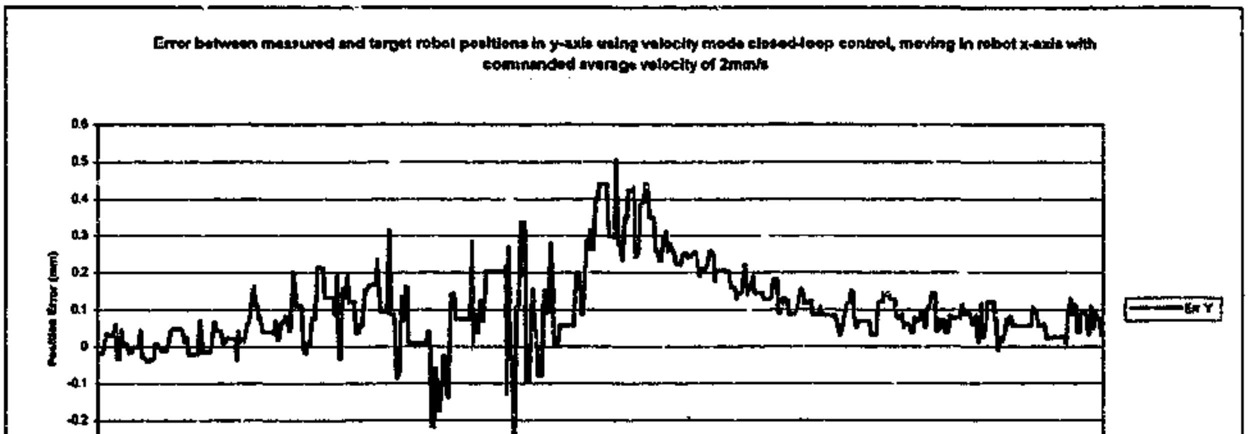


Figure 6.22: Error between measured and target robot positions using velocity mode closed-loop control while moving in robot x-axis with commanded velocity of 2mm/s

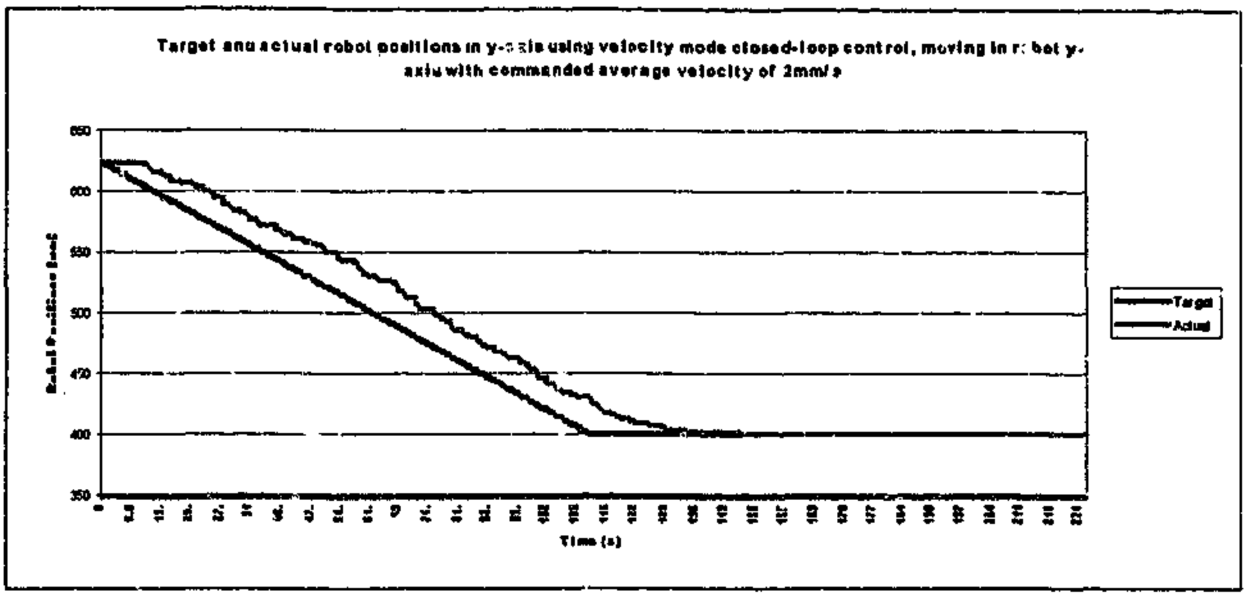


Figure 6.23: Target and actual robot positions in y-axis using velocity mode closed-loop control while moving in robot y-axis with commanded velocity of 2mm/s

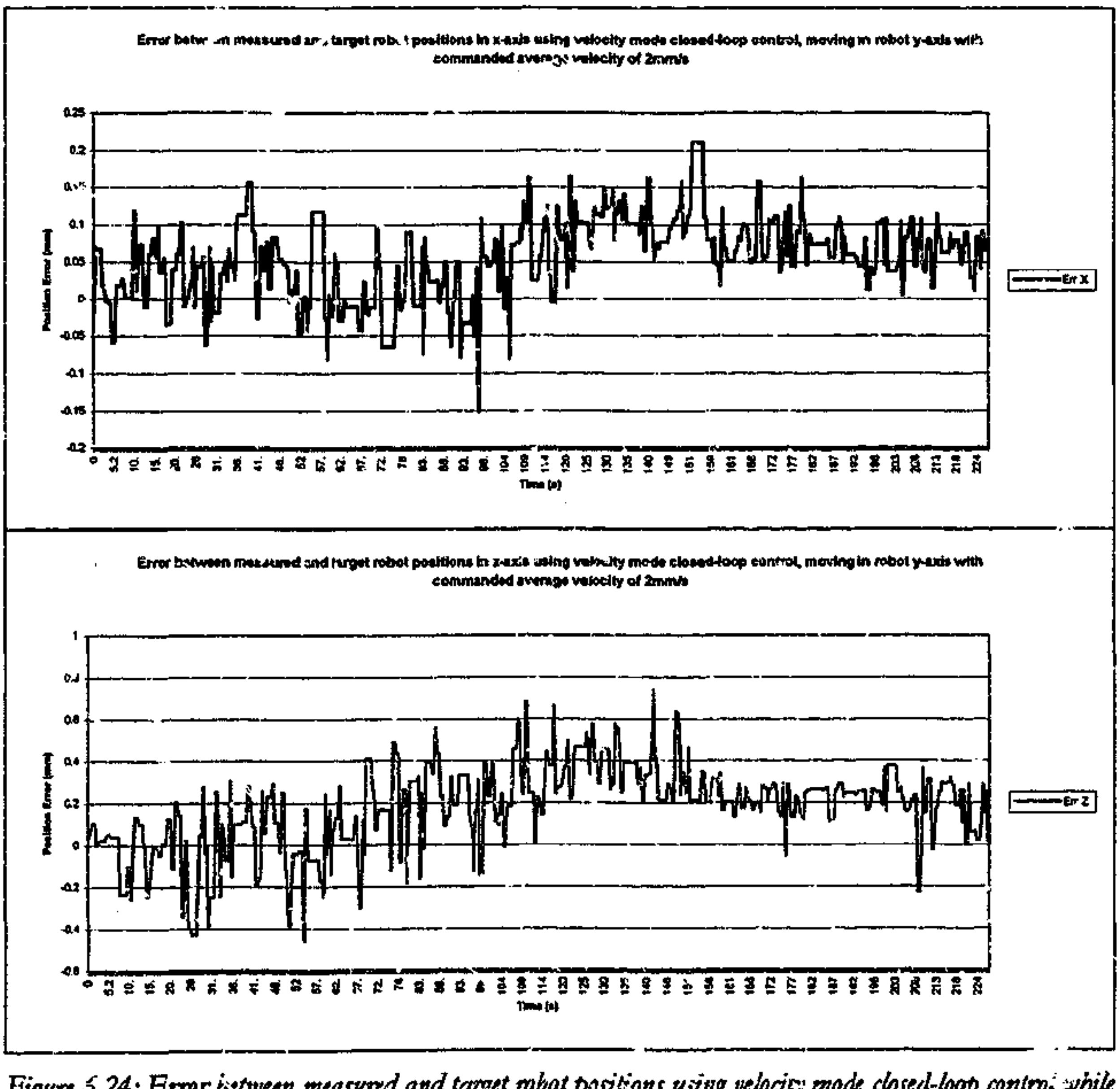


Figure 6.24: Error between measured and target robot positions using velocity mode closed-loop control while moving in robot y-axis with commanded velocity of 2mm/s

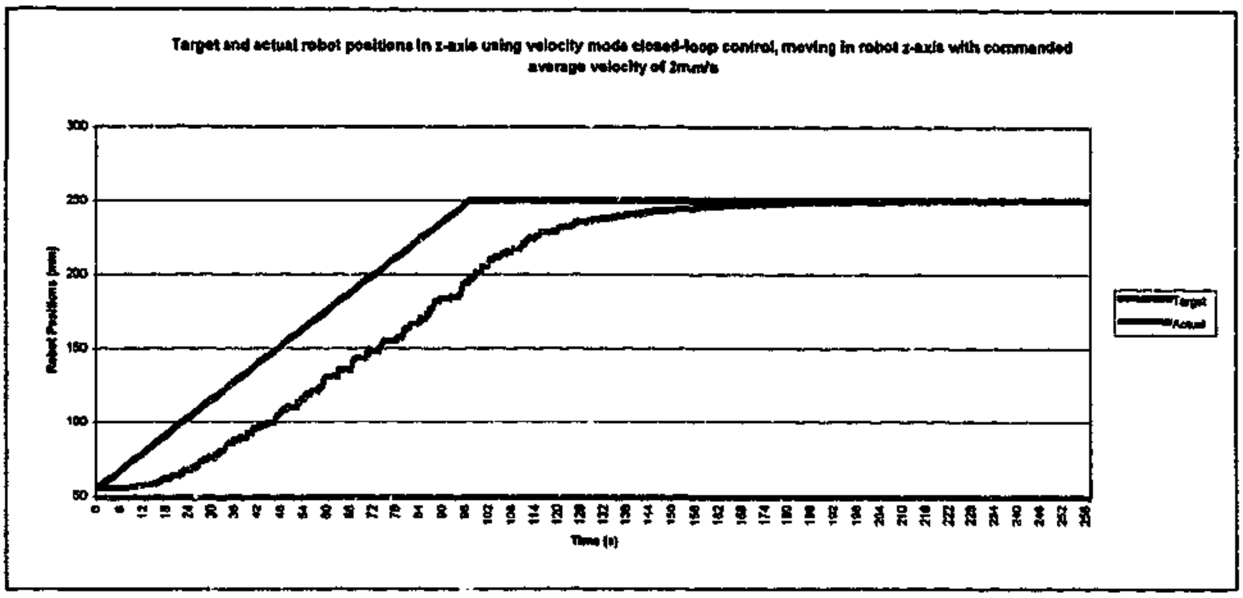


Figure 6.25: Target and actual robot positions in z-axis using velocity mode closed-loop control while moving in robot z-axis with commanded velocity of 2mm/s

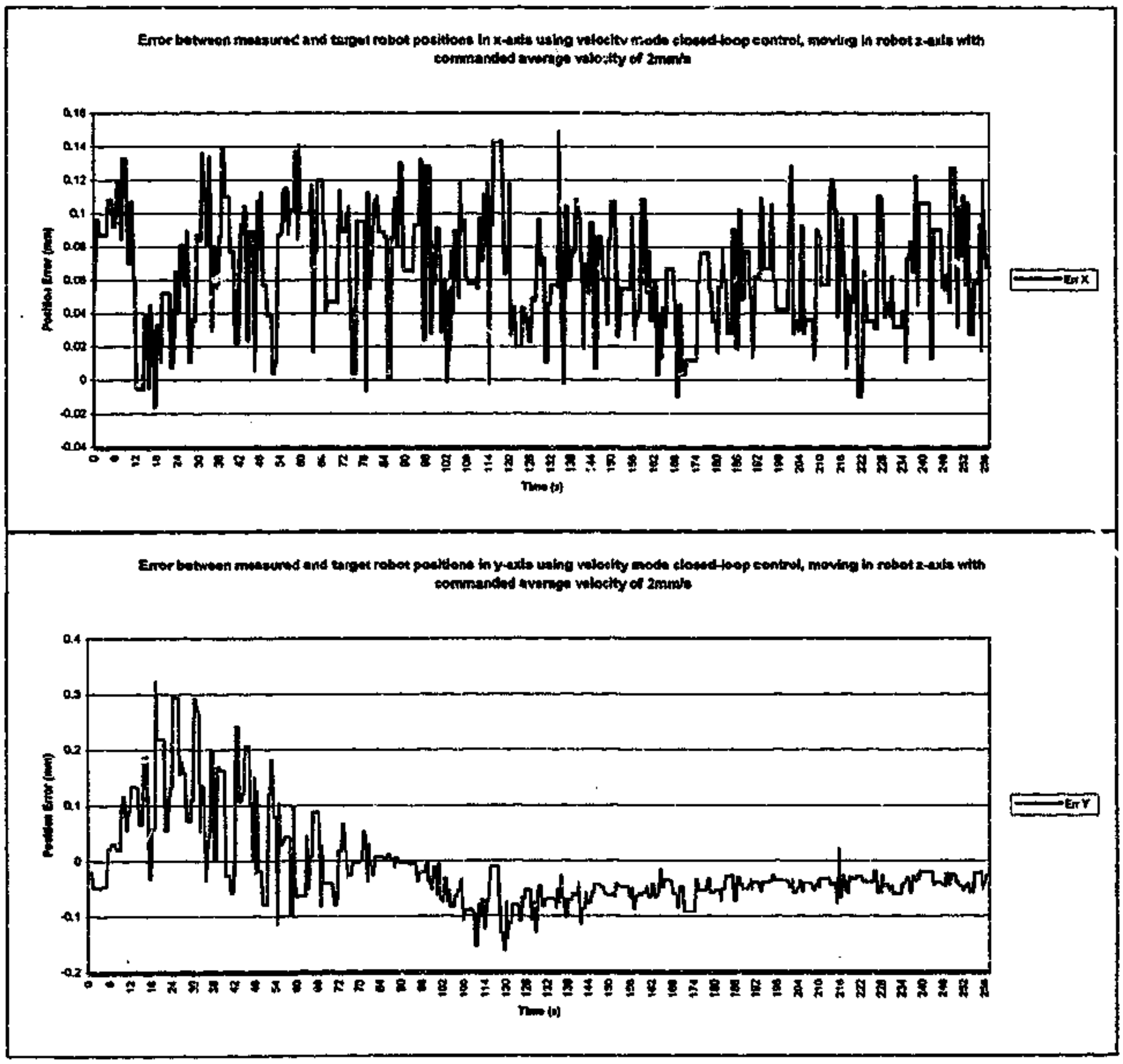


Figure 6.26: Error between measured and target robot positions using velocity mode closed-loop control while moving in robot z-axis with commanded velocity of 2mm/s

Position mode CLC provides better path and positioning accuracy compared to velocity mode CLC. However, in position mode control, the robot has to stop at each target point before moving to the next. This is not desirable in normal robot operations. Path and positioning accuracy of the end-effector in velocity mode CLC can be improved by using a time-delay control algorithm. Moreover, a faster communication interface between the LISM control unit and the robot controller, and a shorter delay for the robot to act upon the command transmitted to the robot controller can further improve path and positioning accuracy in velocity mode CLC.

From the experimental results, it can be concluded that the closed-loop control algorithm using LISM technique improves the steady state positioning accuracy and the path following capability of the robot manipulator. This is accomplished by dynamic measurements of robot positions and compensating the position errors. The final steady state error is close to the resolution of the robot manipulator. The RMS position error of the robot end-effector along the commanded path has been improved to be comparable to the repeatability of the robot, indicating higher path accuracy.

From Chapter 5, it has been shown that the error of the position measured by the experimental apparatus is high. The robot position measurements shown in the experiments have an uncertainty of ±0.5 mm. However, robot manipulators are known to have absolute position errors in the order of 2-10 mm. Although the robot may not be commanded along a path as accurately in the Cartesian space, experiments have indicated that the robot can be commanded to follow closely the path plotted by the LISM apparatus. By improving the accuracy of the LISM apparatus, the proposed closed-loop control methodology is feasible in providing accurate position control of robot manipulators.

6.6 Limitations

There are several aspects that affect the effectiveness of the proposed closed-loop control methodology (CLC). The first limitation is that the robot cannot move at a velocity higher than the tracking velocity of LISM. Secondly, the update rate of the control algorithm affects the effectiveness of the methodology. This rate is dependent on the update rate of the LISM apparatus to perform position measurement, the time required by robot controller to move the robot's end-effector and the delay in communication between the LISM and

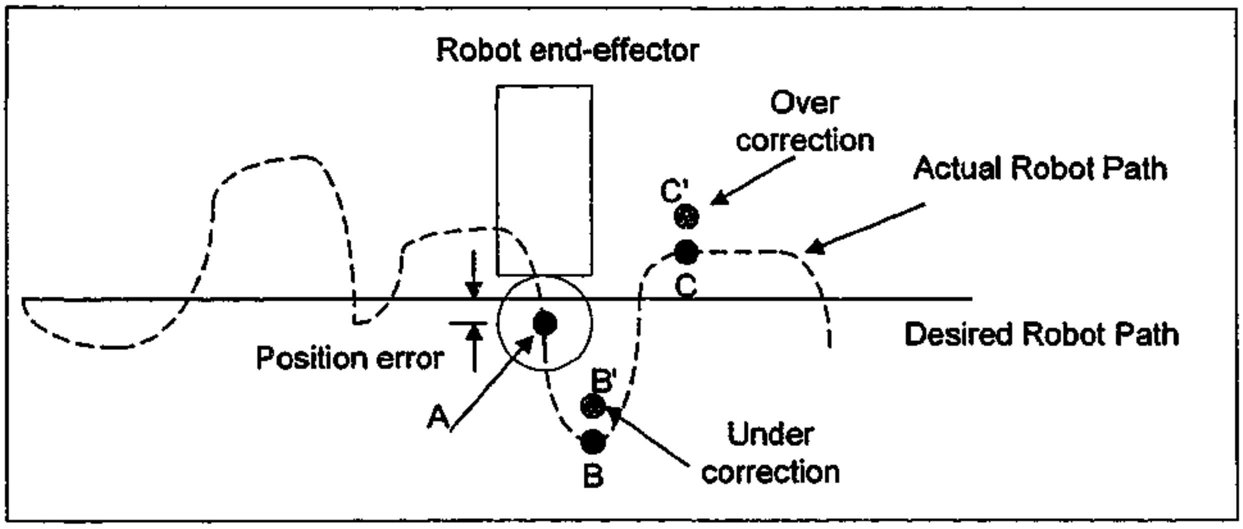


Figure 6.27: Over and under correction of robot due to lengthy update rate

robot controller. A slow CLC update rate may create over or under correction of the position error in velocity mode CLC. This is illustrated by Fig. 6.27. At point A, a position error is detected. At the same time, the end-effector is deviated from A due to some non-linear effects. If the position error compensation is carried out at point B, under correction will occur due to the larger position error at this instant. On the other hand, if the position error compensation is carried out at point C, over correction will occur. This is due to the fact that the position error is now on the other side of the desired path and applying the compensation calculated at A will move the end-effector further away from the desired path.

In the proposed CLC methodology, the position error in Cartesian co-ordinates is transmitted to the robot controller. This position error is converted into the necessary joint angles by the robot controller using the default robot kinematic model. Therefore, the effectiveness of this control methodology is dependent on the accuracy of the kinematic model of the robot manipulator. An inaccurate model will affect the behaviour of the control methodology, as the compensation specification provided by the closed-loop control algorithm is not being accomplished by the robot controller. Further, there are coupling effect among the moving joints, which causes the coupling among the motion directions in Cartesian space. One way of overcoming this limitation is by controlling the robot joints directly. The position error is first converted to the joint angles using the accurate kinematic and dynamic model of the robot. The joint angles computed are then transferred to the robot controller. However, the computation of the joint angles from the kinematic and dynamic model has shown to be time consuming and computationally inefficient [4, 7, 68]. Further, the parameters of the model rely on the instantaneous joint positions and velocities, which has to be obtained from the robot controller. This further increases the load on the communication link, which results in a longer update interval.

The air-path type retroreflector used in this study has a small incident beam acceptance angle. As only the position of the end-effector is controlled, the incident beam will move out of the incident angle range. To rectify this, the gimbal unit developed for orientation measurement in Chapter 4 can be used. The gimbal unit can be commanded to rotate the retroreflector so that it always faces the incoming laser beam. The addition of the gimbal unit also allows the orientation of the end-effector to be measured, which paves the way for the closed-loop control of robot orientation.

The actual path and position accuracy of the closed-loop control methodology is dependent on the resolution of the robot used. By specifying an error limit less than the resolution of the robot will create a 'ringing' effect as the robot controller will be commanded to move the end-effector to a position that cannot be reached due to the low resolution.

6.7 Summary

In this chapter, the development and the evaluation of the proposed closed-loop control methodology using LISM apparatus have been presented. From the experimental results, it can be concluded that the proposed closed-loop control methodology using LISM technique improves the steady state positioning accuracy and the path following capability of the robot manipulator. The final steady state error is close to the resolution of the robot manipulator. The path accuracy of the robot end-effector has also been improved. Next chapter presents the design of the proposed laser interferometry-based guidance control strategy for robot control. Experimental results providing a fundamental understanding insight of the effects of the methodology on robot performance are also discussed.

Chapter 7

Investigation of Laser Interferometry-based Guidance of Robot Manipulators

7.1 Introduction

In this chapter, a Laser Interferometry-based Guidance (LIG) of the robot manipulator using LISM apparatus is presented. This is a control strategy where the LISM apparatus is used to guide the robot's end-effector to a desired point in Cartesian space along a predefined trajectory [76, 77]. For LIG, the sub-systems are the same as described in Chapter 6, however the manners in which these are utilised are altered. It must be emphasised that the objective of this research is to establish a methodology rather than to meet the requirements for high-speed manipulation. The control algorithm for this methodology, together with the measurement and analysis techniques are described in the following sections. Experiments conducted on an industrial robot are also presented. Several aspects governing the effectiveness of the strategy in the guidance of robot manipulators and the limitations of the strategy are investigated and discussed.

7.2 Principle of Laser Interferometry-based Guidance Using LISM

Laser Interferometry-based Guidance (LIG) is the technique of directing the end-effector of a robot manipulator to a desired position in Cartesian space along a predefined straight-line trajectory by steering the laser beam. In this technique, the LISM apparatus is not only used as a position sensor to measure the dynamic Cartesian position of the end-effector, but also acts as a processing unit for robot path generation and guidance offset compensation in the control algorithm. In order to direct the robot end-effector, the laser beam is displaced by a predefined distance along the generated path, and thus generates a guidance offset. The sensors in the LISM apparatus detect the guidance offset and subsequently perform guidance offset compensation by driving the robot manipulator. A flowchart of the LIG control algorithm is provided in Fig. 7.1.

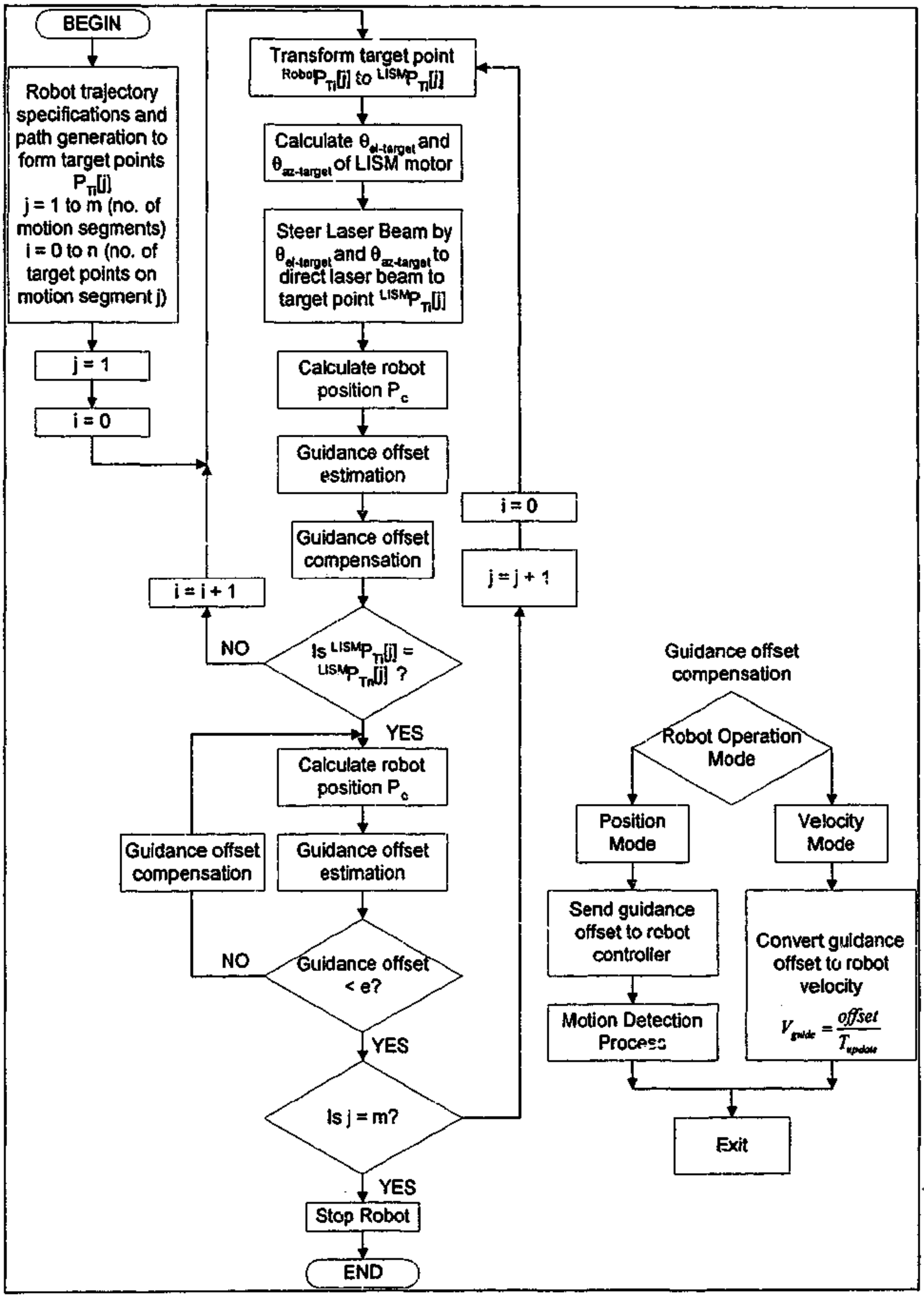

BEGIN
Robot trajectory specifications and path generation to form target points $P_{Ti}[j]$ j = 1 to m (no. of motion segments) i = 0 to n (no. of target points on motion segment j)
j = 1
i = 0
Transform target point $^{Robo}P_{Ti}[j]$ to $^{LISM}P_{Ti}[j]$
Calculate $\theta_{el\text{-}target}$ and $\theta_{az\text{-}target}$ of LISM motor
Steer Laser Beam by $\theta_{el\text{-}target}$ and $\theta_{az\text{-}target}$ to direct laser beam to target point $^{LISM}P_{Ti}[j]$
Calculate robot position P_c
Guidance offset estimation
Guidance offset compensation
i = i + 1
NO
Is $^{LISM}P_{Ti}[j]$ = $^{LISM}P_{Tn}[j]$?
YES
i = 0
j = j + 1
Calculate robot position P_c
Guidance offset compensation
Guidance offset estimation
NO
Guidance offset < e?
YES
Is j = m?
YES
Stop Robot
END
Guidance offset compensation
Robot Operation Mode
Position Mode
Velocity Mode
Send guidance offset to robot controller
Motion Detection Process
Convert guidance offset to robot velocity
$V_{guide} = \frac{offset}{T_{update}}$
Exit

Figure 7.1: Flowchart of LIG control algorithm

7.3 Control Methodology

As shown in Fig. 7.1, the LIG control algorithm consists of 4 main functional modules. These are namely:

1) trajectory specification and path generation;
2) beam steering;
3) guidance offset determination;
4) guidance offset compensation.

The functionality of and the critical aspects within each module will be discussed in the following sections.

7.3.1 Trajectory Specification and Path Generation

The control methodology requires the specification of the desired trajectory for the robot manipulator [76, 77]. This trajectory must consist of the starting point, the end point, and any number of intermediate control points that the end-effector must reach. In order to guide the end-effector to the desired points, the path of the guiding laser has to be first determined. A linear or non-linear path can be used. For the purpose of this study, only linear path is considered.

In the case where the specified start point is different from the current position, a straight-line motion segment will first be generated from the current position to the start point. Subsequent motion segments will be determined to join the start point to the first control point, then from each control point to the next, and finally from the last control point to the end point. The schematic description of the generated path is similar to Fig. 6.3. In order to maintain dynamic measurements of the end-effector's position, a discrete number of via points, to be followed by the laser beam, are generated and plotted along the generated motion segments. These via points are calculated based on the length of each motion segments and the step size specified. The step size between via points must not be greater than the maximum lateral offset required by the laser interferometer to maintain interference. The generated path with the inclusion of the target points is shown in Fig. 6.4. The control software uses all these target points (start, end, control and via points) to direct the laser beam. It must be emphasised that these points are with respect to the robot manipulator's tool co-ordinate frame.

7.3.2 Beam Steering

In view of the fact that the target points are specified with respect to the robot manipulator's co-ordinate frame, these target points must be transformed into the LISM's frame. This is performed by the following equation:

$$^{LISM}P_{\text{target}} = {}^{LISM}A_{Robot} \times {}^{Robot}P_{\text{target}} \qquad (7.1)$$

where $^{LISM}P_{\text{target}}$ is the target point's position with respect to the LISM's co-ordinate frame, $^{LISM}A_{Robot}$ is the transformation matrix from robot's tool co-ordinate frame to LISM's co-ordinate frame, and $^{Robot}P_{\text{target}}$ is the target point's position with respect to the robot's co-ordinate frame.

The detail transformation matrix $^{LISM}A_{Robot}$ is described in Appendix D. From $^{LISM}P_{\text{target}}$, the angular displacement of the motors in LISM apparatus can be obtained using the following equations:

$$\vartheta_{\text{az-target}} = \tan^{-1}\left(\frac{^{LISM}x_{\text{target}}}{^{LISM}y_{\text{target}}}\right) \qquad (7.2)$$

$$\vartheta_{\text{el-target}} = \frac{1}{2}\tan^{-1}\left(\frac{^{LISM}z_{\text{target}}}{\sqrt{^{LISM}x_{\text{target}}{}^{2} + {}^{LISM}y_{\text{target}}{}^{2}}}\right) \qquad (7.3)$$

where $^{LISM}x_{\text{target}}$, $^{LISM}y_{\text{target}}$, $^{LISM}z_{\text{target}}$ are the Cartesian co-ordinates of the target points with respect to the LISM's co-ordinate frame, $\vartheta_{\text{az-target}}$ and $\vartheta_{\text{el-target}}$ are the absolute angular displacement of motor 1 and 2, respectively, to position the laser beam at the target points.

The elevation angular displacement is divided by 2 due to the laser beam travel being perpendicular to the rotation axis of motor 2 (as described in Chapter 3). Commands are issued to move the axes of the beam steering mechanism to position the beam in the required polar configurations, increment by increment, following the target points along the path. When the laser has moved at each increment, there will be a beam offset in the reflected beam. This beam offset is detected through dynamic measurement operations and will be utilised to determine the guidance offset.

7.3.3 Methodology for Guidance Offset Determination

When the laser beam is not pointing to the centre of the retroreflector, there will be an additional displacement of the laser beam as shown in Fig. 3.30. By using the detected laser displacement, together with the PSD measurements and the current angular displacements of the beam steering motors, the current position of the centre of the retroreflector can be obtained by the following equations:

$$ {}^{LISM}x_{current} = L\times\cos\vartheta_{az-current}\cos(2\vartheta_{el-current}) - \frac{\Delta d_1}{\cos\vartheta_{az-current}} \tag{7.4}$$

$$ {}^{LISM}y_{current} = L\times\sin\vartheta_{az-current}\cos(2\vartheta_{el-current}) \tag{7.5}$$

$$ {}^{LISM}z_{current} = L\times\cos(2\vartheta_{el-current}) - \frac{\Delta d_2}{\cos(2\vartheta_{el-current})} \tag{7.6}$$

$$ L=[l_{radial}-\Delta d_1\tan\vartheta_{az-current}-\Delta d_2\tan(2\vartheta_{el-current})] \tag{7.7}$$

where ${}^{LISM}x_{current}$, ${}^{LISM}y_{current}$, ${}^{LISM}z_{current}$ are the current Cartesian co-ordinates of the centre of the retroreflector with respect to the LISM's co-ordinate frame, $\vartheta_{az-current}$ and $\vartheta_{el-current}$ are the current absolute angles of the motors, l_{radial} is the laser displacement recorded , Δd_1 and Δd_2 are the target lateral offsets calculated from Eqs 3.15 and 3.16.

These positions with respect to the LISM's co-ordinate frame were converted to the robot's frame using the following equations:

$$ {}^{Robot}P_{Current} = {}^{Robot}A_{LISM}\times{}^{LISM}P_{Current} \tag{7.8}$$

where ${}^{Robot}P_{Current}$ is the current position with respect to the robot's co-ordinate frame, ${}^{LISM}P_{Current}$ is the current position with respect to the LISM's co-ordinate frame, and ${}^{Robot}A_{LISM}$ is the transformation matrix from LISM's to robot's co-ordinate frame. ${}^{Robot}A_{LISM}$ can be obtained from the inverse of ${}^{LISM}A_{Robot}$ in Eq. 7.1.

The guidance offset ΔP can be estimated by the difference between the target point and the current position:

$$ \Delta P = {}^{Robot}P_{target} - {}^{Robot}P_{Current} \tag{7.9}$$

7.3.4 Methodology for Guidance Offset Compensation

The guidance offset calculated in Eq. 7.9 is corrected by commanding the corresponding axes of the robot manipulators. Two different methods of commanding the robot have been investigated. The first is based on the command of the robot's position. The second is by means of controlling the robot's velocity. In position control mode, the guidance offset is fed to the robot controller. The end-effector of the robot is then commanded to move relative to the current position. As soon as the robot starts moving, a robot motion detection process as described in Section 6.3.1 is triggered. When the robot has stopped, the current robot position is updated. For the case where a via point has been reached, the laser beam is steered to the next via point and the new guidance offset is calculated. If a control point has been reached, the guidance offset is compared to a predefined error limit. If the guidance offset is more than the predefined error limit, this offset is again compensated by commanding the robot end-effector. If the guidance offset is less than the predefined error limit, the laser beam is steered to the next via point on the next motion segment. The predefined error limit can be specified based on the precision required. In this way, the end-effector of the robot is commanded to all the target points, increment by increment, along the specified path.

In velocity control mode, the velocity of the robot is computed during every update interval using the following equation:

$$V_{guide} = \frac{\Delta P}{T_{update}} \tag{7.10}$$

where V_{guide} is the guiding velocity and T_{update} is the LIG update rate. This update rate is dependent on the sampling rate of the LISM apparatus to perform dynamic measurements and to move the axes of the beam steering mechanism, communication delay between the LISM and robot controller, and the time required by the robot controller to move the robot's end-effector.

At each update interval, the current position of the robot is updated and a new guidance offset is calculated using the next via point. As the robot is commanded to move continuously, the LISM apparatus must be able to steer the laser beam to the next via point with the new robot velocity computed and transmitted to the robot before the guidance offset exceeds the maximum lateral offset required by the laser interferometer to maintain interference. Similarly, the guidance offset is compared to the predefined error limit at every control points. The laser beam is steered to the next via point on the next motion segment only when the guidance offset is less than the predefined error limit. In this way, the end-effector of the robot is commanded to move to the target points continuously along the specified path.

7.4 Experimental Set-up

Experiments have been performed to establish the behaviour and capability of the LIG control algorithm for the guidance of robot manipulators. A similar experimental set-up as shown in Fig. 6.5, consists of the experimental LISM apparatus and a Motoman robot manipulator. The same robot tool co-ordinate frame is utilised so that subsequent LISM measurements of the end-effector is recorded with respect to this tool co-ordinate frame.

For the first experiment, the position control mode is used. The end-effector is guided to an end point by moving along a single axis. The Cartesian co-ordinates of the end-effector before the execution of the beam steering module are recorded. This Cartesian co-ordinates are used to determine the position error between the recorded and the target positions. This error shows the effectiveness of the control algorithm in compensating for the previous guidance offsets before the beam is steered to the next target point. The lower the position error, the more effective is the control algorithm. Next, the end-effector is guided to move to an end point in 3-dimensional space. The step size between via points used in this experiment is 0.5 mm. Figs. 7.2 to 7.5 show the error between the recorded and target positions using position mode guidance.

The second experiment involves the command of the robot end-effector in velocity mode. The end-effector is guided to an end point by moving along a single axis. Subsequently, the end-effector is commanded to move to an end point in 3-dimensional space. The step size between via points used in this experiment is 0.5 mm. Similarly, the Cartesian co-ordinates of the end-effector prior to the steering of the beam are recorded. The errors between recorded and target positions are presented in Figs. 7.6 - 7.9. Furthermore, the performance of LIG algorithm using position and velocity modes are compared to identify the behaviour and effects of both control modes.

7.5 Experimental Results and Discussion

From Figs. 7.2 - 7.5, it can be observed that there are fluctuations in the position errors. This is due to the robot controller not being able to move the robot accurately. Moreover, there exist non-linear effects such as friction, stiction, hysteresis, and non-rigid behaviour of the robot manipulator [6, 68, 74]. The z-axis shows the highest fluctuation due to the gravitational forces of the robot manipulator. Another source of error is the position loss in the open-loop stepper motor or misalignment of the encoder in the beam steering mechanism. This results in the deviations of the position of the retroreflector's centre recorded by the LISM apparatus. Furthermore, the update rate of the TPC (Target Position Calculation) is slow as shown in Table 3.2. The previous robot positions may be used in guidance offset determination. The inaccuracy in the measured position and the slow position update result in an inaccurate guidance offset determination, and therefore, an inaccurate position increment is being sent to the robot controller. These errors are being detected through dynamic measurements and are being taken into account in the guidance offset determination. This results in the improved RMS position error when compared to commanding the robot without the LIG control algorithm, as shown in Table 7.1. As indicated, implementation of the LIG control algorithm provides improvement in robot path and positioning accuracy.

Table 7.1: Comparison of RMS and steady state position errors with and without the position mode LIG algorithm

	With position mode guidance RMS Error			Without guidance RMS Error		
Move along	X	Y	Z	X	Y	Z
X	0.065	0.088	0.073	0.286	0.112	0.918
Y	0.036	0.030	0.076	0.441	0.153	0.203
Z	0.031	0.016	0.066	0.491	0.177	0.338
XYZ	0.047	0.037	0.056	0.677	0.063	0.610

	With position mode guidance Steady state Error			Without guidance Steady state Error		
Move along	X	Y	Z	X	Y	Z
X	-0.001	-0.016	0.050	0.346	0.166	1.733
Y	0.018	0.028	-0.042	-0.799	-0.262	0.211
Z	0.020	-0.001	0.040	-0.773	0.25	-0.543
XYZ	0.044	0.012	-0.049	0.635	0.011	0.704

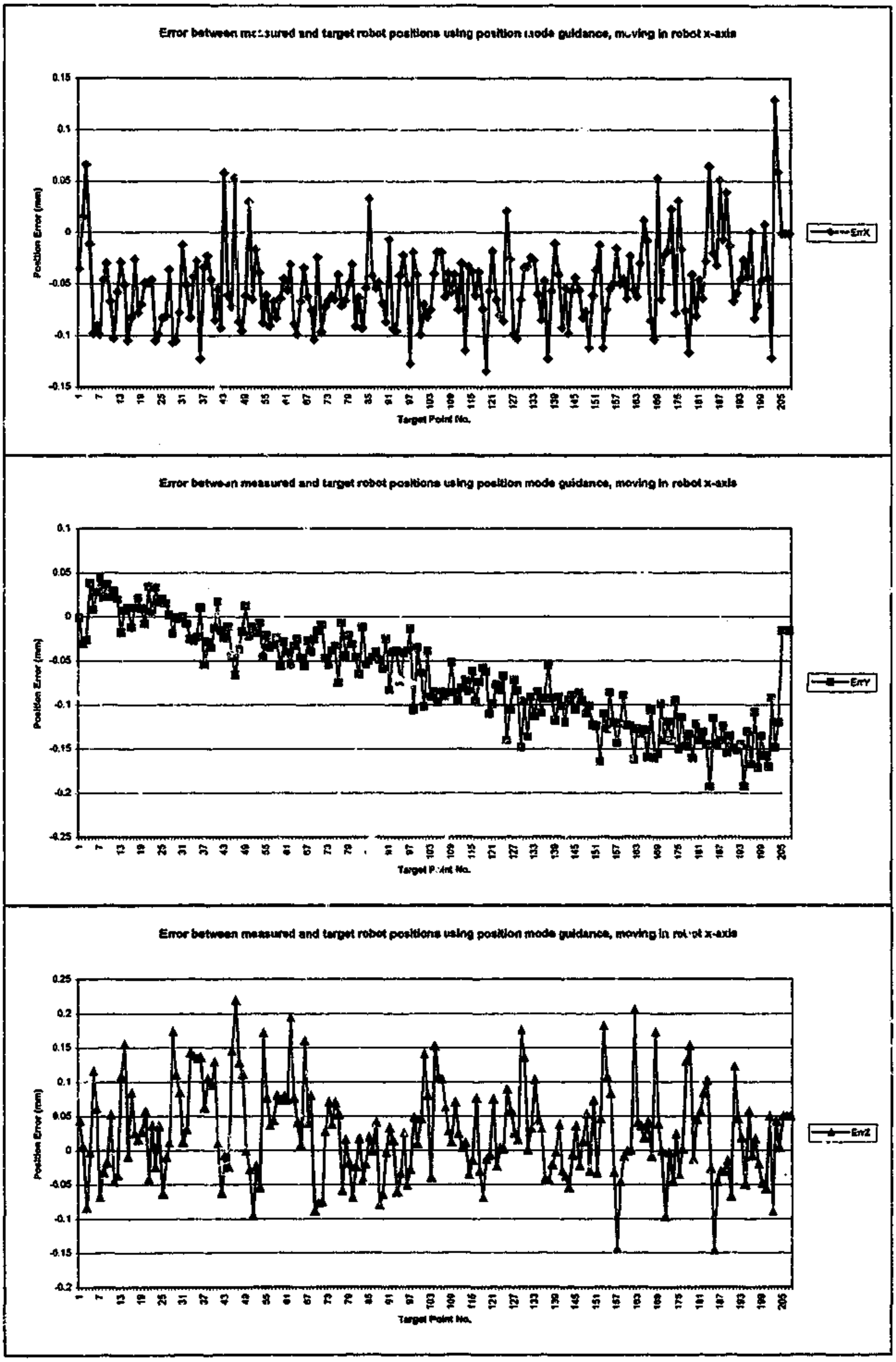


Figure 7.2: Error between measured and target robot positions using position mode guidance while moving in robot x-axis

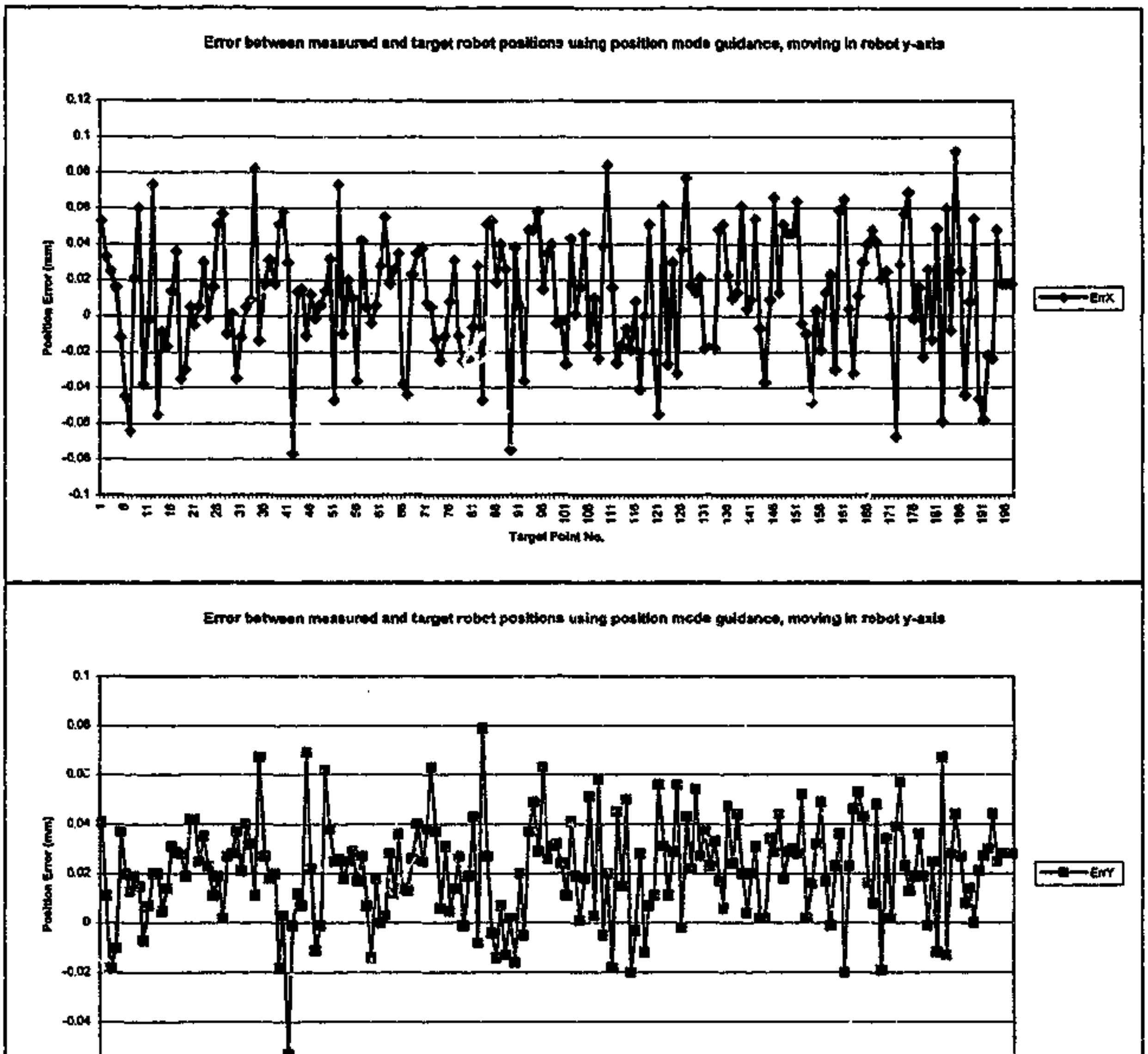


Figure 7.3: Error between measured and target robot positions using position mode guidance while moving in robot y-axis

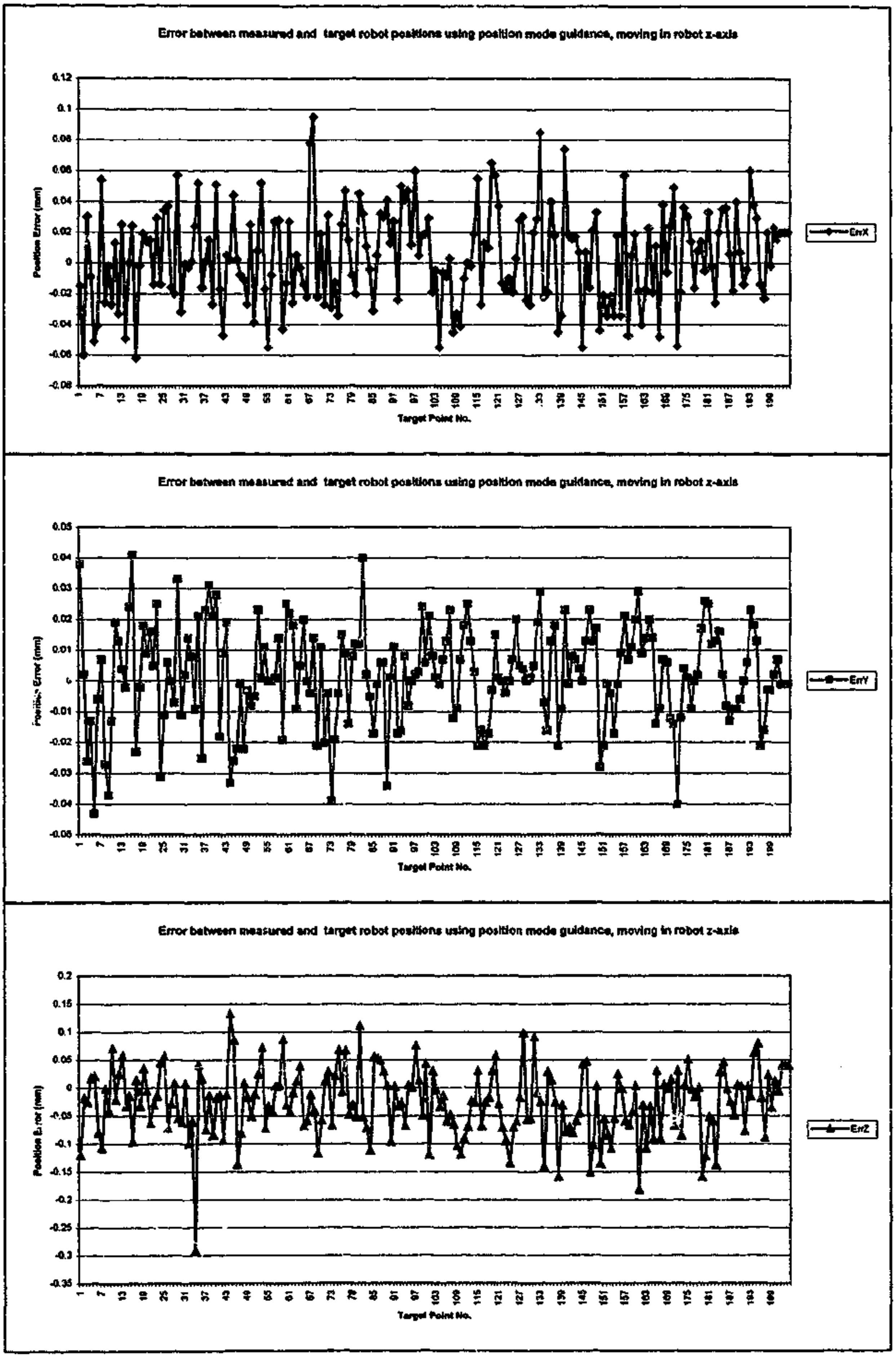


Figure 7.4: Error between measured and target robot positions using position mode guidance while moving in robot z-axis

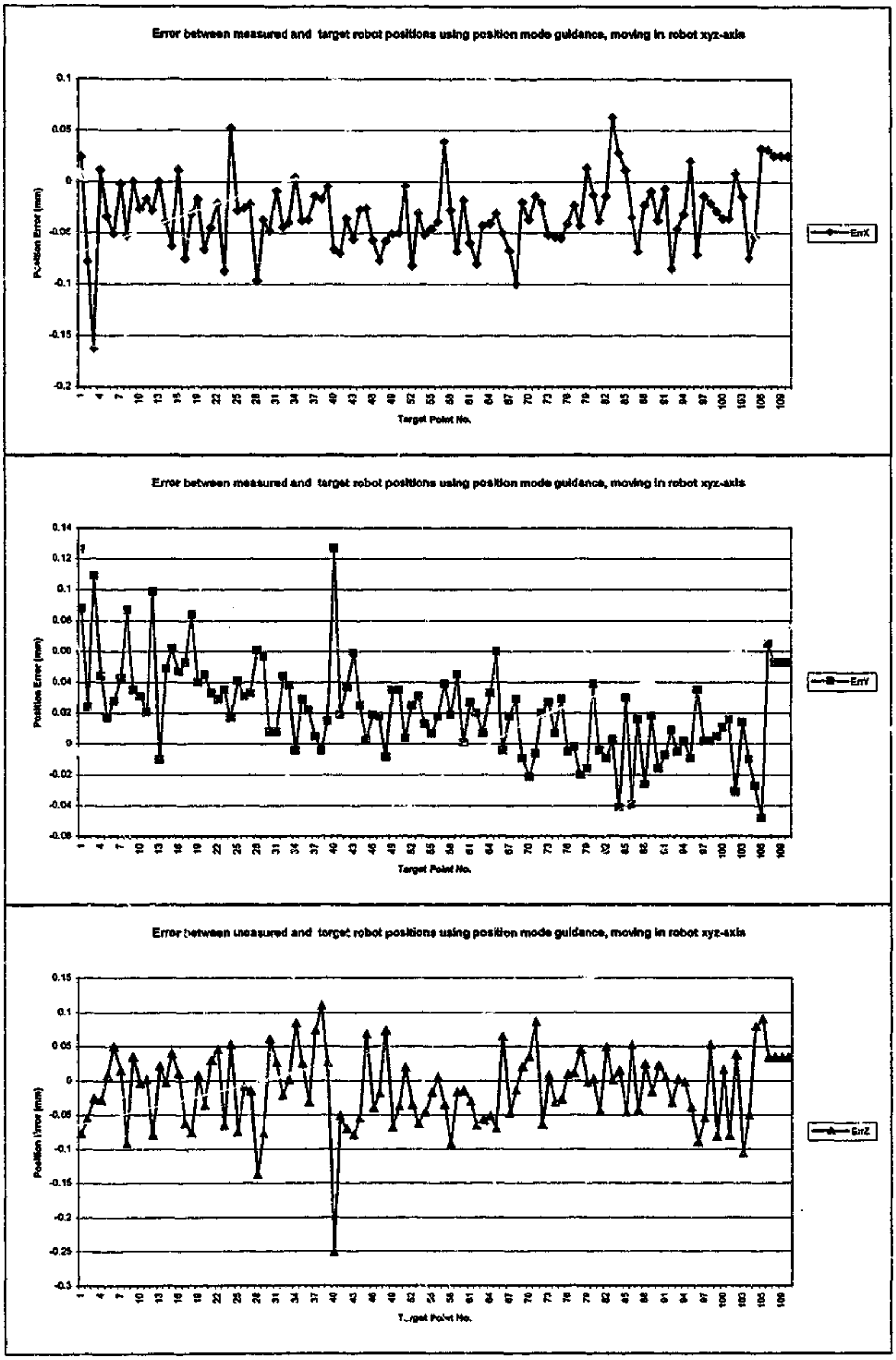


Figure 7.5: Error between measured and target robot positions using position mode guidance while moving in robot xyz-axis

From Figs. 7.6 - 7.9, it can be observed the magnitude of the error fluctuations is higher along the axis that the robot is commanded to move along (i.e. higher fluctuation in x error when the robot is commanded to move along the robot x-axis). This is because when a command is being sent to the robot using the TurboLink interface, there is a delay of 300ms before the command is being acted upon by the robot. At the beginning of each update interval, the robot is still approaching the via point due to this delay. This results in a larger guidance offset when the next via point is used to compute the robot velocity. This higher velocity is then used to move the robot to the next via point, resulting in overshoot of the robot during the next update interval. Following that, the robot is subsequently commanded to move at a lower velocity to the next via point, resulting in the lagging of the robot in the following update interval. A time-delay control algorithm is required to improve the error. Further, the error in robot position is also due to the inaccuracy in the velocity calculated in Eq. 7.10, where a constant update rate is used. During the operation, the update rate monitored can be observed to be varying and the communication between the LISM and the robot controller is not being carried out at a constant rate of 66Hz. This results in the higher magnitude of error when compared to the position mode guidance.

For robot motion along a single axis, the high magnitude of fluctuation along the axis that the robot is commanded to move along does not cause any path deviation. Deviation of the robot from the desired path is dominated by the position error in the other two co-ordinates. From Table 7.2, it can be observed that the RMS position error is higher when compared to position mode guidance. Similarly, these errors are being detected and compensated by the LIG methodology, resulting in an improved RMS position error when compared to commanding the robot without the LIG control algorithm.

By using position mode guidance, the final steady state error between the robot position and the end point is about ±0.05 mm. With the use of velocity mode guidance, the final steady state error between the robot position and the end point is about ±0.08 mm. The final steady state error for both modes of guidance is comparable to the resolution of the robot. Position mode guidance provides a better path and positioning accuracy compared to velocity mode guidance. However, in position mode guidance the robot has to stop at each target point before moving to the next. This is not desirable in normal robot operation. Path and positioning accuracy of the end-effector in velocity mode guidance can be improved by having a faster communication interface between the LISM control unit and the robot controller. A time-delay control algorithm is also necessary to account for the communication delay.

In Chapter 5, it has been shown that the uncertainty of the position measured by the experimental apparatus is high. The robot positions shown in the experiments have an uncertainty of ±0.5 mm. However, such industrial robot manipulators have been known to have position errors in the order of 2-10 mm. Although the robot may not be guided along a path as accurately in the operational space, experiments have indicated that the robot can be commanded to follow closely the path plotted by the LISM apparatus. Accurate robot position control with LIG can be established by improving the measurement accuracy of the LISM apparatus.

Table 7.2: Comparison of RMS and steady state position errors with and without the velocity mode LIG algorithm

	With velocity mode guidance RMS Error			Without guidance RMS Error		
Move along	X	Y	Z	X	Y	Z
X	0.373	0.085	0.181	0.286	0.112	0.918
Y	0.070	0.246	0.176	0.441	0.153	0.203
Z	0.043	0.064	0.301	0.491	0.177	0.338
XYZ	0.301	0.172	0.237	0.677	0.063	0.610

	With velocity mode guidance Steady state Error			Without guidance Steady state Error		
Move along	X	Y	Z	X	Y	Z
X	0.060	-0.083	-0.045	0.346	0.166	1.733
Y	-0.029	-0.064	-0.074	-0.799	-0.262	0.211
Z	0.044	0.012	-0.049	-0.773	0.25	-0.543
XYZ	-0.038	-0.021	0.036	0.635	0.011	0.704

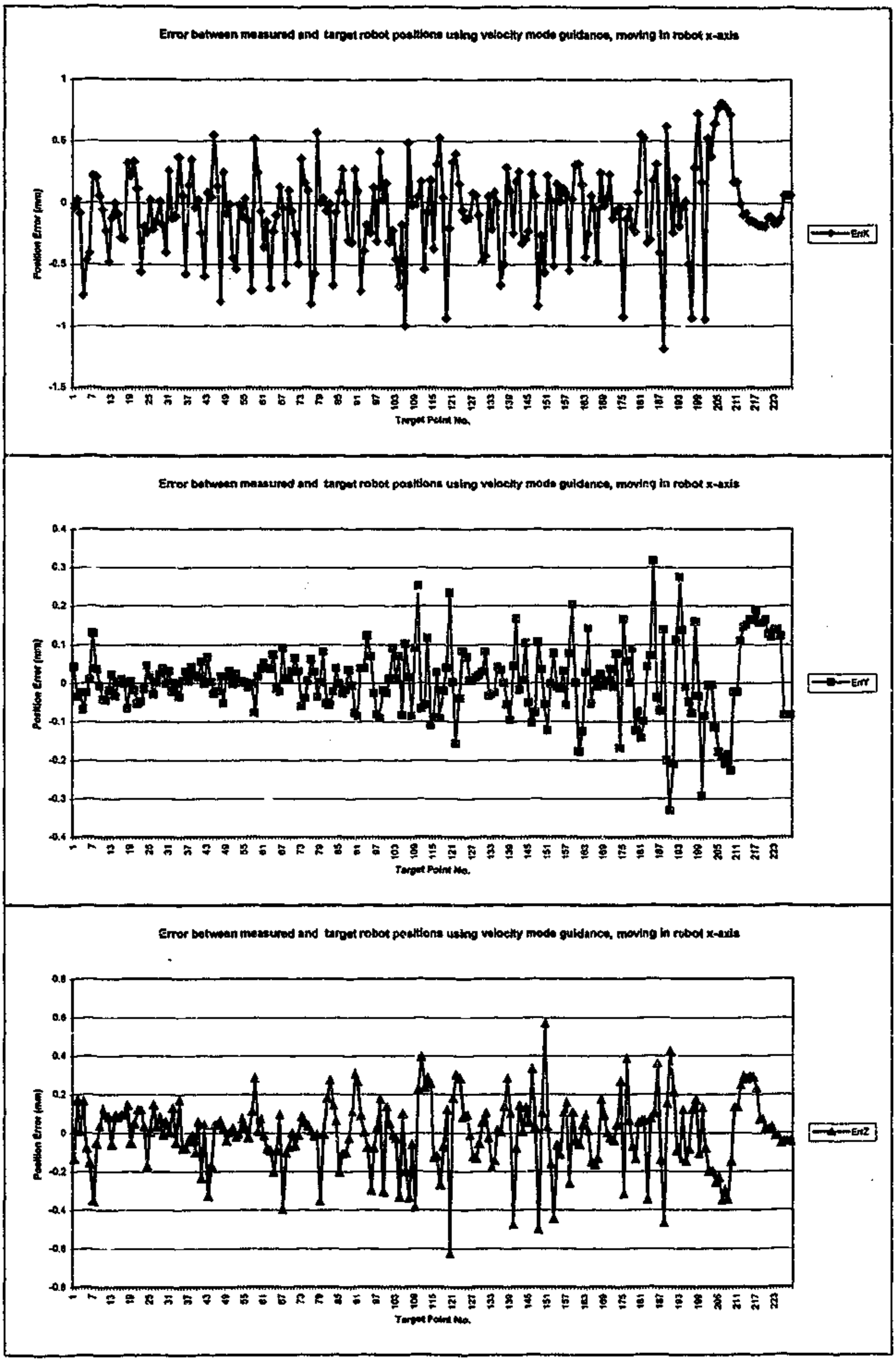


Figure 7.6: Error between measured and target robot positions using velocity mode guidance while moving in robot x-axis

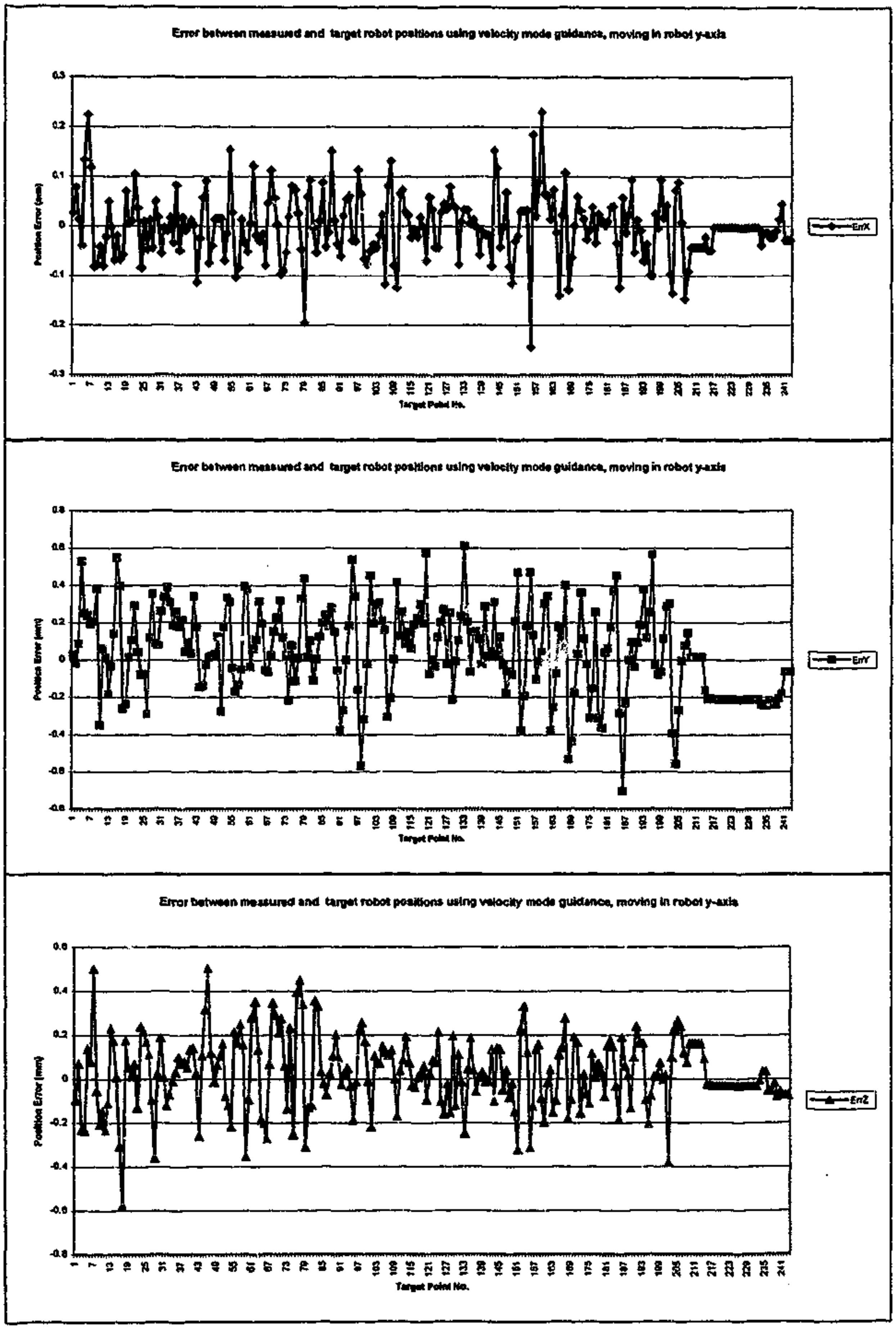


Figure 7.7: Error between measured and target robot positions using velocity mode guidance while moving in robot y-axis

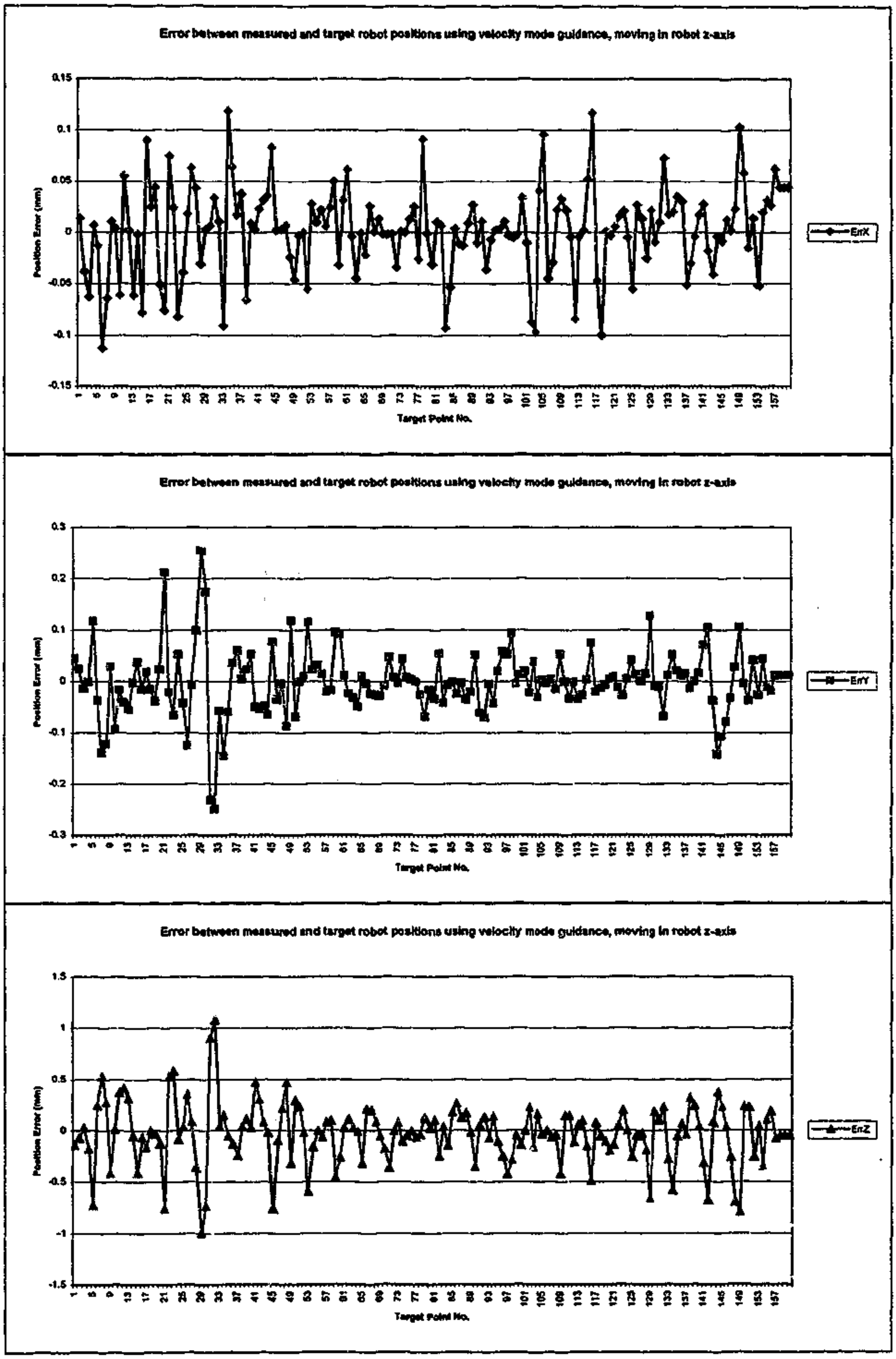


Figure 7.8: Error between measured and target robot positions using velocity mode guidance while moving in robot z-axis

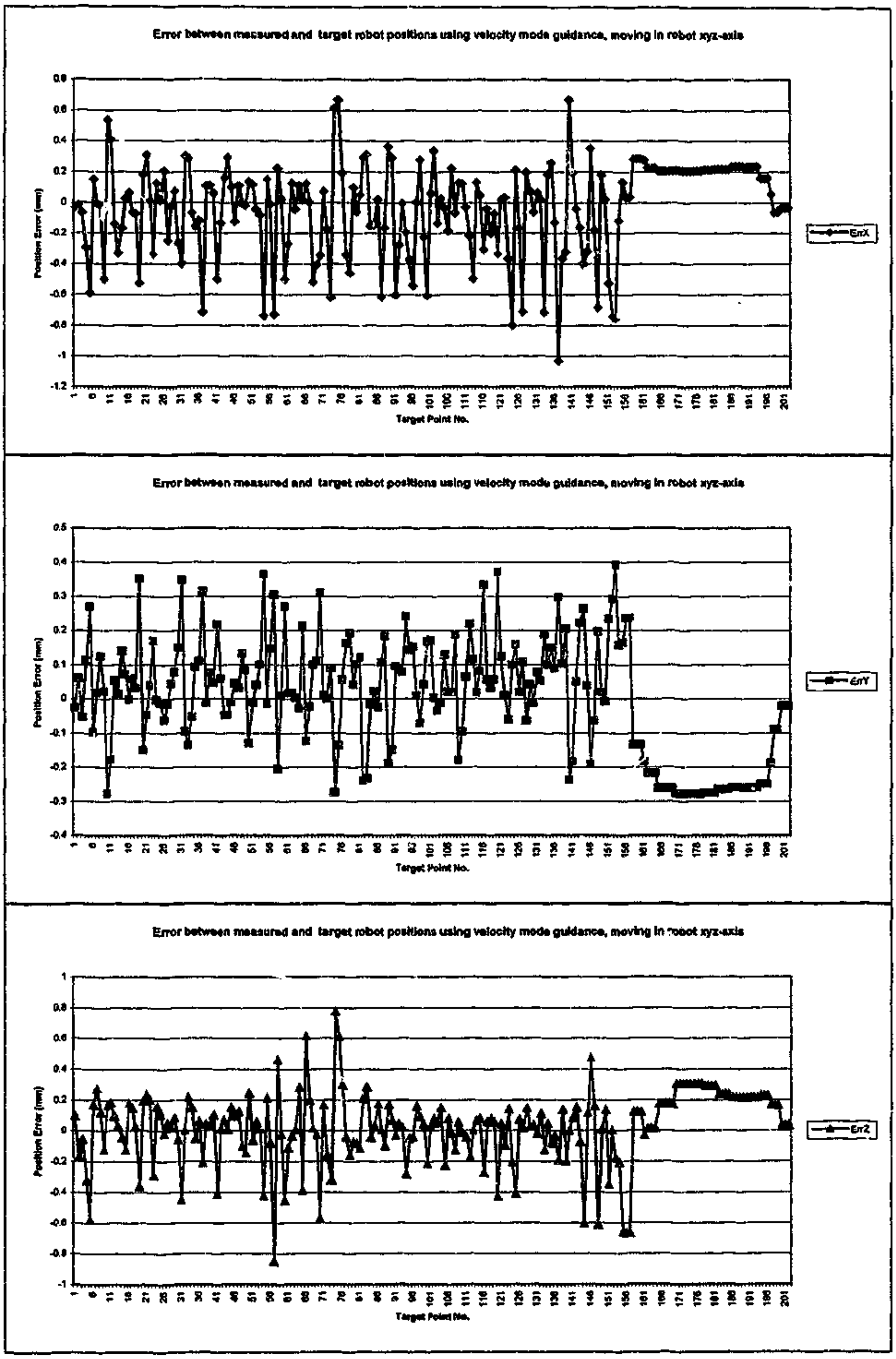


Figure 7.9: Error between measured and target robot positions using velocity mode guidance while moving in robot xyz-axis

7.6 Limitations

Section 7.5 shows that the LIG control algorithm is feasible in providing position control of a robot manipulator to follow the path dictated by the LISM apparatus. However, there are several aspects that affect the effectiveness of the proposed strategy. The first limitation is that the step size between target points, as described in Section 7.3.1, must be less than the maximum lateral offset required by the laser interferometer to maintain interference. Further, this step size is also dependent on the magnitude of vibration experienced by the robot end-effector during motion. This is because the laser beam is not following the motion of the robot in this case. A high magnitude of vibration may create a lateral offset larger than the maximum lateral offset required by the laser interferometer to maintain interference. Laser displacement measurements acquired will not be reliable, and thus dynamic measurements of the robot's position cannot be carried out.

In velocity mode guidance, a step size of 0.5 mm is used. With a LIG update rate of 400 milliseconds, the maximum velocity that the robot can be commanded to move is 1.25 mm/s. In order to increase the robot velocity, the maximum lateral offset and the LIG update rate has to be improved. The lateral offset can be increased by using a laser interferometer with higher beam power and larger beam diameter. As described in Section 7.3.4, the LIG update rate is dependent on the update rate of LISM apparatus and the robot controller. It is also affected by the delay in the communication interface provided by the robot controller. A faster processing unit is required to improve the response time of the LISM apparatus to perform dynamic measurements as well as controlling the axes of the beam steering mechanism. Faster processing unit can also ensure a constant communication rate between the PC and the robot controller, resulting in a more accurate velocity commands. An open architecture robot controller with a shorter delay between receiving and acting upon velocity command is necessary to improve the LIG control algorithm update rate. Additionally, the LIG control algorithm update rate affects the path accuracy of the LIG control strategy in velocity mode. A slow update rate may create over or under correction of the guidance offset as described in Section 6.6.

In the proposed LIG control algorithm, the guidance offset in Cartesian co-ordinate is transmitted to the robot controller. Therefore, the effectiveness of LIG control strategy is dependent on the accuracy of the kinematic model of the robot manipulator. An inaccurate model will affect the effectiveness of the LIG control strategy as discussed in Section 6.6.

The air-path type retroreflector used in this study has the same limitation as discussed in Section 6.6. The addition of the gimbal unit can allow the orientation of the end-effector to be measured, which paves a way for the guidance of robot orientation.

As shown in Eq. 7.4 - 7.7, the computation of the position of the centre of the retroreflector when the incident beam is off-centre is dependent on the beam offset detected by the PSD. Inaccurate PSD readings will affect the precision of the guidance offset calculated using Eq. 7.9. Moreover, imperfections in the retroreflector as described in Chapter 5 may cause the incident beam not being reflected through an angle of 180°, but rather with a slight angular deviation. This angular deviation results in inaccurate PSD readings. This inaccuracy in PSD readings increases with the distance between the LISM apparatus and the robot end-effector. Inaccurate guidance offset calculated will affect the effectiveness of the guidance offset compensation in the LIG control strategy.

The path and position accuracy of the LIG control strategy is dependent on the resolution of the robot used. By specifying an error limit less than the resolution of the robot will create a 'ringing' effect as the robot controller will be commanded to move the end-effector to a position that cannot be reached due to low resolution.

7.7 Summary

In this chapter, the development and the evaluation of the proposed Laser Interferometry-based Guidance (LIG) technique have been presented. Experiments have indicated that the robot can be guided to follow the path plotted by the LIG algorithm. The absolute positioning accuracy of the robot can be improved to about 0.1mm. This is much more accurate than most industrial robots with an absolute accuracy of 2-10 mm. The RMS position error of the robot manipulator has been improved to be comparable to the repeatability of the robot, indicating higher path accuracy.

Next chapter presents the conclusions and recommendations for future work.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this study, an experimental laser interferometry-based sensing and measuring (LISM) apparatus for the tracking and dynamic position measurements of a moving target with an unknown trajectory has been developed. The experimental LISM apparatus has allowed the characterisation of sub-systems and their effects. Control strategies for target following and dynamic position measurement has also been developed and applied on the LISM apparatus. The performance of the LISM apparatus was verified by experimental results on a Motoman SK120 robot manipulator. It has been shown that the current experimental set-up is satisfactory for the tracking and the dynamic position measurements of the target.

A methodology for dual PSD-based orientation measurement using a specially developed Gimbal unit has been established and verified by experimental results. It has been shown that the proposed approach can be used to measure the retroreflector's pitch and yaw angles accurately. The accuracy of roll angle measurement is affected by the initial fixed beam offset applied in the set-up. The proposed approach can also be used to rotate the target retroreflector, such that the line of sight for the incoming laser beam is maintained. This approach effectively removes the limitation of small incident acceptance angle for the retroreflector. The orientation measurement methodology can be incorporated into the experimental LISM apparatus to provide for dynamic position and orientation (pose) measurements of the robot end-effector.

A comprehensive derivation of the kinematic model for the experimental LISM apparatus has been established. This model was implemented for the experimental calibration of the LISM apparatus. A total of 21 parameters of the model have been identified. Based on the calibrated model, the accuracy of position measurements made by LISM apparatus was established. This shows an improvement of 83% when compared to position measurements using the uncalibrated model.

An approach to analyse the sources of errors and the uncertainties for the position and orientation measurements acquired by the LISM apparatus and the Gimbal unit, respectively, has also been presented. The dominant variables in the overall uncertainties of the measurements acquired have been established. It has been shown that for position measurements acquired by the LISM apparatus, the dominant variables in the overall uncertainties are the accuracy of the beam steering motors used and the accuracy of the PSD utilised. For orientation measurements, the dominant variable in the overall uncertainties is the accuracy of the PSD utilised. Furthermore, the uncertainty for roll angle measurements is also dependent on the magnitude of the beam offset implemented. The knowledge of these dominant variables allows the correct selection of hardware components to improve the measurement accuracy.

In Chapter 6 and 7, the methodologies for the closed-loop control (CLC) and the laser interferometry-based guidance (LIG) of robot manipulator have been established. Based on the above, the experimental investigations of the proposed methodologies on the Motoman SK120 have been performed. It was observed that the effectiveness of the CLC and the LIG methodologies is affected by the delay for the robot to act upon the command transmitted to the robot controller. This delay contributes to a slow update rate, which results in the over or under correction of the robot's position error. Moreover, the robot kinematic model employed by the robot controller also affects the effectiveness of the CLC and the LIG methodologies because the Cartesian position values are used. In addition, the length of the robot's path to be controlled or guided is restricted by the small incident beam acceptance angle of the target retroreflector. In CLC, the maximum velocity of the robot is, as expected, restricted by the tracking speed of the LISM apparatus. In LIG, the maximum lateral offset required by the laser interferometer to maintain interference and the update rate of the LIG control algorithm restricts the maximum velocity of the robot. Based on the experimental results, it can be concluded that the implementation of the proposed CLC and LIG methodologies improves the robot's positioning accuracy and path following capability. The improved robot's positioning accuracy is comparable to the resolution of the robot.

8.2 Recommendations for Future Work

The research presented in this dissertation is part of a research program being conducted in the Robotic & Mechatronics Research Laboratory. The ultimate goal of this program is to study the issues such as bending and oscillation associated with long-reach manipulators using the experimental LISM apparatus and to establish an effective technique of compensation through a laser-interferometry based closed-loop control and guidance. The current research can be extended in the following areas:

Dynamic orientation measurement

The next step of the experimental apparatus development and subsequent investigation should be the integration of the orientation measurement methodology into the experimental LISM apparatus. This apparatus is essential for providing dynamic position and orientation (pose) measurements of the robot end-effector, as well as increasing the measuring range of the LISM apparatus by maintaining the line of sight of the incident laser beam. This apparatus will also make possible the calibration of the robot end-effector's orientation.

Implementation of hardware and software

The speed of tracking and the accuracy of pose measurement can be improved with the implementation of appropriate hardware and software. Chapter 5 shows that uncertainties of the position and orientation measurements can be reduced by 94% and 98%, respectively, with the utilisation of appropriate hardware. For the software, a real time extension for Windows is required [78]. Alternatively, a PC-DSP hybrid based control architecture can be used (Section 3.6.3). An open architecture motor controller may be used to realise a better tracking performance. Additional control strategies can be designed and implemented for the driving of the beam steering motors in the LISM apparatus. Dynamic modelling of the motors can also be established to improve the performance by removing the coupling effect among the beam steering motors and tumbling of the motors.

CLC and LIG control algorithms

The performance of CLC and LIG control algorithms can be improved by the implementation of a time-delayed control algorithm to account for the delay in communication and the delay for the robot to act upon the command transmitted to the robot controller. Further, the experimental investigations of the CLC and LIG control methodologies on robot orientation can be performed with the implementation of the proposed orientation measurement methodology. This can then be followed by the experimental study of the CLC and LIG control methodologies on a long-reach manipulator.

References

1. J. J. Craig, Introduction to Robotics 2nd edition, Addison Wesley, 1991.
2. ISO 9283, Manipulating industrial robots – performance criteria and related test methods, International Organisation for Standardisation, 1987.
3. R. P. Paul, Robot Manipulators: Mathematics, programming, and control, MIT Press, 1981.
4. C. H. An, C. Atkeson, J. Hollerbach, Model-based control of a robot manipulator, MIT Press, pp. 1-110, 1988.
5. B. C. Jiang, J. T. Black, R. Duraisamy, "A review of recent developments in robot metrology", Journal of Manufacturing Systems, Vol. 7 No. 4, pp. 339-357, 1988.
6. K. Lau, R. J. Hocken, "A Survey of Current Robot Metrology Methods", Annals of the CIRP, Vol. 32 No. 2, pp. 485-488, 1984.
7. J. Owens, "Robotrak: Calibration on a shoestring", Industrial Robot, Vol. 21 No. 6, pp. 10-13, 1994.
8. R. Gooch, "Optical metrology in manufacturing automation", Industrial Robot, Vol. 18 No. 2, pp. 81-87, 1998.
9. M. R. Driels, U. S. Pathre, "Robot calibration using an automatic theodolite", International Journal of Advance Manufacturing Technology, Vol. 9, pp. 114-125, 1994.
10. H. Gander, M. Vincze, J.P. Prenninger, "An external 6-D-sensor for industrial robots", Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems. Yokohama, Japan, pp. 975-978, July 26-30, 1993.
11. M. Vincze, J.P. Prenninger, H. Gander, "A laser tracking system to measure position and orientation of robot end effectors under motion", International Journal of Robotics Research Vol. 13 No. 4, pp. 305-314. 1994.
12. J. P. Prenninger, H. Gander, M. Vincze, "Contactless position and orientation measurement of robot end-effectors", IEEE Proceedings of Robotics and Automation. Vol. 1, pp. 180-185, 1993.
13. B. Shirinzadeh, "Laser interferometry-based tracking for dynamic measurements", Industrial Robot, Vol. 25 No. 1, pp. 35-41, 1998.
14. K. Lau, R. Hocken, L. Haynes, "Robot performance measurements using automatic laser techniques, Robotics and Computer-Integrated Manufacturing", Vol. 2 No. 3/4, pp 227-236, 1985.

15. J. H. Gilby, G. A. Parker, "Laser tracking system to measure robot arm performance", Sensor Review, pp. 180-184, October 1982.

16. J. R. R. Mayer, G. Parker, "A portable instrument for 3-D dynamic robot measurement using triangulation and laser tracking", IEEE Transactions on Robotics and Automation. Vol. 10 No. 4, pp. 504-516, August 1994.

17. H. Weule, U. Reichling, "Optical test system for industrial robots", Summary paper. CLEO '89 Conference on Laser Electro-Optics. Vol. 8, p. 122. 1989.

18. T. A. G. Heeren, F. E. Veldpaus, "An optical system to measure the end-effector position for on-line control purposes", International Journal of Robotics Research Vol. 11 No. 1, pp. 53-63, 1992.

19. O. Nakamura, et. al., "A laser tracking robot-performance calibration system using ball-seated bearing mechanisms and a spherically shaped cat's eye retroreflector", Review of Scientific Instruments, Vol. 65 No. 4, pp. 1006-1011, April 1994.

20. C. Leigh-Lancaster, B. Shirinzadeh, Y. L. Koh, "Development of laser tracking system", IEEE Proceedings of the 4th Annual Conference on Mechatronics and Machine Vision in Practice (M2VIP), Australia, pp. 163-168, September 1997.

21. B. Shirinzadeh, Y. L. Koh, Alvin, "Control strategy for laser tracking and robot dynamic measurements", Proceedings of the 13th ISPE/ International Conference On CAD/CAM, Robotics & Factories of the Future, Vol 2, pp. 886-891, 1997.

22. S. Spiess, M. Vincze, M. Ayromlou, "On the calibration of 6-D laser tracking system for dynamic robot measurements", IEEE Transactions on Instrumentation and Measurement, Vol. 47 No. 1, pp. 270-274, February 1998.

23. G. Alici and R. Daniel, "Experimental comparison of model-based robot position control strategies", Proceedings of the 1993 IEEE/RJS International Conference on Intelligent Robots and Systems, Yokohama, Japan, pp. 76-83, July 1993.

24. Teletrac Corporation, Obtaining ultimate accuracy, Version 6, Teletrac Corporation, March 1996.

25. Zygo Corporation, ZMI-1000 System Manual OMP-0411D, Zygo Corporation, 2000.

26. J. Graeme, Photodiode Amplifiers, McGraw-Hill, 1995.

27. Melles Griot, Melles Griot 1995/96 Catalogue, Melles Griot USA, 1996.

28. UDT Sensors Inc., Standard photo detector component catalogue, UDT Sensors Inc., 1998.

29. W. Zürcher, R. Loser, S. A. Kyle, "Improved reflector for interferometric tracking in three dimensions", Optical Engineering, Vol. 34 No. 9, pp. 2740-2743, September 1995.

30. S. Decker, H. Gander, M. Vincze, J. P. Prenninger, "Dynamic measurement of position and orientation of robots", IEEE Transactions on Instrumentation and Measurement, Vol. 41 No. 6, December 1992.

31. K.M. Filz, M. Vincze, J.P. Prenninger, "Camera system to detect the orientation of a corner cube in real time", Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 6, pp. 1713-1718, 1995.

32. H. Gander, M. Vincze, J. P. Prenninger, "Application of a floating point digital signal processor to the control of a laser tracking system", IEEE Transactions on Control Systems Technology, Vol. 2 No. 4, December 1994.

33. Y. Bao, N. Fujiwara, "Dynamic measurement orientation by LTS", Proceedings of the Japan/USA Symposium on Flexible Automation, Vol. 1, pp. 545-548, 1996.

34. S. J. Ovaska, "FIR prediction using newton's backward interpolation algorithm with smoothed successive differences", IEEE Transactions on Instrumentation and Measurement, Vol. 40 No. 5, pp. 811-815, October 1991.

35. S. J. Ovaska, "Newton-type predictors - A signal processing oriented viewpoint", Signal Processing, Vol. 25, pp. 251-257, 1991.

36. B. Shirinzadeh, P. L. TEOH, "A study of predictive control for laser tracking of robots", Proceedings of Pacific Conference on Manufacturing (PCM98), Brisbane, Australia, pp. 328-333, August 1998.

37. F. C. Demarest, "High-resolution, high-speed, low data age uncertainty, heterodyne displacement measuring interferometer electronics", Measurement Science and Technology, Vol. 9 No. 7, pp. 1024-1030, 1998.

38. P. D. Lawrance and K. Mauch, Real-Time Microcomputer System Design: An Introduction, McGraw-Hill, 1987.

39. Data Translation Inc., DT300 Series detailed specifications, http://www.datatranslation.com.

40. M. R. Spiegel, L. J. Stephens, Schaum's Outline of Theory and Problems of Statistics 3rd Edition, McGraw-Hill, pp. 63, 1999.

41. D. A. Bradley, D. Dawson, N. C. Burd and A. J. Loader, Mechatronics, Chapman & Hall, 1996.

42. Data Translation Inc., ENOB (Effective Number Of Bits) - The Accurate Way to Choose a Data Acquisition Board, http://www.datatranslation.com.

43. Data Translation Inc., Choosing data acquisition boards and software, http://www.datatranslation.com.

44. Data Translation Inc., DataAcq SDK: Getting started manual, Data Translation Inc., USA.

45. Parker Hannifin Corporation, Dynaserv DM & DR Direct Drive Servos User Guide, Parker Hannifin Corporation, Compumotor Division, USA, March 1995.

46. Rorze Corporation, Rorze RD-023MSH Instruction Manual, Rorze Corporation, Hiroshima, Japan.

47. Parker Hannifin Corporation, Engineering Reference Guide, Compumotor Catalogue 8000-4, Parker Hannifin Corporation, Compumotor Division, USA, pp. 330, 2002.

48. Parker Hannifin Corporation, ACR Technical Brochure Version 3.0, Parker Hannifin Corporation, Compumotor Division, USA.

49. Parker Hannifin Corporation, Compumotor Online Support, http://www.compumotor.com.

50. A. Burns, A. Wellings, Real-Time Systems and Programming Languages 2nd Edition, Addison-Wesley, pp. 169-202, 1997.

51. B. Overland, C++ in Plain English 3rd edition, M&T Books, pp299-483. 2001.

52. Parker Hannifin Corporation, AcroLIB API User's Guide Version 3.00, Parker Hannifin Corporation, Compumotor Division, USA, April 1997.

53. Julian Templeman, Beginning Windows NT Programming, Wrox Press, pp 199-568, 2000.

54. Charles Petzold, Programming Windows 5th edition, Microsoft Press, pp 1197-1240, 1999.

55. E. C. Ifeachor, B. W. Jervis, Digital Signal Processing, Addison-Wesley, pp. 614-672, 1998.

56. B. Shirinzadeh, P. L. Teoh, C. W. Foong, "Orientation measurement using vision and non-vision based techniques in laser tracking system", 30th International Symposium on Robotics, Tokyo, Japan, pp. 317-324. 1999.

57. B. Shirinzadeh, P. L. Teoh, C. W. Foong, Y. D. Liu, "Orientation measurement technique in laser interferometry based tracking system for robot manipulator calibration", Proceedings of Pacific Conference on Manufacturing (PCM2000), Detroit, USA, Vol. 2, pp. 788-793, September 2000.

58. P. L. Teoh, B. Shirinzadeh, "Dual position sensitive diode-based orientation measurement in laser interferometry-based sensing and measurement technique", Proceedings of SPIE- The International Society for Optical Engineering, Vol. 4564, pp. 98-106, October 2001.

59. W. Augustyn, P. Davis, "An analysis of polarization mixing errors in distance measuring interferometers", Journal of Vacuum Science and Technology B, Vol. 8 No. 6, pp. 2032-2036, 1990.

60. M. Vincze, J. Prenninger, H. Gander, "A model of tumbling to improve robot accuracy", Journal of Mechanism and Machine Theory, Vol. 30 No. 6, pp. 849-859, 1995.

61. P. L. Teoh, B. Shirinzadeh, C. W. Foong, G, Alici, "The measurement uncertainties in laser interferometry-based sensing and tracking technique", Journal of Measurement, Vol. 32 No. 2, pp 135-150, September 2002.

62. A. Ben-Israel, T. N. E. Greville, Generalised Inverses: Theory and Applications, Wiley-Interscience, pp. 7-38, 1974.

63. ISO, Guide to expression of uncertainties in measurement, International Organisation for Standardisation, Switzerland, 1993.

64. L. Scivicco and B. Siciliano, Modelling and control of robot manipulators, Springer, pp. 213-268, 2000.

65. F. Reyes and R. Kelly, "Experimental evaluation of model-based controllers on a direct-drive robot arm", Journal of Mechatronics, Vol. 11 No. 3, pp. 267-282, 2001.

66. A. De Carli and R. Caccia, "A comparison of some control strategies for motion control", Mechatronics, Vol. 5 No. 1, pp. 61-71, 1995.

67. T. Yasuho *et. al.*, "Identification and model based control of a 6 D. O. F industrial manipulator", Robot Control, Proceedings of the IFAC Symposium, Nantes, France, Vol. 1, pp. 111-117, September 1997.

68. G. Alici, Robot force control for remote drilling in hazardous environments, PhD Thesis, University of Oxford, UK, 1993.

69. P. L. Teoh, B. Shirinzadeh, G. Alici, "Experimental analysis of laser interferometry-based sensing and measuring technique for a 3D dynamic positioning system", Proceedings of Pacific Conference on Manufacturing (PCM2002), Vol. 2, pp. 882-887, November 2002.

70. P. L. Teoh, B. Shirinzadeh, "3D external ground truth feedback sensing for robot manipulators using laser interferometer-based sensing and measurement technique", Proceedings of the 8th IEEE Conference on Mechatronics and Machine Vision in Practice (M2VIP), Hong Kong, pp. 261-265, August 2001.

71. P. L. Teoh, PhD Professional Disputation Thesis, Department of Mechanical Engineering, Monash University, Australia, August 2002.

72. Yaskawa Electric Corporation, Yasnac MRC instructions: turbo functions, Yaskawa Electric Corporation, Japan, 1997.

73. Yaskawa Electric Corporation, Motoman-SK120 Instructions, Yaskawa Electric Corporation, Japan, pp. 18, 1997.

74. G. Alici, and R. W. Daniel, "Static friction effects during hard-on-hard contact tasks and their implications for manipulator design", The International Journal of Robotics Research, Vol 13 No. 6, pp. 508-520, 1994.

75. T. J. Tarn and K. Brady, "A framework for the control of time-delayed telerobotic systems", Robot Control, Proceedings of the IFAC Symposium, Nantes, France, Vol. 1, pp. 599-604, September 1997.

76. B. Shirinzadeh, H. C. Chong, K. C. Lee, P. L. Teoh, "Issues and techniques for interferometry-based laser guidance of a manipulator", The 5th International Conference on Control, Automation, Robotics and Vision, Singapore, Vol. 1, pp. 271-275, 1998.

77. B. Shirinzadeh, P. L. Teoh, C. W. Foong, Y. D. Liu, "A strategy for accurate guidance of a manipulator using laser interferometry-based sensing technique", Sensor Review, Vol. 19 No. 4, pp. 292-299, 1999.

78. Venturcom Inc., RTX: Real time extension datasheet, Venturcom Inc., http://www.vci.com.

Doctoral / MPhil Thesis Library Release Authorisation

www.monash.edu.au/phdschol/examinations/library_release_authorisation.rtf

Privacy Notice: The information on this form is collected for the primary purpose of seeking your consent to release you thesis to the library. If you choose not to complete all the questions on this form, it may not be possible for your thesis to be released to the library. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. If you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer on (03) 9905 6011.

1. Details of the candidate

Full name: TEOH [redacted]

Postal address: [redacted]

Telephone: [redacted] [redacted]

Email address: [redacted]

Title of thesis: A STUDY OF LASER INTERFEROMETRY-BASED SENSING AND MEASURING TECHNIQUE IN ROBOT MANIPULATOR CONTROL AND GUIDANCE

2. Key words

Please nominate the key words which identify the thesis for the purpose of library cataloguing. Please note that some disciplines have their own thesaurus for this purpose.

LASER INTERFEROMETER, ROBOT CONTROL, ROBOT GUIDANCE

3. Consent for use of thesis

Please circle as appropriate

- [✓ I agree] / do not agree that this thesis, held in any form, eg paper, micro, electronic, may be made available for consultation within the Library.
- [✓ I agree] / do not agree that this thesis may be available for reproduction on paper or in micro/electronic form.
- I note that in any case, my consent is required only for the three years following acceptance of my thesis.

The Library, when supplying information to the national bibliographic database, often needs to distinguish between two or more authors of similar name. Your help, through providing the following additional details, would be appreciated.

Date of birth: 13-11-1976

Any other publications:

4. Declaration by candidate

Candidate's signature: [redacted] Date: 9/9/05

5. Ratification by academic unit

This is to ascertain that the Department / School / Centre / Institute has no objection to the candidate's options regarding access to the Library thesis copy. If so, please sign below and return the completed form to: **Monash Graduate School, Building 3D, Clayton Campus.**

Supervisor's signature: [redacted] Date: 12/9/05
(please print name) ASSOCIATE PROFESSOR BIJAN SHIRINZADEH

A Study of Single Laser Interferometry-based Sensing and Measuring Technique in Robot Manipulator Control and Guidance

Volume 2

Submitted By

Pek Loo TEOH
Bachelor of Engineering (Monash University – Australia)

A thesis submitted in fulfillment of the
requirements for the degree of

Doctor of Philosophy in Engineering Science

Department of Mechanical Engineering
Monash University

September 2003

Appendix A

Calculation of PSD resolution

The 4 channels of the PSD within the experimental LISM apparatus give a voltage of V_c at the output of the signal conditioning device. The voltage V_c ($c = 1, 2, 3, 4$) varies from 1 V to 5 V within the PSD range of –3 to 3 mm. For the DAQ board, an input range of ±10V and a resolution of 12 bits were used. The smallest detectable voltage change is thus 0.0048V.

By assuming that the PSD is linear within the ±3 mm range, $V_3 = 5$ V when $V_1 = 1$ V. Moreover, when $dV_3 = 0.0048$, $dV_1 = -0.0048$.

$$\begin{aligned} \text{resolution along X} &= 6.076\times\left(\frac{-1\times 0.0048-25\times -0.0048}{216}\right) \\ &+14.944\left(\frac{1\times 0.0048-5\times -0.0048}{36}\right) \\ &= 0.0152\,mm \end{aligned} \tag{A.1}$$

When $V_3 = 1$ V when $V_1 = 5$ V. Moreover, when $dV_3 = -0.0048$, $dV_1 = +0.0048$.

$$\begin{aligned} \text{resolution along X} &= -5.132\times\left(\frac{-25\times -0.0048-1\times 0.0048}{216}\right) \\ &+14.782\left(\frac{5\times -0.0048-1\times 0.0048}{36}\right) \\ &= -0.0146\,mm \end{aligned} \tag{A.2}$$

Similiarly, $V_2 = 5$ V when $V_4 = 1$ V and $dV_2 = 0.0048$ when $dV_4 = -0.0048$.

$$\begin{aligned} \text{resolution along Y} &= 7.076\times\left(\frac{-1\times 0.0048-25\times -0.0048}{216}\right) \\ &+14.688\left(\frac{1\times 0.0048-5\times -0.0048}{36}\right) \\ &= 0.0155\,mm \end{aligned} \tag{A.3}$$

When $V_2 = 1$ V when $V_4 = 5$ V and $dV_2 = -0.0048$ when $dV_4 = 0.0048$.

$$\begin{aligned} \text{resolution along Y} &= -5.404\times\left(\frac{-25\times -0.0048-1\times 0.0048}{216}\right) \\ &+15.668\left(\frac{5\times -0.0048-1\times 0.0048}{36}\right) \\ &= -0.0154\,mm \end{aligned} \tag{A.4}$$

The resolution of the PSD is thus 15μm.

Appendix B

Beam propagation for laser displacement calculation

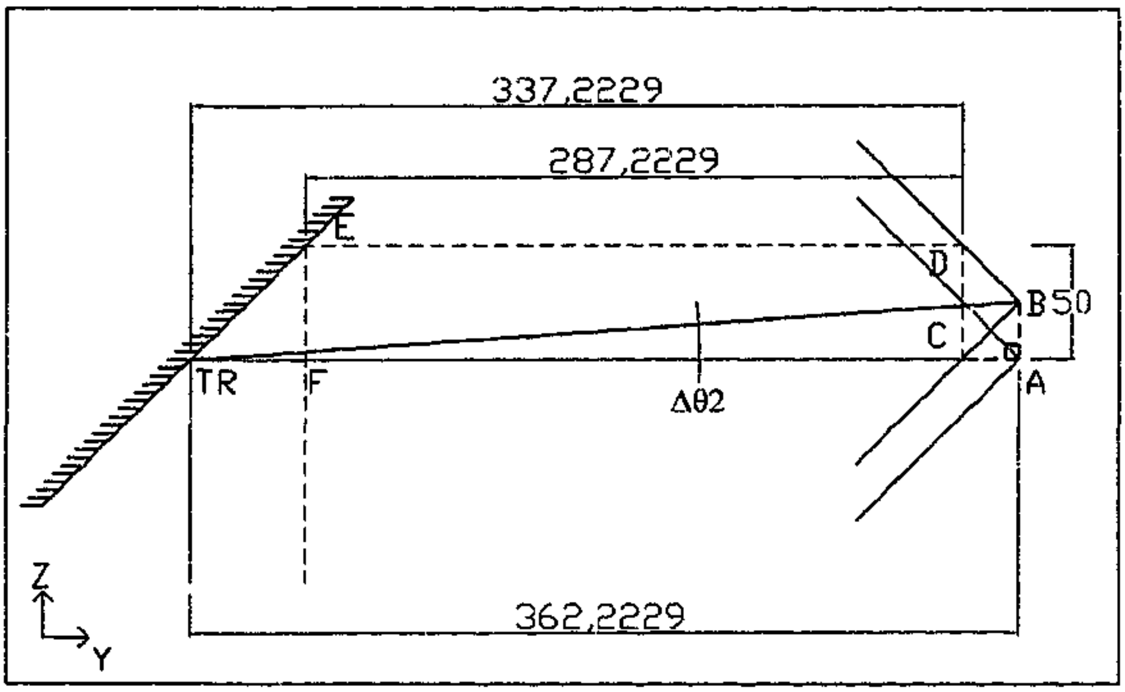


Fig B.1: Beam propagation when the target has moved along z-axis

Fig. B.1 shows the retroreflector's initial position at point A with the laser beam path shown in solid red line. The laser beam is hitting the centre of the retroreflector and there is no beam offset. Point B indicates a new retroreflector position. The dotted red line indicates the propagation of the laser beam before any rotation of the mirror has taken place. The current laser radial distance recorded will be as follows:

$$\begin{aligned}\text{laser radial distance} &= \frac{1}{2}\left(\mathrm{TR} \rightarrow \mathrm{C} + \mathrm{C} \rightarrow \mathrm{D} + \mathrm{D} \rightarrow \mathrm{E} + \mathrm{E} \rightarrow \mathrm{F}\right) \\ &= \frac{1}{2}\left(337.2229 + 50 + 287.2229 + 50\right) \\ &= 362.2229 \\ &= \mathrm{TR} \rightarrow \mathrm{A}\end{aligned} \tag{B.1}$$

The rotation angle $\Delta\vartheta_2$ required to position the beam at the point B can be calculated using the right angle triangle $\mathrm{TR} \rightarrow \mathrm{A} \rightarrow \mathrm{B}$.

Fig. B.2 shows the retroreflector's initial position at point A with the laser beam path shown in solid red line. Point D indicates a new retroreflector position when the retroreflector is displaced along the yz-plane. The dotted red line indicates the propagation of the laser beam before any rotation of the mirror has taken place. The current laser radial distance recorded will be as follows:

$$\begin{aligned}\text{laser radial distance} &= \frac{1}{2}\left(\mathrm{TR}\rightarrow \mathrm{D}+\mathrm{D}\rightarrow \mathrm{E}+\mathrm{E}\rightarrow \mathrm{F}+\mathrm{F}\rightarrow \mathrm{G}\right)\\ &= \frac{1}{2}\left(437.2229+50+387.2229+50\right)\\ &= 462.2229\\ &= \mathrm{TR}\rightarrow \mathrm{B}\end{aligned} \tag{B.2}$$

The rotation angle $\Delta\vartheta_2$ required to position the beam at the point C can be calculated using the right angle triangle $\mathrm{TR}\rightarrow \mathrm{B}\rightarrow \mathrm{C}$.

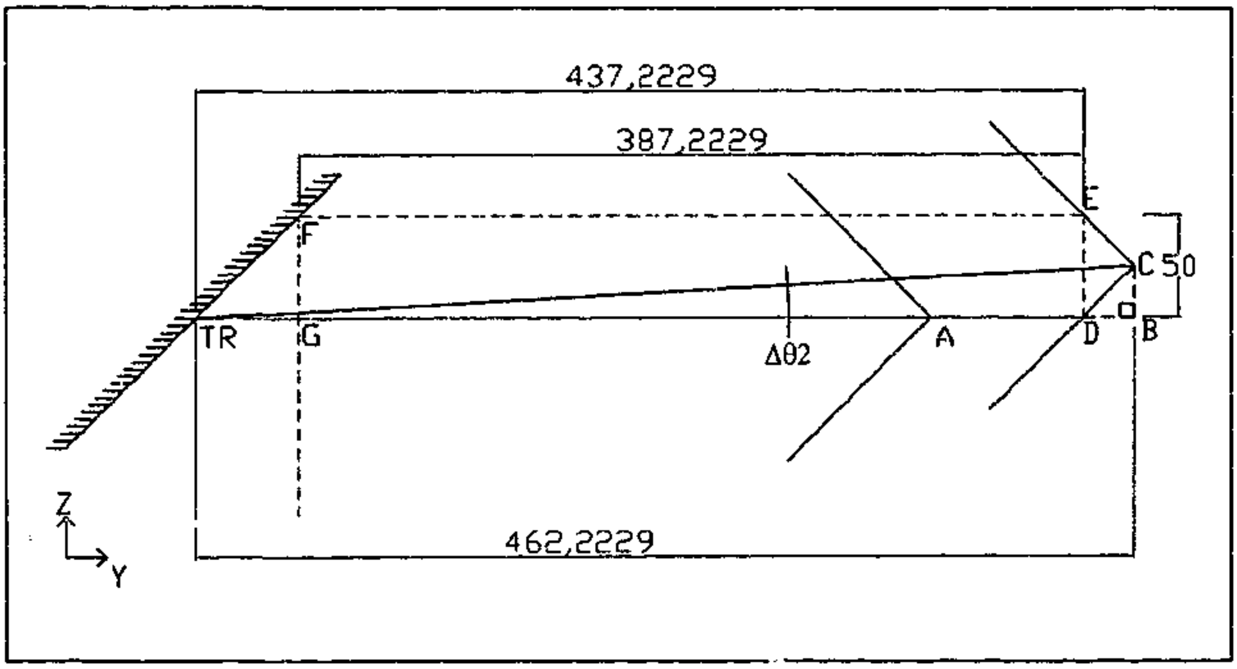


Fig B.2: Beam propagation when the target has moved along yz-plane

Appendix C

Program prepared in MATLAB to solved the generalised inverse of *J* matrix and identify differential change in parameters used in the kinematic model of LISM apparatus

```
syms Malpha1 Mbeta1 Mzed1
syms Malpha2 Mbeta2 Mzed2
syms Malpha3 Mbeta3 Mzed3
syms que1 que2
syms A1 A2 A3
syms xm1 ym1 zm1
syms xm2 ym2 zm2
syms dxm3 dym3 dzm3
syms xL yL zL
syms alphaL betaL gammaL
syms alphaW betaW gammaW
syms xPsd yPsd Zygo

load calibrationFull.txt;
pL=calibrationFull(:,3);
pque1=calibrationFull(:,4);
pque2=calibrationFull(:,5);
LeicaX=calibrationFull(:,6);
LeicaY=calibrationFull(:,7);
LeicaZ=calibrationFull(:,8);

CMalpha1 = cos(Malpha1);
SMalpha1 = sin(Malpha1);
CMalpha2 = cos(Malpha2);
SMalpha2 = sin(Malpha2);

CMBeta1 = cos(Mbeta1);
SMBeta1 = sin(Mbeta1);
CMBeta2 = cos(Mbeta2);
SMBeta2 = sin(Mbeta2);

Cque1 = cos(que1);
Sque1 = sin(que1);

Cque2 = cos(que2);
Sque2 = sin(que2);

Rx1 = [1 0        0        0;
       0 CMalpha1 -SMalpha1 0;
       0 SMalpha1 CMalpha1  0;
       0 0        0        1];
Ry1 = [CMBeta1  0 SMBeta1 0;
       0        1 0       0;
       -SMBeta1 0 CMBeta1 0;
       0        0 0       1];
Tz1 = [1 0 0 0;
       0 1 0 0;
       0 0 1 Mzed1;
       0 0 0 1];

Rx2 = [1 0        0        0;
       0 CMalpha2 -SMalpha2 0;
       0 SMalpha2 CMalpha2  0;
       0 0        0        1];
Ry2 = [CMBeta2  0 SMBeta2 0;
       0        1 0       0;
       -SMBeta2 0 CMBeta2 0;
       0        0 0       1];
Tz2 = [1 0 0 0;
       0 1 0 0;
       0 0 1 Mzed2;
       0 0 0 1];

Am1 = Rx1 * Ry1 * Tz1;
Amirr1 = Am1;
Am2 = Rx2 * Ry2 * Tz2;
Amirr2 = Am2;

Tm1 = [1 0 0 xm1;
       0 1 0 ym1;
       0 0 1 zm1;
       0 0 0 1];
Tm2 = [1 0 0 xm2;
       0 1 0 ym2;
       0 0 1 zm2;
       0 0 0 1];
Tm3 = [1 0 0 dxm3;
       0 1 0 dym3;
       0 0 1 dzm3;
       0 0 0 1];
Rzq1 = [Cque1 -Sque1 0 0;
        Sque1 Cque1  0 0;
        0     0      1 0;
        0     0      0 1];
Rxq2 = [1  0     0      0;
        0  Cque2 -Sque2 0;
        0  Sque2 Cque2  0;
        0  0     0      1];

T1 = Tm1;
T2 = Tm2;
M1s = T1 * Amirr1;
M1 = M1s;
M2s = T2 * Rzq1 * Tm3 * Rxq2 * Amirr2;
M2 = M2s;

CalphaL = cos(alphaL);
SalphaL = sin(alphaL);
CbetaL = cos(betaL);
SbetaL = sin(betaL);
CgammaL = cos(gammaL);
SgammaL = sin(gammaL);

RxL = [1 0       0        0;
       0 CalphaL -SalphaL 0;
       0 SalphaL CalphaL  0;
       0 0       0        1];
RyL = [CbetaL  0 SbetaL 0;
       0       1 0      0;
       -SbetaL 0 CbetaL 0;
       0       0 0      1];
RzL = [CgammaL -SgammaL 0 0;
       SgammaL CgammaL  0 0;
       0       0        1 0;
       0       0        0 1];
TL = [1 0 0 xL;
      0 1 0 yL;
      0 0 1 zL;
      0 0 0 1];
L0s = TL * RzL * RyL * RxL;
L0 = L0s;
```

```
n = M1(1,4)*M1(1,3) + M1(2,4)*M1(2,3) + M1(3,4)*M1(3,3) - M1(1,3)*L0(1,4)-M1(2,3)*L0(2,↙
4)- M1(3,3)*L0(3,4);
d = M1(1,3)*L0(1,3) + M1(2,3)*L0(2,3) + M1(3,3)*L0(3,3);
A1 = n/d;
nLC11 = L0(1,1)*M1(1,3) + L0(2,1)*M1(2,3) + L0(3,1)*M1(3,3);
nLC21 = L0(1,2)*M1(1,3) + L0(2,2)*M1(2,3) + L0(3,2)*M1(3,3);
v = 2*d*d;
c=1-v;
v_m=v/(1-d*d);
B = eye(4);
B11 = nLC21*nLC21*v_m + c;
B12 = -nLC21*nLC11*v_m;
B13 = -nLC11*2*d;
B21 = B12;
B22 = nLC11*nLC11*v_m + c;
B23 = -nLC21*2*d;
B31 = -B13;
B32 = -B23;
B33 = c;
B34 = A1;
B = [B11 B12 B13 0;
     B21 B22 B23 0;
     B31 B32 B33 B34;
     0   0   0   1];

ARefl1 = B;
L1s = L0 * ARefl1;
L1 = L1s;
ok = 1

n = M2(1,4)*M2(1,3) + M2(2,4)*M2(2,3) + M2(3,4)*M2(3,3) - M2(1,3)*L1(1,4)-M2(2,3)*L1(2,↙
4)- M2(3,3)*L1(3,4);
d = M2(1,3)*L1(1,3) + M2(2,3)*L1(2,3) + M2(3,3)*L1(3,3);
A3 = n/d;
nLC13 = L1(1,1)*M2(1,3) + L1(2,1)*M2(2,3) + L1(3,1)*M2(3,3);
nLC23 = L1(1,2)*M2(1,3) + L1(2,2)*M2(2,3) + L1(3,2)*M2(3,3);
v = 2*d*d;
c=1-v;
v_m=v/(1-d*d);
B = eye(4);
B11 = nLC23*nLC23*v_m + c;
B12 = -nLC23*nLC13*v_m;
B13 = -nLC13*2*d;
B21 = B12;
B22 = nLC13*nLC13*v_m + c;
B23 = -nLC23*2*d;
B31 = -B13;
B32 = -B23;
B33 = c;
B34 = A3;
B = [B11 B12 B13 0;
     B21 B22 B23 0;
     B31 B32 B33 B34;
     0   0   0   1];
ARefl3 = B ;
L2s = L1 * ARefl3;
L2 = L2s;
ok = 2

T4 = [1 0 0 xPsd;
      0 1 0 -yPsd;
      0 0 1 Zygo;
      0 0 0 1];
L3s = L2 * T4;
L3 = L3s;%subs(L4s);
ok = 3

Rs = L3;
pX = Rs(1,4);
pY = Rs(2,4);
pZ = Rs(3,4);

Malpha1 = -3*pi/4;
Mbeta1 = 0.0;
Mzed1 = 0.0;
xm1 = 0.0;
ym1 = 465.13;
zm1 = 162;
xm2 = 0.0;
ym2 = 465.13;
zm2 = 548.365;
Malpha2 = pi/4;
Mbeta2 = 0.0;
Mzed2 = 0.0;
dxm3 = 0.0;
dym3 = 0.0;
dzm3 = 0.0;

xPsd = 0.0;
yPsd = 0.0;

alphaL = -pi/2;
betaL = 0.0;
gammaL = 0.0;
xL = 0.0;
yL = 0.0;
zL = 162;

A = L3;
R = subs(A);
ok = 4

el_1 =diff(pX, 'Malpha1');
el_2 =diff(pX, 'Mbeta1');
el_3 =diff(pX, 'Mzed1');
el_4 =diff(pX, 'Malpha2');
el_5 =diff(pX, 'Mbeta2');
el_6 =diff(pX, 'Mzed2');
el_7 =diff(pX, 'xm1');
el_8 =diff(pX, 'ym1');
el_9 =diff(pX, 'zm1');
el_10=diff(pX, 'xm2');
el_11=diff(pX, 'ym2');
el_12=diff(pX, 'zm2');
el_13=diff(pX, 'dxm3');
el_14=diff(pX, 'dym3');
el_15=diff(pX, 'dzm3');
el_16=diff(pX, 'xL');
el_17=diff(pX, 'yL');
el_18=diff(pX, 'zL');
el_19=diff(pX, 'alphaL');
el_20 =diff(pX, 'betaL');
```

```
e1_21 =diff(pX, 'gammaL');

e2_1 =diff(pY, 'Malpha1');
e2_2 =diff(pY, 'Mbeta1');
e2_3 =diff(pY, 'Mzed1');
e2_4 =diff(pY, 'Malpha2');
e2_5 =diff(pY, 'Mbeta2');
e2_6 =diff(pY, 'Mzed2');
e2_7 =diff(pY, 'xm1');
e2_8 =diff(pY, 'ym1');
e2_9 =diff(pY, 'zm1');
e2_10=diff(pY, 'xm2');
e2_11=diff(pY, 'ym2');
e2_12=diff(pY, 'zm2');
e2_13=diff(pY, 'dxm3');
e2_14=diff(pY, 'dym3');
e2_15=diff(pY, 'dzm3');
e2_16=diff(pY, 'xL');
e2_17=diff(pY, 'yL');
e2_18=diff(pY, 'zL');
e2_19=diff(pY, 'alphaL');
e2_20 =diff(pY, 'betaL');
e2_21 =diff(pY, 'gammaL');

e3_1 =diff(pZ, 'Malpha1');
e3_2 =diff(pZ, 'Mbeta1');
e3_3 =diff(pZ, 'Mzed1');
e3_4 =diff(pZ, 'Malpha2');
e3_5 =diff(pZ, 'Mbeta2');
e3_6 =diff(pZ, 'Mzed2');
e3_7 =diff(pZ, 'xm1');
e3_8 =diff(pZ, 'ym1');
e3_9 =diff(pZ, 'zm1');
e3_10=diff(pZ, 'xm2');
e3_11=diff(pZ, 'ym2');
e3_12=diff(pZ, 'zm2');
e3_13=diff(pZ, 'dxm3');
e3_14=diff(pZ, 'dym3');
e3_15=diff(pZ, 'dzm3');
e3_16=diff(pZ, 'xL');
e3_17=diff(pZ, 'yL');
e3_18=diff(pZ, 'zL');
e3_19=diff(pZ, 'alphaL');
e3_20 =diff(pZ, 'betaL');
e3_21 =diff(pZ, 'gammaL');

e_row1=[e1_1 e1_2 e1_3 e1_4 e1_5 e1_6 e1_7 e1_8 e1_9 e1_10 e1_11 e1_12 e1_13 e1_14 e1_1↙
5 e1_16 e1_17 e1_18 e1_19 e1_20 e1_21];
e_row2=[e2_1 e2_2 e2_3 e2_4 e2_5 e2_6 e2_7 e2_8 e2_9 e2_10 e2_11 e2_12 e2_13 e2_14 e2_1↙
5 e2_16 e2_17 e2_18 e2_19 e2_20 e2_21];
e_row3=[e3_1 e3_2 e3_3 e3_4 e3_5 e3_6 e3_7 e3_8 e3_9 e3_10 e3_11 e3_12 e3_13 e3_14 e3_1↙
5 e3_16 e3_17 e3_18 e3_19 e3_20 e3_21];
r1 = subs(e_row1);
r2 = subs(e_row2);
r3 = subs(e_row3);

for i=1:1:134
    que1 = pque1(i,1);
    que2 = pque2(i,1);
    Zygo = pL(i,1);
    r11 = subs(r1);
    r22 = subs(r2);
    r33 = subs(r3);

    E = [r11;r22;r33];
    if i==1
        EE=[E];
    else
       EE=[EE;E];
   end
    i
end
ok = 5

for j=1:1:134
    que1 = pque1(j,1);
    que2 = pque2(j,1);
    Zygo = pL(j,1);

    RR = subs(R);

    XYZ(1,j) = RR(1,4);
    XYZ(2,j) = RR(2,4);
    XYZ(3,j) = RR(3,4);
    ERR=[XYZ(1,j) - LeicaX(j,1);XYZ(2,j) - LeicaY(j,1);XYZ(3,j) - LeicaZ(j,1)];
    if j==1
        er_f=[ERR];
    else
       er_f=[er_f;ERR];
   end
   j
end
ok = 6
LLM=EE\er_f;
ok = 7

para = [Malpha1 Mbeta1 Mzed1 Malpha2 Mbeta2 Mzed2 xm1 ym1 zm1 xm2 ym2 zm2 dxm3 dym3 dzm↙
3 xL yL zL alphaL betaL gammaL]
para_c = para' - LLM;
Malpha1 = para_c(1);
Mbeta1 = para_c(2);
Mzed1 = para_c(3);
Malpha2 = para_c(4);
Mbeta2 = para_c(5);
Mzed2 = para_c(6);
xm1 = para_c(7);
ym1 = para_c(8);
zm1 = para_c(9);
xm2 = para_c(10);
ym2 = para_c(11);
zm2 = para_c(12);
dxm3 = para_c(13);
dym3 = para_c(14);
dzm3 = para_c(15);
xL = para_c(16);
yL = para_c(17);
zL = para_c(18);
alphaL = para_c(19);
betaL = para_c(20);
gammaL = para_c(21);
```

```
syms que1;
syms que2;
syms Zygo;

Malpha1 = -3*pi/4;
Mbeta1 = 0.0;
Mzed1 = 0.0;
xm1 = 0.0;
ym1 = 465.13;
zm1 = 162;
xm2 = 0.0;
ym2 = 465.13;
zm2 = 548.365;
Malpha2 = pi/4;
Mbeta2 = 0.0;
Mzed2 = 0.0;
dxm3 = 0.0;
dym3 = 0.0;
dzm3 = 0.0;
xPsd = 0.0;
yPsd = 0.0;
alphaL = -pi/2;
betaL = 0.0;
gammaL = 0.0;
xL = 0.0;
yL = 0.0;
zL = 162;

load calibrationFull.txt;
pL=calibrationFull(:,3);
pque1=calibrationFull(:,4);
pque2=calibrationFull(:,5);
LeicaX=calibrationFull(:,6);
LeicaY=calibrationFull(:,7);
LeicaZ=calibrationFull(:,8);

CMalpha1 = cos(Malpha1);
SMalpha1 = sin(Malpha1);
CMalpha2 = cos(Malpha2);
SMalpha2 = sin(Malpha2);

CMBeta1 = cos(Mbeta1);
SMBeta1 = sin(Mbeta1);
CMBeta2 = cos(Mbeta2);
SMBeta2 = sin(Mbeta2);

Cque1 = cos(que1);
Sque1 = sin(que1);

Cque2 = cos(que2);
Sque2 = sin(que2);

Rx1 = [1 0        0        0;
       0 CMalpha1 -SMalpha1 0;
       0 SMalpha1 CMalpha1  0;
       0 0        0        1];
Ry1 = [CMBeta1  0 SMBeta1 0;
       0        1 0       0;
       -SMBeta1 0 CMBeta1 0;
       0        0 0       1];
Tz1 = [1 0 0 0;
```

```
       0 1 0 0;
       0 0 1 Mzed1;
       0 0 0 1];

Rx2 = [1 0        0        0;
       0 CMalpha2 -SMalpha2 0;
       0 SMalpha2 CMalpha2  0;
       0 0        0        1];
Ry2 = [CMBeta2  0 SMBeta2 0;
       0        1 0       0;
       -SMBeta2 0 CMBeta2 0;
       0        0 0       1];
Tz2 = [1 0 0 0;
       0 1 0 0;
       0 0 1 Mzed2;
       0 0 0 1];

Am1 = Rx1 * Ry1 * Tz1;
Amirr1 = Am1;
Am2 = Rx2 * Ry2 * Tz2;
Amirr2 = Am2;

Tm1 = [1 0 0 xm1;
       0 1 0 ym1;
       0 0 1 zm1;
       0 0 0 1];
Tm2 = [1 0 0 xm2;
       0 1 0 ym2;
       0 0 1 zm2;
       0 0 0 1];
Tm3 = [1 0 0 dxm3;
       0 1 0 dym3;
       0 0 1 dzm3;
       0 0 0 1];

Rzq1 = [Cque1 -Sque1 0 0;
        Sque1 Cque1  0 0;
        0     0      1 0;
        0     0      0 1];
Rxq2 = [1  0     0      0;
        0  Cque2 -Sque2 0;
        0  Sque2 Cque2  0;
        0  0     0      1];

T1 = Tm1;
T2 = Tm2;
M1s = T1 * Amirr1;
M1 = M1s;
M2s = T2 * Rzq1 * Tm3 * Rxq2 * Amirr2;
M2 = M2s;

CalphaL = cos(alphaL);
SalphaL = sin(alphaL);
CbetaL = cos(betaL);
SbetaL = sin(betaL);
CgammaL = cos(gammaL);
SgammaL = sin(gammaL);

RxL = [1 0       0       0;
       0 CalphaL -SalphaL 0;
       0 SalphaL CalphaL  0;
       0 0        0        1];
RyL = [CbetaL  0 SbetaL 0;
       0       1 0      0;
       -SbetaL 0 CbetaL 0;
       0       0 0      1];
RzL = [CgammaL -SgammaL 0 0;
       SgammaL CgammaL  0 0;
       0       0        1 0;
       0       0        0 1];
TL = [1 0 0 xL;
      0 1 0 yL;
      0 0 1 zL;
      0 0 0 1];
L0s = TL * RzL * RyL * RxL;
L0 = L0s;

n = M1(1,4)*M1(1,3) + M1(2,4)*M1(2,3) + M1(3,4)*M1(3,3) - M1(1,3)*L0(1,4)-M1(2,3)*L0(2,↙
4)- M1(3,3)*L0(3,4);
d = M1(1,3)*L0(1,3) + M1(2,3)*L0(2,3) + M1(3,3)*L0(3,3);
A1 = n/d;
nLC11 = L0(1,1)*M1(1,3) + L0(2,1)*M1(2,3) + L0(3,1)*M1(3,3);
nLC21 = L0(1,2)*M1(1,3) + L0(2,2)*M1(2,3) + L0(3,2)*M1(3,3);
v = 2*d*d;
c=1-v;
v_m=v/(1-d*d);
B = eye(4);
B11 = nLC21*nLC21*v_m + c;
B12 = -nLC21*nLC11*v_m;
B13 = -nLC11*2*d;
B21 = B12;
B22 = nLC11*nLC11*v_m + c;
B23 = -nLC21*2*d;
B31 = -B13;
B32 = -B23;
B33 = c;
B34 = A1;
B = [B11 B12 B13 0;
     B21 B22 B23 0;
     B31 B32 B33 B34;
     0   0   0   1];

ARefl1 = B;
L1s = L0 * ARefl1;
L1 = L1s;%subs(L1s);
ok = 1

n = M2(1,4)*M2(1,3) + M2(2,4)*M2(2,3) + M2(3,4)*M2(3,3) - M2(1,3)*L1(1,4)-M2(2,3)*L1(2,↙
4)- M2(3,3)*L1(3,4);
d = M2(1,3)*L1(1,3) + M2(2,3)*L1(2,3) + M2(3,3)*L1(3,3);
A3 = n/d;
nLC13 = L1(1,1)*M2(1,3) + L1(2,1)*M2(2,3) + L1(3,1)*M2(3,3);
nLC23 = L1(1,2)*M2(1,3) + L1(2,2)*M2(2,3) + L1(3,2)*M2(3,3);
v = 2*d*d;
c=1-v;
v_m=v/(1-d*d);
B = eye(4);
B11 = nLC23*nLC23*v_m + c;
B12 = -nLC23*nLC13*v_m;
B13 = -nLC13*2*d;
B21 = B12;
B22 = nLC13*nLC13*v_m + c;
```

```
B23 = -nLC23*2*d;
B31 = -B13;
B32 = -B23;
B33 = c;
B34 = A3;
B = [B11 B12 B13 0;
     B21 B22 B23 0;
     B31 B32 B33 B34;
     0   0   0   1];
ARefl3 = B ;
L2s = L1 * ARefl3;
L2 = L2s;
ok = 2

T4 = [1 0 0 xPsd;
      0 1 0 -yPsd;
      0 0 1 Zygo;
      0 0 0 1];
L3s = L2 * T4;
L3 = L3s;%subs(L4s);
ok = 3

Rs = L3;
pX = Rs(1,4);
pY = Rs(2,4);
pZ = Rs(3,4);

A = L3;
R = subs(A);
ok = 4

for k=1:1:134
    que1 = pque1(k,1);
    que2 = pque2(k,1);
    Zygo = pL(k,1);

    RR = subs(R);

    XYZ(1,k) = RR(1,4);
    XYZ(2,k) = RR(2,4);
    XYZ(3,k) = RR(3,4);
    ERR=[XYZ(1,k) - LeicaX(k,1);XYZ(2,k) - LeicaY(k,1);XYZ(3,k) - LeicaZ(k,1)];
    if j==1
        er_f=[ERR];
    else
        er_f=[er_f;ERR];
    end
    k
end
```

Appendix D

Alignment procedures and establishment of robot's tool co-ordinate frame

This appendix provides procedures for aligning the co-ordinate frame of the experimental LISM apparatus so that the apparatus is measuring position information with respect to the robot's tool co-ordinate frame.

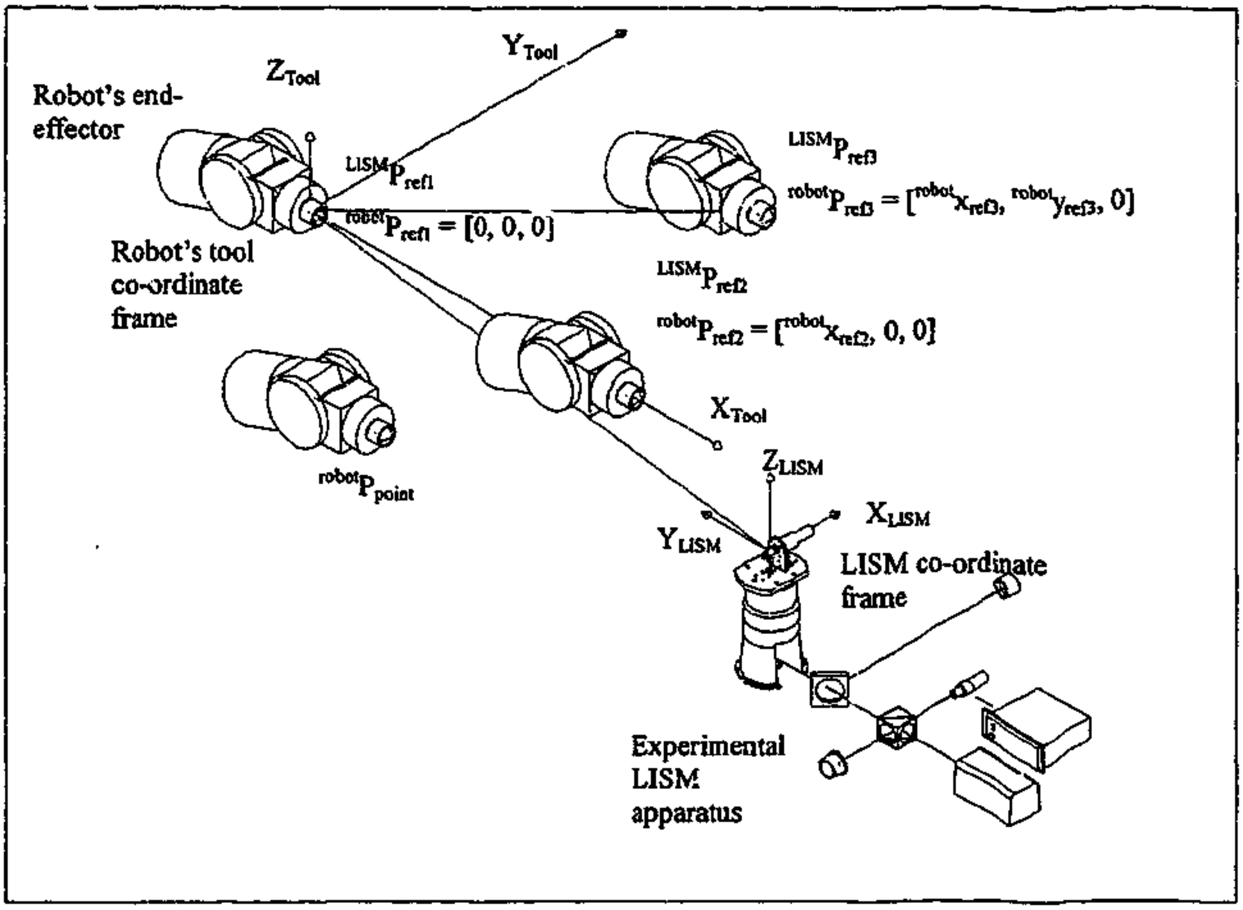

Y_{Tool}
Z_{Tool}
Robot's end-effector
$^{LISM}P_{ref1}$
$^{robot}P_{ref1} = [0, 0, 0]$
$^{LISM}P_{ref3}$
$^{robot}P_{ref3} = [^{robot}x_{ref3}, ^{robot}y_{ref3}, 0]$
Robot's tool co-ordinate frame
$^{LISM}P_{ref2}$
$^{robot}P_{ref2} = [^{robot}x_{ref2}, 0, 0]$
X_{Tool}
$^{robot}P_{point}$
Z_{LISM}
X_{LISM}
Y_{LISM}
LISM co-ordinate frame
Experimental LISM apparatus

Figure D.1: Robot positions utilised for the establishment of robot's tool co-ordinate frame

Alignment procedure

To establish the robot's tool co-ordinate frame, the following procedures are used:

1) Mount the LISM apparatus in a convenient position as shown in Fig. D.1. The position of the LISM apparatus must be chosen so that all the reference points can be pointed to by the laser beam from the LISM apparatus;

2) Place the target retroreflector at the birdbath. Start the tracking of the target retroreflector;

3) Place the retroreflector at the end-effector of the robot manipulator;

4) Jog the robot to move the retroreflector to the first reference point (${}^{LISM}P_{ref1}$). Click on the button STATIC SAMPLE as shown in Fig. D.2. A static point sampling dialog box (Fig. D.3) will appear. Enter the name for the current point and press OK. The current position of the reference point with respect to the LISM co-ordinate frame will be recorded. The first reference point is to become the origin of the robot's tool co-ordinate frame.

5) Repeat step 4 by jogging the robot in the robot's positive x direction to the second reference point (${}^{LISM}P_{ref2}$). This reference point is used to indicate the positive x-axis of the robot's tool co-ordinate frame.

6) Repeat step 4 by jogging the robot in the robot's positive x and positive y direction to the third reference point (${}^{LISM}P_{ref3}$). This reference point is used to indicate the positive xy-plane of the robot's tool co-ordinate frame.

7) All these reference points will be written in a file RefPtLISM.ini under the directory c:\winnt.

8) To align the axis, click on the TRANSFORMATION button. A dialog box as shown in Fig. D.4 will be presented to the user. Select the appropriate reference points in the combo box. Enter a name for the co-ordinate frame and press OK. The function Transformation() will be executed to compute the transformation matrix ${}^{LISM}A_{robot}$ used for the transformation of position information with respect to the LISM co-ordinate frame to the robot's tool co-ordinate frame. The determination of the matrix ${}^{LISM}A_{robot}$ is presented in the following section.

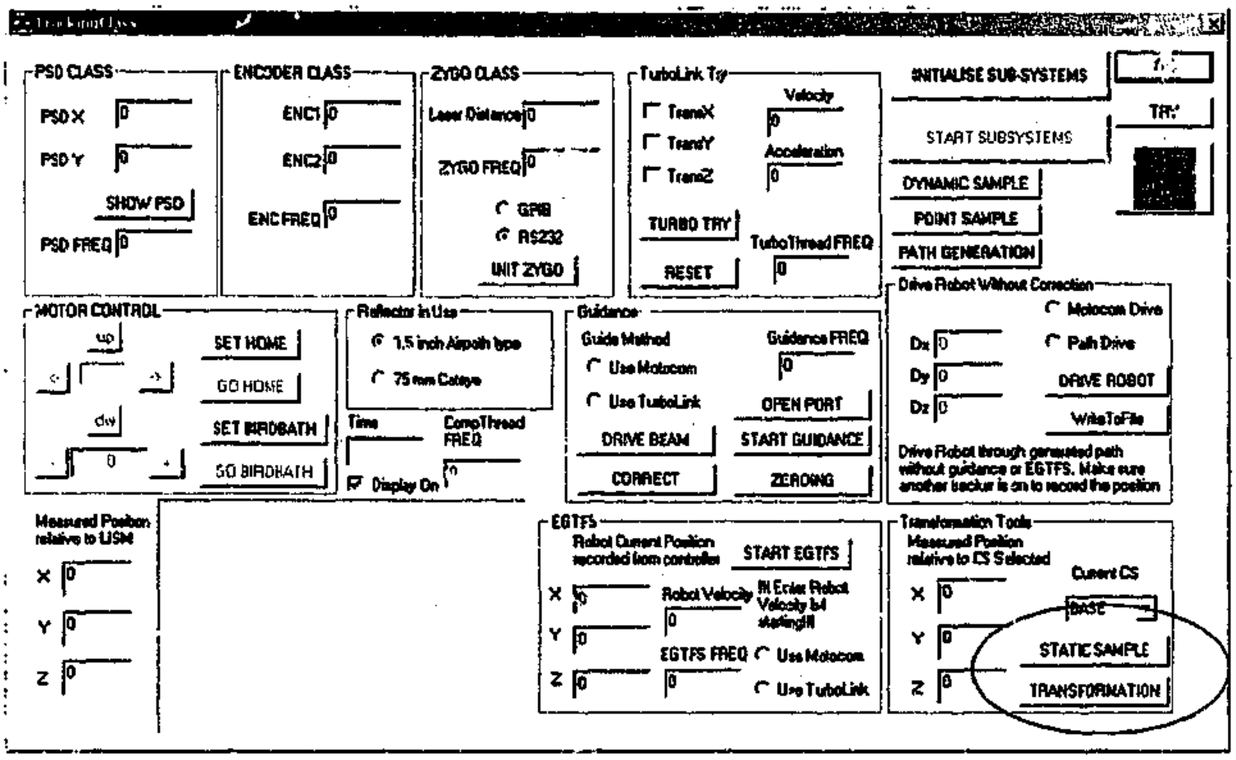
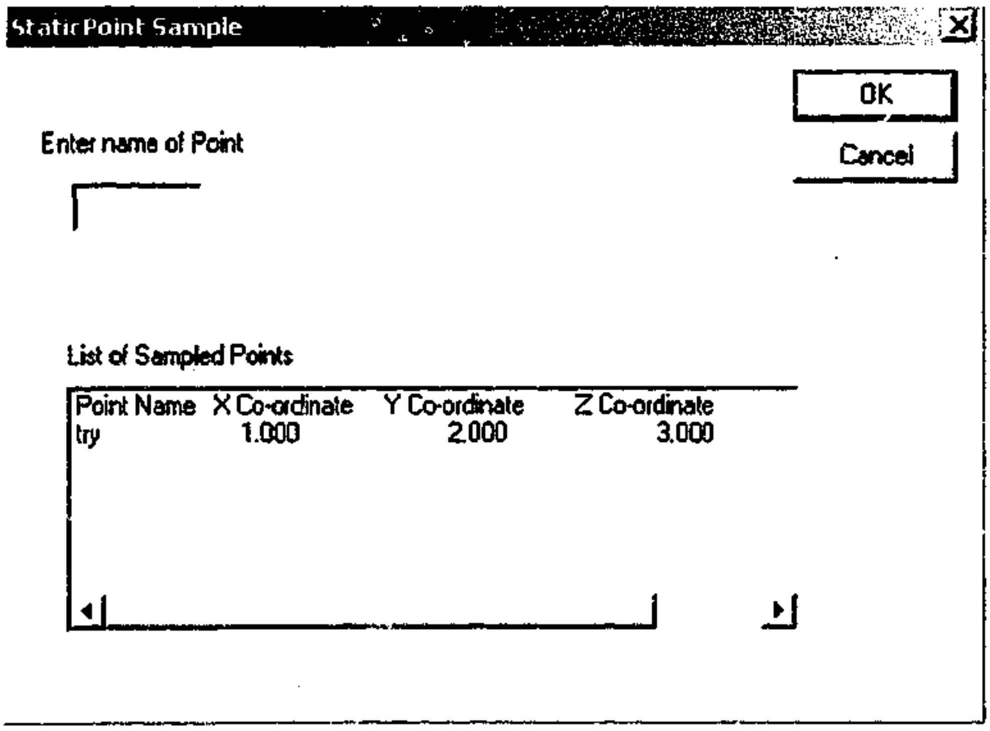
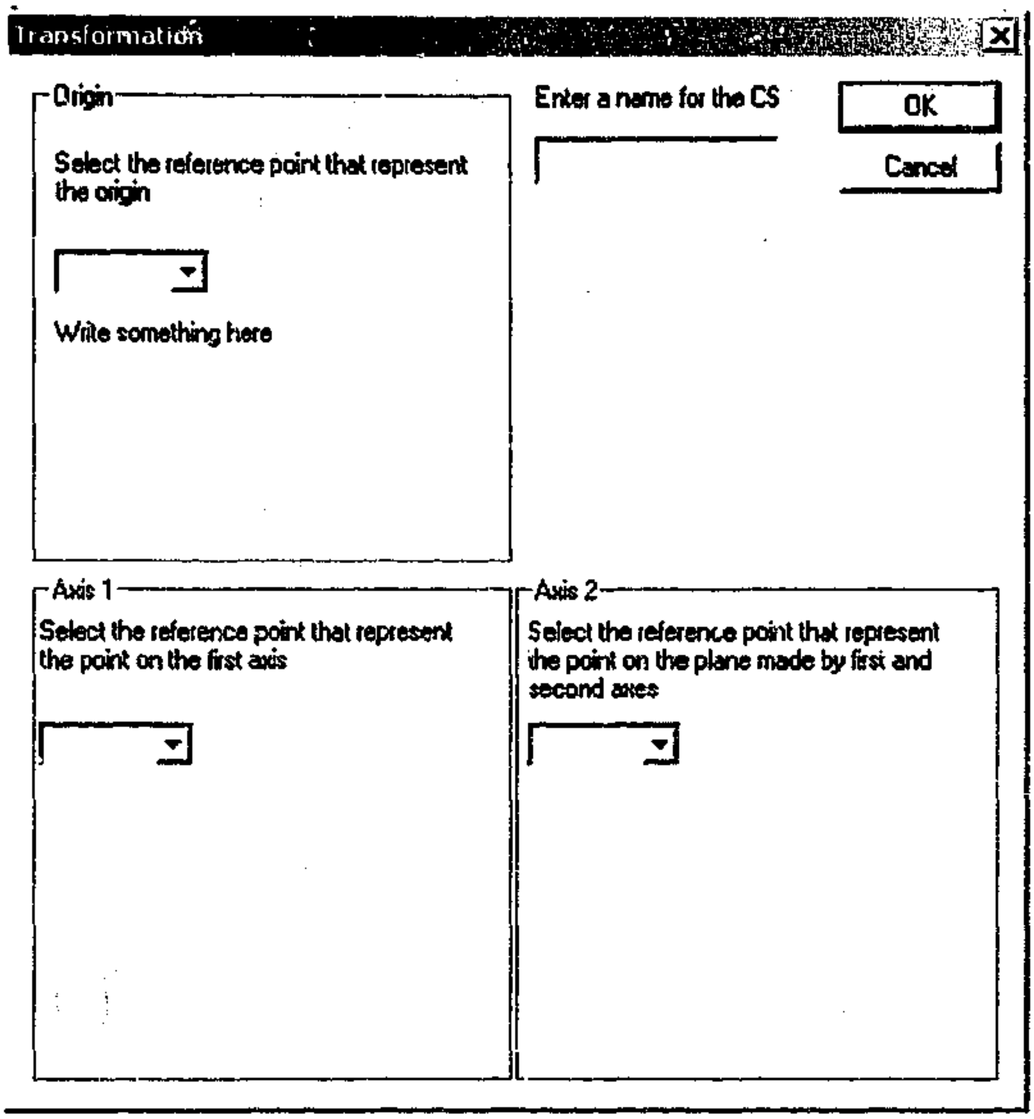


Figure D.2: LISM control program



```
Point Name  X Co-ordinate  Y Co-ordinate    Z Co-ordinate
try             1.000          2.000            3.000
```

Figure D.3: Static point sampling dialog box



Enter a name for the CS

Select the reference point that represent the origin

Write something here

Select the reference point that represent the point on the first axis

Select the reference point that represent the point on the plane made by first and second axes

Figure D.4: Transformation dialog box

Transformation matrix determination

${}^{LISM}A_{robot}$ is represented by the Z-Y-X Euler angle set [1] given by the following equation:

$$ {}^{LISM}A_{robot} = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma & {}^{LISM}x_{ref1} \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & {}^{LISM}y_{ref1} \\ -s\beta & c\alpha s\gamma & c\beta c\gamma & {}^{LISM}z_{ref1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(D.1)} $$

where cα = cos α and sα = sin α, etc., ${}^{LISM}x_{ref1}$, ${}^{LISM}y_{ref1}$, ${}^{LISM}z_{ref1}$ are the x, y and z positions of the reference point with respect to the LISM co-ordinate frame.

The aim is to determine the angles α, β and γ in the transformation matrix. This is done by first rewrite the position information of the references points with respect to the robot's tool co-ordinate frame as follows:

$$^{robot}P_{ref1}=\begin{bmatrix}0 & 0 & 0\end{bmatrix} \tag{D.2}$$

$$^{robot}P_{ref2}=\begin{bmatrix}^{robot}x_{ref2} & 0 & 0\end{bmatrix} \tag{D.3}$$

$$^{robot}P_{ref3}=\begin{bmatrix}^{robot}x_{ref3} & ^{robot}y_{ref3} & 0\end{bmatrix} \tag{D.4}$$

Following that, $^{LISM}P_{ref2}$ is rewritten as the following equation:

$$\begin{aligned}
&^{LISM}P_{ref2}={}^{LISM}A_{robot}\,{}^{robot}P_{ref2}\\
&\begin{bmatrix}^{LISM}x_{ref2}\\ ^{LISM}y_{ref2}\\ ^{LISM}z_{ref2}\\ 1\end{bmatrix}=\begin{bmatrix}c\alpha c\beta & c\alpha s\beta s\gamma-s\alpha c\gamma & c\alpha s\beta c\gamma+s\alpha s\gamma & ^{LISM}x_{ref1}\\ s\alpha c\beta & s\alpha s\beta s\gamma+c\alpha c\gamma & s\alpha s\beta c\gamma-c\alpha s\gamma & ^{LISM}y_{ref1}\\ -s\beta & c\alpha s\gamma & c\beta c\gamma & ^{LISM}z_{ref1}\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}^{robot}x_{ref2}\\ 0\\ 0\\ 1\end{bmatrix}\\
&\qquad=\begin{bmatrix}A\\ B\\ C\\ 1\end{bmatrix}
\end{aligned} \tag{D.5}$$

$$A=c\alpha c\beta\,{}^{robot}x_{ref2}+{}^{LISM}x_{ref1}={}^{LISM}x_{ref2} \tag{D.6}$$

$$B=s\alpha c\beta\,{}^{robot}x_{ref2}+{}^{LISM}y_{ref1}={}^{LISM}y_{ref2} \tag{D.7}$$

$$C=-s\beta\,{}^{robot}x_{ref2}+{}^{LISM}z_{ref1}={}^{LISM}z_{ref2} \tag{D.8}$$

From Eqs. D.6 and D.7, the equations can be rearranged as follows:

$$^{LISM}x_{ref2}-{}^{LISM}x_{ref1}=c\alpha c\beta\,{}^{robot}x_{ref2} \tag{D.9}$$

$$^{LISM}y_{ref2}-{}^{LISM}y_{ref1}=s\alpha c\beta\,{}^{robot}x_{ref2} \tag{D.10}$$

By dividing Eq. D.10 with Eq. D.9, the following equations can be obtained:

$$\tan\alpha=\frac{^{LISM}y_{ref2}-{}^{LISM}y_{ref1}}{^{LISM}x_{ref2}-{}^{LISM}x_{ref1}} \tag{D.11}$$

$$\alpha=\tan^{-1}\left(\frac{^{LISM}y_{ref2}-{}^{LISM}y_{ref1}}{^{LISM}x_{ref2}-{}^{LISM}x_{ref1}}\right) \tag{D.12}$$

From Eq. D.8, the equation can be rearranged as follows:

$$^{LISM}z_{ref2} - {}^{LISM}z_{ref1} = -s\beta\,{}^{robot}x_{ref2} \tag{D.13}$$

$$s\beta = -\frac{^{LISM}z_{ref2} - {}^{LISM}z_{ref1}}{^{robot}x_{ref2}} \tag{D.14}$$

$$\beta = \sin^{-1}\left(-\frac{^{LISM}z_{ref2} - {}^{LISM}z_{ref1}}{^{robot}x_{ref2}}\right) \tag{D.15}$$

${}^{robot}x_{ref2}$ can be determined by computing the length between ${}^{LISM}P_{ref1}$ and ${}^{LISM}P_{ref2}$ as shown in the following equation:

$$^{robot}x_{ref2} = \pm\sqrt{\left({}^{LISM}x_{ref1} - {}^{LISM}x_{ref2}\right)^2 + \left({}^{LISM}y_{ref1} - {}^{LISM}y_{ref2}\right)^2 + \left({}^{LISM}z_{ref1} - {}^{LISM}z_{ref2}\right)^2} \tag{D.16}$$

Only the positive solution is used as ${}^{robot}x_{ref2}$ indicates the positive x-axis of the robot's tool co-ordinate frame. The length between ${}^{LISM}P_{ref1}$ and ${}^{LISM}P_{ref2}$ cannot be zero (i.e. ${}^{LISM}P_{ref1} \neq {}^{LISM}P_{ref2}$) to prevent β from becoming infinity.

In order to find γ, ${}^{LISM}P_{ref3}$ is rewritten as:

$$^{LISM}P_{ref3} = {}^{LISM}A_{robot}\,{}^{robot}P_{ref3}$$

$$\begin{bmatrix} ^{LISM}x_{ref3} \\ ^{LISM}y_{ref3} \\ ^{LISM}z_{ref3} \\ 1 \end{bmatrix} = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma & ^{LISM}x_{ref1} \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & ^{LISM}y_{ref1} \\ -s\beta & c\alpha s\gamma & c\beta c\gamma & ^{LISM}z_{ref1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ^{robot}x_{ref3} \\ ^{robot}y_{ref3} \\ 0 \\ 1 \end{bmatrix} \tag{D.17}$$

$$= \begin{bmatrix} D \\ E \\ F \\ 1 \end{bmatrix}$$

$$D = c\alpha c\beta\,{}^{robot}x_{ref3} + (c\alpha s\beta s\gamma - s\alpha c\gamma)\,{}^{robot}y_{ref3} + {}^{LISM}x_{ref1} = {}^{LISM}x_{ref3} \tag{D.18}$$

$$E = s\alpha c\beta\,{}^{robot}x_{ref3} + (s\alpha s\beta s\gamma + c\alpha c\gamma)\,{}^{robot}y_{ref3} + {}^{LISM}y_{ref1} = {}^{LISM}y_{ref3} \tag{D.19}$$

$$F = -s\beta\,{}^{robot}x_{ref3} + c\beta s\gamma\,{}^{robot}y_{ref3} + {}^{LISM}z_{ref1} = {}^{LISM}z_{ref3} \tag{D.20}$$

Eq. D.20 can be rearrange to give the following equations:

$$s\gamma = \frac{{}^{LISM}z_{ref3} - {}^{LISM}z_{ref1} + s\beta\, {}^{robot}x_{ref3}}{c\beta\, {}^{robot}y_{ref3}} \tag{D.21}$$

$$\gamma = \sin^{-1}\left(\frac{{}^{LISM}z_{ref3} - {}^{LISM}z_{ref1} + s\beta\, {}^{robot}x_{ref3}}{c\beta\, {}^{robot}y_{ref3}} \right) \tag{D.22}$$

${}^{robot}x_{ref3}$ and ${}^{robot}y_{ref3}$ can be determined by reierring to Fig D.5. In this figure, the vector between ${}^{LISM}P_{ref1}$ and ${}^{LISM}P_{ref2}$ is represented by $\bar{l}_1$ and the vector between ${}^{LISM}P_{ref1}$ and ${}^{LISM}P_{ref3}$ is represented by $\bar{l}_2$. Angle ϑ can be found using the dot product between $\bar{l}_1$ and $\bar{l}_2$ as shown in the following equations:

$$\bar{l}_1 \cdot \bar{l}_2 = \left|\bar{l}_1\right|\left|\bar{l}_2\right| \cos\vartheta \tag{D.23}$$

$$\begin{aligned} &\begin{bmatrix} {}^{robot}x_{ref2} & {}^{robot}y_{ref2} & {}^{robot}z_{ref2} \end{bmatrix} \cdot \begin{bmatrix} {}^{robot}x_{ref3} & {}^{robot}y_{ref3} & {}^{robot}z_{ref3} \end{bmatrix} \\ &= \left|\bar{l}_1\right|\left|\bar{l}_2\right| \cos\vartheta \end{aligned} \tag{D.24}$$

$${}^{robot}x_{ref2}\, {}^{robot}x_{ref3} + {}^{robot}y_{ref2}\, {}^{robot}y_{ref3} + {}^{robot}z_{ref2}\, {}^{robot}z_{ref3} = \left|\bar{l}_1\right|\left|\bar{l}_2\right| \cos\vartheta \tag{D.25}$$

$$\left|\bar{l}_1\right| = \sqrt{\left({}^{LISM}x_{ref1} - {}^{LISM}x_{ref2}\right)^2 + \left({}^{LISM}y_{ref1} - {}^{LISM}y_{ref2}\right)^2 + \left({}^{LISM}z_{ref1} - {}^{LISM}z_{ref2}\right)^2} \tag{D.26}$$

$$\left|\bar{l}_2\right| = \sqrt{\left({}^{LISM}x_{ref1} - {}^{LISM}x_{ref3}\right)^2 + \left({}^{LISM}y_{ref1} - {}^{LISM}y_{ref3}\right)^2 + \left({}^{LISM}z_{ref1} - {}^{LISM}z_{ref3}\right)^2} \tag{D.27}$$

$$\vartheta = \cos^{-1}\left(\frac{{}^{robot}x_{ref2}\, {}^{robot}x_{ref3} + {}^{robot}y_{ref2}\, {}^{robot}y_{ref3} + {}^{robot}z_{ref2}\, {}^{robot}z_{ref3}}{\left|\bar{l}_1\right|\left|\bar{l}_2\right|} \right) \tag{D.28}$$

As a result, reference point 3 can be obtained using the following equations:

$${}^{robot}x_{ref3} = \left|\bar{l}_2\right| \cos\vartheta \tag{D.29}$$

$${}^{robot}y_{ref3} = \left|\bar{l}_2\right| \sin\vartheta \tag{D.30}$$

With ${}^{LISM}A_{robot}$, any other point recorded by the LISM apparatus can be transformed into position with respect to robot's tool co-ordinate frame by:

$${}^{robot}P_{point} = {}^{robot}A_{LISM}\, {}^{LISM}P_{point} \tag{D.31}$$

where ${}^{robot}A_{LISM} = {}^{LISM}A_{robot}{}^{-1}$.

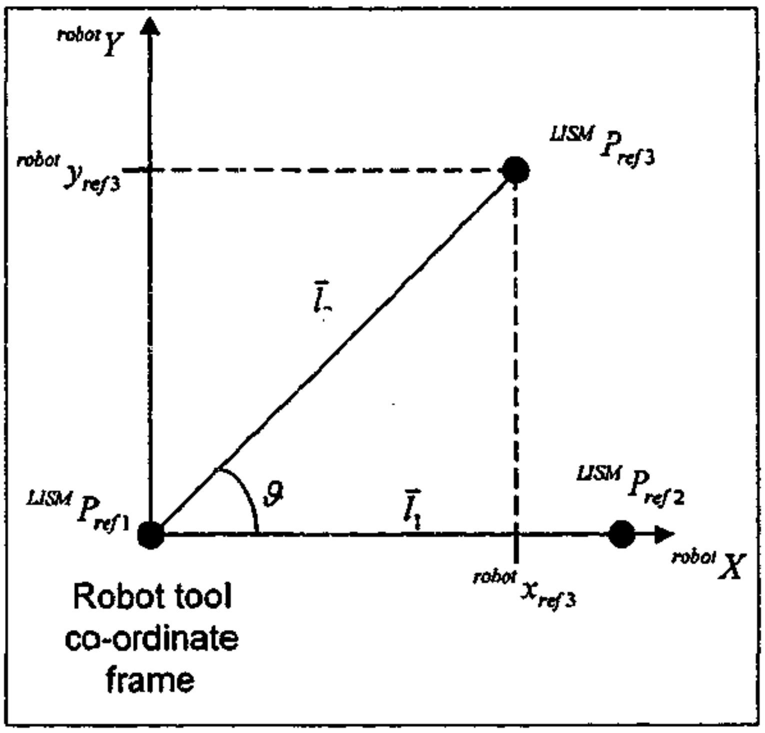


Figure D.5: Plane view of robot's reference positions

Appendix E

Control programs created for the tracking, target position measurement, robot closed-loop control and robot guidance

```
// TrackingClassDlg.cpp : implementation file
//

#include "stdafx.h"
#include "TrackingClass.h"
#include "TrackingClassDlg.h"
#include "acrolib.h"
#include "math.h"
#include "SSClass.h"
#include "StaticPt.h"
#include "Transformation.h"
#include "Path.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

int SelPathType;
int LocationPointer, OriginSelect, PosXSelect, PosYSelect;
int CurrLocationPointer;
bool bReplace;
int NumOfPt[4];
int CurrPt;
int NumOfLine;
int CurrLine;
unsigned long pACRAddress[1];
unsigned long pACRAddress2[1];
bool m_bAPRorCAT;
REC_DATA_CARTESIAN RefRelToLISMCartesian[3];
REC_DATA_CARTESIAN PtRelToLISMCartesian[Max_Point];
REC_DATA_CARTESIAN PtRelToCS[1];
REC_DATA_CARTESIAN CommandPos[1];                       //Current position of point in Cartesian
                                                        //co-ordinates
REC_DATA_CARTESIAN StartPoint[1];                       //Start position of path in Cartesian
                                                        //co-ordinates
REC_DATA_CARTESIAN LastPoint[1];                        //End position of path in Cartesian
                                                        //co-ordinates
REC_DATA_CARTESIAN PointList[Max_Line][Max_ViaPoint];    //Positions of path in Cartesian
                                                        //co-ordinates Max no. 128
double alpha, beta, gamma;
double dist12, dist13;
double TransMatrix[4][4];
double InvTMatrix[4][4];
CString CS[10];                                         //Max of 10 CS can be stored
int CS_Counter;                                         //Counter to store no of co-ordinates
                                                        //system set-up
double RobotCurX, RobotCurY, RobotCurZ;
double RobotDesX, RobotDesY, RobotDesZ;
double RobotX, RobotY, RobotZ;
double RobotMovXEGTFSTurbo, RobotMovYEGTFSTurbo, RobotMovZEGTFSTurbo;
double LISMDesX, LISMDesY, LISMDesZ;
float aveVel;
FILE* FileEgtfs;
FILE* FilePath;
FILE* FileGuidance;
FILE* FileRobotDr;
/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)

    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT


void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTrackingClassDlg dialog

CTrackingClassDlg::CTrackingClassDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CTrackingClassDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CTrackingClassDlg)
    m_sStatus = _T("");
    m_uiPSDFreq = 0;
    m_uiENCFreq = 0;
    m_uiZYGOFreq = 0;
    m_lEnc1 = 0;
    m_lEnc2 = 0;
    m_lfPSDX = 0.0;
    m_lfPSDY = 0.0;
    m_lfLaserDist = 0.0;
    m_uiStepSize = 0;
    m_sBeam = _T("");
    m_uiCompThreadFreq = 0;
    m_lfPosX = 0.0;
    m_lfPosY = 0.0;
    m_lfPosZ = 0.0;
    m_lfPosX2 = 0.0;
    m_lfPosY2 = 0.0;
    m_lfPosZ2 = 0.0;
    m_uiTurboFreq = 0;
    m_uiEGTFSFreq = 0;
    m_uiGuidanceFreq = 0;
    m_lfVel = 0.0;
    m_lfRobotX = 0.0;
    m_lfRobotY = 0.0;
    m_lfRobotZ = 0.0;
    m_lfTurboTryAcc = 0.0;
    m_lfTurboTryVel = 0.0;
    m_fDx = 0.0f;
    m_fDy = 0.0f;
    m_fDz = 0.0f;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CTrackingClassDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CTrackingClassDlg)
    DDX_Control(pDX, IDC_BUTTON_POINT_SAMPLE, m_ctrlPointSample);
    DDX_Control(pDX, IDC_RADIO_TURBO_CORR2, m_ctrlRadioTurbo2);
    DDX_Control(pDX, IDC_RADIO_MOTOCOM_CORR2, m_ctrlRadioMotocom2);
    DDX_Control(pDX, IDC_BUTTON_WRITE_FILE, m_ctrlWriteFile);
    DDX_Control(pDX, IDC_DZ, m_ctrlDz);
    DDX_Control(pDX, IDC_DY, m_ctrlDy);
    DDX_Control(pDX, IDC_DX, m_ctrlDx);
    DDX_Control(pDX, IDC_RADIO_PATH_DRIVE, m_ctrlRadioPathDrive);
    DDX_Control(pDX, IDC_RADIO_MOTOCOM_DRIVE, m_ctrlRadioMotocomDrive);
    DDX_Control(pDX, IDC_BUTTON_TURBOTRY, m_ctrlTurboTry);
    DDX_Control(pDX, IDC_RADIO_TURBO_CORR, m_ctrlRadioTurbo);
    DDX_Control(pDX, IDC_RADIO_MOTOCOM_CORR, m_ctrlRadioMotocom);
    DDX_Control(pDX, IDC_BUTTON_GUIDANCE, m_ctrlButtonGuidance);
    DDX_Control(pDX, IDC_BUTTON_EGTFS, m_ctrlButtonEGTFS);
    DDX_Control(pDX, IDC_COMBO_CS, m_ctrlComboCS);
    DDX_Control(pDX, IDC_BUTTON_DRIVE_ROBOT, m_ctrlDriveRobot);
    DDX_Control(pDX, IDC_BUTTON_OPEN_PORT, m_ctrlOpenPort);
    DDX_Control(pDX, IDC_BUTTON_DYNAMIC_SAMPLE, m_ctrlDynamicSample);
    DDX_Control(pDX, IDC_BUTTON_STATIC_SAMPLE, m_ctrlSample);
    DDX_Control(pDX, IDC_RADIO_CAT, m_ctrlRadioCAT);
    DDX_Control(pDX, IDC_RADIO_APR, m_ctrlRadioAPR);
    DDX_Control(pDX, IDC_BUTTON_GO_BB, m_ctrlGoBB);
    DDX_Control(pDX, IDC_BUTTON_TRACK, m_ctrlTrack);
    DDX_Control(pDX, IDC_CHECK_DISPLAY, m_ctrlCheckDisplay);
    DDX_Control(pDX, IDC_RADIO_RS232, m_ctrlRadioRS232);
    DDX_Control(pDX, IDC_RADIO_GPIB, m_ctrlRadioGPIB);
    DDX_Control(pDX, IDC_BUTTON_UP, m_ctrlUp);
    DDX_Control(pDX, IDC_BUTTON_INIT, m_ctrlInit);
    DDX_Control(pDX, IDC_EDIT_STEP_SIZE, m_ctrlStepSize);
    DDX_Control(pDX, IDC_BUTTON_SUBSYSTEM, m_ctrlSubSystem);
    DDX_Control(pDX, IDC_BUTTON_DEC_STEPSIZE, m_ctrlDecStepSize);
    DDX_Control(pDX, IDC_BUTTON_INC_STEPSIZE, m_ctrlIncStepSize);
    DDX_Control(pDX, IDC_BUTTON_RIGHT, m_ctrlRight);
    DDX_Control(pDX, IDC_BUTTON_LEFT, m_ctrlLeft);
    DDX_Control(pDX, IDC_BUTTON_DOWN, m_ctrlDown);
    DDX_Text(pDX, IDC_STATIC_STATUS, m_sStatus);
    DDX_Text(pDX, IDC_EDIT_PSD_FREQ, m_uiPSDFreq);
    DDX_Text(pDX, IDC_EDIT_ENC_FREQ, m_uiENCFreq);
    DDX_Text(pDX, IDC_EDIT_ZYGO_FREQ, m_uiZYGOFreq);
    DDX_Text(pDX, IDC_EDIT_ENC1, m_lEnc1);
    DDX_Text(pDX, IDC_EDIT_ENC2, m_lEnc2);
    DDX_Text(pDX, IDC_EDIT_PSDX, m_lfPSDX);
    DDX_Text(pDX, IDC_EDIT_PSDY, m_lfPSDY);
    DDX_Text(pDX, IDC_EDIT_ZYGO, m_lfLaserDist);
    DDX_Text(pDX, IDC_EDIT_STEP_SIZE, m_uiStepSize);
    DDX_Text(pDX, IDC_STATIC_BEAM, m_sBeam);
    DDX_Text(pDX, IDC_EDIT_COMPTHREAD_FREQ, m_uiCompThreadFreq);
    DDX_Text(pDX, IDC_POSX, m_lfPosX);
    DDX_Text(pDX, IDC_POSY, m_lfPosY);
    DDX_Text(pDX, IDC_POSZ, m_lfPosZ);
    DDX_Text(pDX, IDC_POSX2, m_lfPosX2);
    DDX_Text(pDX, IDC_POSY2, m_lfPosY2);
    DDX_Text(pDX, IDC_POSZ2, m_lfPosZ2);
    DDX_Text(pDX, IDC_EDIT_TURBO_FREQ, m_uiTurboFreq);
    DDX_Text(pDX, IDC_EDIT_EGTFS_FREQ, m_uiEGTFSFreq);
    DDX_Text(pDX, IDC_EDIT_GUIDANCE_FREQ, m_uiGuidanceFreq);
    DDX_Text(pDX, IDC_EDIT_VEL, m_lfVel);
    DDX_Text(pDX, IDC_EDIT_ROBOT_CUR_X, m_lfRobotX);
    DDX_Text(pDX, IDC_EDIT_ROBOT_CUR_Y, m_lfRobotY);
    DDX_Text(pDX, IDC_EDIT_ROBOT_CUR_Z, m_lfRobotZ);
    DDX_Text(pDX, IDC_EDIT_ACC2, m_lfTurboTryAcc);
    DDX_Text(pDX, IDC_EDIT_VEL2, m_lfTurboTryVel);
    DDX_Text(pDX, IDC_DX, m_fDx);
    DDX_Text(pDX, IDC_DY, m_fDy);
    DDX_Text(pDX, IDC_DZ, m_fDz);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTrackingClassDlg, CDialog)
    //{{AFX_MSG_MAP(CTrackingClassDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTON_INIT, OnButtonInit)
    ON_WM_TIMER()
    ON_BN_CLICKED(IDC_BUTTON_SUBSYSTEM, OnButtonSubsystem)
    ON_BN_CLICKED(IDC_BUTTON_UP, OnButtonUp)
    ON_BN_CLICKED(IDC_BUTTON_DOWN, OnButtonDown)
    ON_BN_CLICKED(IDC_BUTTON_LEFT, OnButtonLeft)
    ON_BN_CLICKED(IDC_BUTTON_RIGHT, OnButtonRight)
    ON_EN_CHANGE(IDC_EDIT_STEP_SIZE, OnChangeEditStepSize)
    ON_BN_CLICKED(IDC_BUTTON_ReINIT_ZYGO, OnBUTTONReINITZYGO)
    ON_BN_CLICKED(IDC_RADIO_GPIB, OnRadioGpib)
    ON_BN_CLICKED(IDC_RADIO_RS232, OnRadioRs232)
    ON_BN_CLICKED(IDC_BUTTON_TRACK, OnButtonTrack)
    ON_BN_CLICKED(IDC_BUTTON_SET_BB, OnButtonSetBb)
    ON_BN_CLICKED(IDC_BUTTON_GO_BB, OnButtonGoBb)
    ON_BN_CLICKED(IDC_RADIO_APR, OnRadioApr)
    ON_BN_CLICKED(IDC_RADIO_CAT, OnRadioCat)
    ON_BN_CLICKED(IDC_BUTTON_STATIC_SAMPLE, OnButtonStaticSample)
    ON_BN_CLICKED(IDC_BUTTON_DYNAMIC_SAMPLE, OnButtonDynamicSample)
    ON_BN_CLICKED(IDC_BUTTON_OPEN_PORT, OnButtonOpenPort)
    ON_BN_CLICKED(IDC_BUTTON_DRIVE_ROBOT, OnButtonDriveRobot)
    ON_BN_CLICKED(IDC_CHECK_TX, OnCheckTx)
    ON_BN_CLICKED(IDC_CHECK_TY, OnCheckTy)
    ON_BN_CLICKED(IDC_CHECK_TZ, OnCheckTz)
    ON_BN_CLICKED(IDC_BUTTON_RESET, OnButtonReset)
    ON_EN_CHANGE(IDC_EDIT_ACC, OnChangeEditAcc)
    ON_BN_CLICKED(IDC_BUTTON_TRANSFORMATION, OnButtonTransformation)
    ON_BN_CLICKED(IDC_BUTTON_PATH_GENERATION, OnButtonPathGeneration)
    ON_BN_CLICKED(IDC_BUTTON_EGTFS, OnButtonEgtfs)
    ON_BN_CLICKED(IDC_BUTTON_GUIDANCE, OnButtonGuidance)
    ON_BN_CLICKED(IDC_RADIO_TURBO_CORR, OnRadioTurboCorr)
    ON_BN_CLICKED(IDC_RADIO_MOTOCOM_CORR, OnRadioMotocomCorr)
    ON_BN_CLICKED(IDC_BUTTON_TURBOTRY, OnButtonTurbotry)
    ON_BN_CLICKED(IDC_RADIO_MOTOCOM_DRIVE, OnRadioMotocomDrive)
    ON_BN_CLICKED(IDC_RADIO_PATH_DRIVE, OnRadioPathDrive)
    ON_EN_CHANGE(IDC_DX, OnChangeDx)
    ON_EN_CHANGE(IDC_DY, OnChangeDy)
    ON_EN_CHANGE(IDC_DZ, OnChangeDz)
    ON_CBN_SELCHANGE(IDC_COMBO_CS, OnSelchangeComboCs)
    ON_BN_CLICKED(IDC_BUTTON_WRITE_FILE, OnButtonWriteFile)
    ON_EN_CHANGE(IDC_EDIT_VEL, OnChangeEditVel)
    ON_BN_CLICKED(IDC_BUTTON_DRIVE_BEAM, OnButtonDriveBeam)
    ON_BN_CLICKED(IDC_BUTTON_CORRECT, OnButtonCorrect)
    ON_BN_CLICKED(IDC_BUTTON_ZERO, OnButtonZero)
    ON_BN_CLICKED(IDC_BUTTON_TRY, OnButtonTry)
    ON_EN_CHANGE(IDC_EDIT_ACC2, OnChangeEditAcc2)
    ON_BN_CLICKED(IDC_BUTTON_POINT_SAMPLE, OnButtonPointSample)
    ON_BN_CLICKED(IDC_RADIO_MOTOCOM_CORR2, OnRadioMotocomCorr2)
    ON_BN_CLICKED(IDC_RADIO_TURBO_CORR2, OnRadioTurboCorr2)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTrackingClassDlg message handlers

BOOL CTrackingClassDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);          // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

    // TODO: Add extra initialization here
    hImageStart.LoadBitmap(IDB_START);
    hImageStop.LoadBitmap(IDB_STOP);
    m_ctrlTrack.SetBitmap(hImageStart);

    m_ctrlCheckDisplay.SetCheck(1);

    m_bTrack = false;
    m_bInitFlag = false;
    m_bSubSystemRun = false;
    m_bDataThread = false;
    m_bFileFlag = false;
    m_bSample = false;
    m_bDynamicSample = false;

    m_bEncoderThread = false;
    m_bPsdThread = false;
    m_bZygoThread = false;

    m_bGPIBorRS232 = false;
    m_ctrlRadioGPIB.SetCheck(0);
    m_ctrlRadioRS232.SetCheck(1);

    m_ctrlRadioAPR.SetCheck(1);
    m_ctrlRadioCAT.SetCheck(0);
    m_bAPRorCAT = true;

    m_lfLaserDist = 0.0;
    m_lfPSDX = 0.0;
    m_lfPSDY = 0.0;
    m_lEnc1 = 0;
    m_lEnc2 = 0;

    m_bUpdateTurbo = false;
    m_bTurboTryThread = false;
    m_bDriveRobot = false;
    m_bDriveRobotWrite = false;

    m_bTx = false;
    m_bTy = false;
    m_bTz = false;

    bReplace = false;
    char temp[256];
    char buffer[128];

    GetPrivateProfileString("No", "Total Pt Num", NullString, temp, (int)sizeof(temp), "RefPtLI
SM.ini");
    int no = atoi(temp);
    UpdateData(false);
    for (LocationPointer=0 ; LocationPointer < no ; LocationPointer++)
    {
        sprintf(buffer, "RefPoint%d", LocationPointer+1);
        GetPrivateProfileString(buffer, "Name", NullString, temp, (int)sizeof(temp), "RefPtLISM
.ini");
        PtRelToLISMCartesian[LocationPointer].name = temp;
        GetPrivateProfileString(buffer, "X", NullString, temp, (int)sizeof(temp), "RefPtLISM.in
i");
        PtRelToLISMCartesian[LocationPointer].x = atof(temp);
        GetPrivateProfileString(buffer, "Y", NullString, temp, (int)sizeof(temp), "RefPtLISM.in
i");
        PtRelToLISMCartesian[LocationPointer].y = atof(temp);
        GetPrivateProfileString(buffer, "Z", NullString, temp, (int)sizeof(temp), "RefPtLISM.in
i");
        PtRelToLISMCartesian[LocationPointer].z = atof(temp);
    }

    m_bGuide = false;
    m_bEGTFS = false;
    m_bEGTFSTurbo = false;
    m_bPathDrive = false;
    m_bMotocomDrive = false;
    m_bTurboCorrThread = false;
    for (int j=0; j<3; j++)
    {
        for (int k=0; k<3; k++)
        {
            if (j==k)
            {
                TransMatrix[j][k] = 1.0;
                InvTMatrix[j][k] = 1.0;
            }
            else
            {
                TransMatrix[j][k] = 0.0;
                InvTMatrix[j][k] = 0.0;
            }
        }
    }

    LocationPointer = 0;
    m_ctrlComboCS.AddString("BASE");
    m_ctrlComboCS.SetCurSel(0);
    CS[0] = "BASE";
    CS_Counter = 1;

    CString empty;
    GetPrivateProfileString("CSName", "Name", NullString, temp, (int)sizeof(temp), "TransMatrix
ISM.ini");
    empty = temp;
    if (empty !="")
    {
        CS[1] = temp;
        m_ctrlComboCS.AddString("ROBOT");
        //CS_Counter = 2;
    }

    MyRobot = new CRobotController(MRC);
    CurrPt = 0;
    CurrLine = 0;
    RobotX = 0.0;
    RobotY = 0.0;
    RobotZ = 0.0;
    RobotMovXEGTFSTurbo = 0.0;
    RobotMovYEGTFSTurbo = 0.0;
    RobotMovZEGTFSTurbo = 0.0;
    aveVel = 0.0;

    return TRUE;  // return TRUE  unless you set the focus to a control


void CTrackingClassDlg::OnSysCommand(UINT nID, LPARAM lParam)

    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }


// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CTrackingClassDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CTrackingClassDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CTrackingClassDlg::OnButtonInit()
```

```
    // TODO: Add your control notification handler code here
    CString StatBuf;

    m_sStatus = "Creating Log File...";
    UpdateData(false);
    if ((fp = fopen("tracking.txt", "w")) == NULL)
    {
        puts("cannot open file");
        exit(1);
    }
    rewind(fp);          // set the cursor to the beginning of file
    fprintf(fp, "      PSD X     PSD Y   LASER DIST  M1CurrAngle M2CurrAngle \n");
    m_sStatus += "Created\n";
    m_bFileFlag = true;
    UpdateData(false);

    m_sStatus = "Initialising all sub-systems....\n";
    UpdateData(false);

    //Initialise Motor Controller
    m_sStatus += "Initialising Motor Controller...\n";
    UpdateData(false);
    Encoder.InitialiseMotor(&StatBuf, pACRAddress, pACRAddress2);
    m_sStatus += StatBuf;
    UpdateData(false);

    //Initialise ZYGO Laser Interferometer
    m_sStatus += "Communicating with Zygo Laser...\n";
    UpdateData(false);
    Zygo.InitialiseLaser(m_bGPIBorRS232, &StatBuf);
    m_sStatus += StatBuf;
    UpdateData(false);

    //Initialise DT303
    m_sStatus += "Initialising DT303...\n";
    UpdateData(false);
    Psd.InitialisePSDSv(&StatBuf);
    m_sStatus += StatBuf;
    UpdateData(false);

    //Start Display Timer
    m_sStatus += "Starting Timer for Display...\n";
    UpdateData(false);
    SetTimer(0x10, 1000, NULL);

    //ReadMotorStatusThread. Start thread
    m_sStatus += "Starting ReadMotorStatusThread...\n";
    UpdateData(false);

    m_bEncoderThread = true;
    Encoder.StartThread();

    m_bDataThread = true;
    THREADPARAMS* pDataThread = new THREADPARAMS;
    pDataThread->lParam = (LPARAM) this;
    g_pComputationThread = AfxBeginThread(ComputationThread, pDataThread, THREAD_PRIORITY_NORMA
L, NULL);
    //SetTimer(0x11, 10, NULL);

    m_bInitFlag = true;

    m_ctrlUp.EnableWindow(1);
    m_ctrlDown.EnableWindow(1);
    m_ctrlLeft.EnableWindow(1);
    m_ctrlRight.EnableWindow(1);
    m_ctrlIncStepSize.EnableWindow(1);
    m_ctrlDecStepSize.EnableWindow(1);
    m_ctrlSubSystem.EnableWindow(1);
    m_ctrlStepSize.EnableWindow(1);
    m_ctrlInit.EnableWindow(0);
    m_sStatus += "IDLE";

    UpdateData(false);
}

void CTrackingClassDlg::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    //double PSDx, PSDy, LaserDist;
    //long pPosParameter[2];
    //CMotor M1, M2;
    //double CosTheta1, CosTheta2, SinTheta1, SinTheta2;
    //SYSTEMTIME st;
    //double AngleDiff, CorrDist1, CorrDist2, M1Diff, M2Diff;
    //char cmdline[256];
    //long pReturn[3];
    //float XYZ[3];

    if (nIDEvent == 0x10)
    {
        m_uiZYGOFreq = Zygo.GetFreq();
        m_uiPSDFreq = Psd.GetFreq();
        m_uiENCFreq = Encoder.GetFreq();

        /*A8_BIN_PEEK_LONG(0x00, 1, pACRAddress2[0]+1, pReturn, 0);
        A8_BIN_PEEK_LONG(0x00, 1, pACRAddress2[0]+2, pReturn+1, 0);
        A8_BIN_PEEK_LONG(0x00, 1, pACRAddress2[0]+3, pReturn+2, 0);
        A8_BIN_PEEK_IEEE(0x01, 1, pReturn[0]+1, XYZ, 0);
        A8_BIN_PEEK_IEEE(0x01, 1, pReturn[1]+1, XYZ+1, 0);
        A8_BIN_PEEK_IEEE(0x01, 1, pReturn[2]+1, XYZ+2, 0);
        m_lfPosX2 = XYZ[0];
        m_lfPosY2 = XYZ[1];
        m_lfPosZ2 = XYZ[2];*/
        UpdateData(false);
        m_uiTurboFreq = 0;
        m_uiCompThreadFreq = 0;
        m_uiEGTFSFreq = 0;
        m_uiGuidanceFreq = 0;
    }
    CDialog::OnTimer(nIDEvent);

void CTrackingClassDlg::OnButtonSubsystem()

    // TODO: Add your control notification handler code here
    if (!m_bSubSystemRun)
    {
        m_bSubSystemRun = true;
        m_ctrlSubSystem.SetWindowText("STOP SUBSYSTEMS");

        m_ctrlTrack.EnableWindow(1);

        Psd.StartThread();
        m_bPsdThread = true;

        Zygo.StartThread();
        m_bZygoThread = true;

        if (!m_bEncoderThread)
        {
            Encoder.StartThread();
            m_bEncoderThread = true;
        }
        /*AcroSendString("DA0(9) = 0", 0);
        AcroSendString("DA1(9) = 0", 0);
        AcroSendString("DA2(9) = 0", 0);
        AcroSendString("DA0(8) = 0", 0);
        AcroSendString("DA1(8) = 0", 0);
        AcroSendString("DA2(8) = 0", 0);
        AcroSendString("DA0(7) = 0", 0);
        AcroSendString("DA1(7) = 0", 0);
        AcroSendString("DA2(7) = 0", 0);
        AcroSendString("DA0(6) = 0", 0);
        AcroSendString("DA1(6) = 0", 0);
        AcroSendString("DA2(6) = 0", 0);
        AcroSendString("DA0(5) = 0", 0);
        AcroSendString("DA1(5) = 0", 0);
        AcroSendString("DA2(5) = 0", 0);
        AcroSendString("DA0(4) = 0", 0);
        AcroSendString("DA1(4) = 0", 0);
        AcroSendString("DA2(4) = 0", 0);
        AcroSendString("DA0(3) = 0", 0);
        AcroSendString("DA1(3) = 0", 0);
        AcroSendString("DA2(3) = 0", 0);
        AcroSendString("DA0(2) = 0", 0);
        AcroSendString("DA1(2) = 0", 0);
        AcroSendString("DA2(2) = 0", 0);
        AcroSendString("DA0(1) = 0", 0);
        AcroSendString("DA1(1) = 0", 0);
        AcroSendString("DA2(1) = 0", 0);
        AcroSendString("DA0(0) = 0", 0);
        AcroSendString("DA1(0) = 0", 0);
        AcroSendString("DA2(0) = 0", 0);
        AcroSendString("LV4 = 0", 0);*/
        /*AcroSendString("DA3(0) = (DA0(0) - DA0(1)) / 0.03
        AcroSendString("DA3(1) = (DA1(0) - DA1(1)) / 0.03
        AcroSendString("DA3(2) = (DA2(0) - DA2(1)) / 0.03
        AcroSendString("DA4(0) = (DA0(0) - 2 * DA0(1) + DA0(2)) / (0.03 * 0.03)
        AcroSendString("DA4(1) = (DA1(0) - 2 * DA1(1) + DA1(2)) / (0.03 * 0.03)
        AcroSendString("DA4(2) = (DA2(0) - 2 * DA2(1) + DA2(2)) / (0.03 * 0.03)
        AcroSendString("DA5(0) = DA3(0) * 0.03 + 0.5 * DA4(0) * 0.03 * 0.03 + DA0(0)
        AcroSendString("DA5(1) = DA3(1) * 0.03 + 0.5 * DA4(1) * 0.03 * 0.03 + DA1(0)
        AcroSendString("DA5(2) = DA3(2) * 0.03 + 0.5 * DA4(2) * 0.03 * 0.03 + DA2(0)
/
    }
    else
    {
        m_bSubSystemRun = false;
        m_ctrlSubSystem.SetWindowText("START SUBSYSTEMS");
        //fclose(fp);
        //m_bFileFlag = false;

        Psd.StopThread();
        m_bPsdThread = false;

        Zygo.StopThread();
        m_bZygoThread = false;

        Encoder.StopThread();
        m_bEncoderThread = false;
    }


void CTrackingClassDlg::OnButtonUp()

    // TODO: Add your control notification handler code here
    m_sBeam = "UP";
    UpdateData(false);
    Motor1.DriveAngle = 0.0;
    Motor2.DriveAngle = double(m_uiStepSize) / 80000.0;
    DriveMotor(Motor1, Motor2);


void CTrackingClassDlg::OnButtonDown()
{
    // TODO: Add your control notification handler code here
    m_sBeam = "DOWN";
    UpdateData(false);
    Motor1.DriveAngle = 0.0;
    Motor2.DriveAngle = -1 * double(m_uiStepSize) / 80000.0;
    DriveMotor(Motor1, Motor2);
}

void CTrackingClassDlg::OnButtonLeft()
{
    // TODO: Add your control notification handler code here
    m_sBeam = "LEFT";
    UpdateData(false);
    Motor1.DriveAngle = -1 * double(m_uiStepSize) / 614400.0;
    Motor2.DriveAngle = 0.0;
    DriveMotor(Motor1, Motor2);
}

void CTrackingClassDlg::OnButtonRight()
{
    // TODO: Add your control notification handler code here
    m_sBeam = "RIGHT";
    UpdateData(false);
    Motor1.DriveAngle = double(m_uiStepSize) / 614400.0;
    Motor2.DriveAngle = 0.0;
    DriveMotor(Motor1, Motor2);
}

void CTrackingClassDlg::DriveMotor(CMotor MotorData1, CMotor MotorData2)
{
    char cmdline[256];

    //if (fabs(MotorData1.DriveAngle) < 0.5)
    //{
        MotorData2.DriveAngle = -1 * MotorData2.DriveAngle;
        //Increment mode used in jogging
        sprintf(cmdline, "JOG INC X%lf Y%lf", MotorData1.DriveAngle, MotorData2.DriveAngle);
        AcroSendString(cmdline, 0);
        return;
    //}
    //else
    //{
    //  MessageBeep(0);
    //  AfxMessageBox("DANGER!! MOTOR 1 trajectory is out of range", MB_ICONEXCLAMATION);
    //}


oid CTrackingClassDlg::OnChangeEditStepSize()

    // TODO: If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialog::OnInitDialog()
    // function and call CRichEditCtrl().SetEventMask()
    // with the ENM_CHANGE flag ORed into the mask.

    // TODO: Add your control notification handler code here
    UpdateData(true);


oid CTrackingClassDlg::OnBUTTONReINITZYGO()

    // TODO: Add your control notification handler code here
    CString StatBuf;
    Zygo.InitialiseLaser(m_bGPIBorRS232, &StatBuf);
    m_sStatus = StatBuf;
    UpdateData(false);


INT CTrackingClassDlg::ComputationThread(LPVOID pParam)

    THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
    CTrackingClassDlg* pWnd = (CTrackingClassDlg*) pThreadParams->lParam;
    delete pThreadParams;
    double PSDx, PSDy, LaserDist;
    double SumPSDx, SumPSDy, SumLaserDist, SumAngle1, SumAngle2, SumOffsetX, SumOffsetY;
    double AvePSDx, AvePSDy, AveLaserDist, AveAngle1, AveAngle2, AveOffsetX, AveOffsetY;
    //long pReturn[3];
    //float XYZ[3];

    double CosTheta1, CosTheta2, SinTheta1, SinTheta2, TanTheta1, TanTheta2;
    SYSTEMTIME st;
    //double AngleDiff, CorrDist1, CorrDist2, M1Diff, M2Diff;
    //char cmdline[256];
    int NoOfPoints;
    double PreviousX, PreviousY, PreviousZ;
    double VelX, VelY, VelZ;
    double PrevVX, PrevVY, PrevVZ;
    double AccX, AccY, AccZ;
    long PreviousTime;
    double TimeDiff;
    double OffsetX, OffsetY, L;

    NoOfPoints = 0;
    SumPSDx = 0.0;
    SumPSDy = 0.0;
    SumOffsetX = 0.0;
    SumOffsetY = 0.0;
    SumLaserDist = 0.0;
    SumAngle1 = 0.0;
    SumAngle2 = 0.0;
    PreviousX = 0.0;
    PreviousY = 0.0;
    PreviousZ = 0.0;
    PreviousTime = 0;
    PrevVX = 0.0;
    PrevVY = 0.0;
    PrevVZ = 0.0;

    while (pWnd->m_bDataThread)
    {
        pWnd->Psd.GetSample(&PSDx, &PSDy);
        pWnd->Encoder.GetSample(&pWnd->M1, &pWnd->M2);
        LaserDist = pWnd->Zygo.GetSample();
        pWnd->m_lfLaserDist = LaserDist;
        pWnd->m_lfPSDX = PSDx;
        pWnd->m_lfPSDY = PSDy;
        pWnd->m_lEnc1 = pWnd->M1.enc;
        pWnd->m_lEnc2 = pWnd->M2.enc;

        /*A8_BIN_PEEK_LONG(0x00, 1, pACRAddress2[0]+1, pReturn, 0);
        A8_BIN_PEEK_LONG(0x00, 1, pACRAddress2[0]+2, pReturn+1, 0);
        A8_BIN_PEEK_LONG(0x00, 1, pACRAddress2[0]+3, pReturn+2, 0);
        A8_BIN_PEEK_IEEE(0x01, 1, pReturn[0]+1, XYZ, 0);
        A8_BIN_PEEK_IEEE(0x01, 1, pReturn[1]+1, XYZ+1, 0);
        A8_BIN_PEEK_IEEE(0x01, 1, pReturn[2]+1, XYZ+2, 0);
        pWnd->m_lfPosX = XYZ[0];
        pWnd->m_lfPosY = XYZ[1];
        pWnd->m_lfPosZ = XYZ[2];
/
        CosTheta1 = cos(pWnd->M1.CurrentAngle * 2 * pi);
        SinTheta1 = sin(pWnd->M1.CurrentAngle * 2 * pi);
        CosTheta2 = cos(pWnd->M2.CurrentAngle * 4 * pi);
        SinTheta2 = sin(pWnd->M2.CurrentAngle * 4 * pi);
        TanTheta1 = SinTheta1/CosTheta1;
        TanTheta2 = SinTheta2/CosTheta2;
        OffsetX = -0.5 * (PSDx * CosTheta1 + PSDy * SinTheta1);
        OffsetY = 0.5 * (PSDx * -1 * SinTheta1 + PSDy * CosTheta1);
        L = LaserDist + OffsetX * TanTheta1 - OffsetY * TanTheta2;

        pWnd->m_lfPosX = -1.0 * L * CosTheta2 * SinTheta1 - OffsetX / CosTheta1;//-1.0 *  Laser
ist * CosTheta2 * SinTheta1;
        pWnd->m_lfPosY = L * CosTheta2 * CosTheta1;//LaserDist * CosTheta2 * CosTheta1;
        pWnd->m_lfPosZ = L * SinTheta2 + OffsetY / CosTheta2; //LaserDist * SinTheta2;

        ::GetSystemTime(&st);
        pWnd->m_sTime.Format("%d", (st.wMinute * 60000 + st.wSecond * 1000 + st.wMilliseconds))

        TimeDiff = atof(pWnd->m_sTime) - PreviousTime; //millisecond
        PreviousTime = atol(pWnd->m_sTime);
        VelX = (pWnd->m_lfPosX - PreviousX) / TimeDiff; //m/s
        VelY = (pWnd->m_lfPosY - PreviousY) / TimeDiff;
        VelZ = (pWnd->m_lfPosZ - PreviousZ) / TimeDiff;
        AccX = (VelX - PrevVX) / TimeDiff;
        AccY = (VelY - PrevVY) / TimeDiff;
        AccZ = (VelZ - PrevVZ) / TimeDiff;

        PreviousX = pWnd->m_lfPosX;
        PreviousY = pWnd->m_lfPosY;
        PreviousZ = pWnd->m_lfPosZ;
        PrevVX = VelX;
        PrevVY = VelY;
        PrevVZ = VelZ;

        if (pWnd->m_bDynamicSample)
        {
            ::fprintf(pWnd->fp, "%s \t%lf\t %lf\t %lf\t%lf\t %lf\t %lf\t %lf\t %lf\t %lf\t\n",
                            pWnd->m_sTime, PSDx, PSDy, LaserDist, VelX, VelY, VelZ, AccX, AccY, Acc
);
        }
        if (pWnd->m_bSample)
        {
            if (NoOfPoints < 100)
            {
                SumPSDx += PSDx;
                SumPSDy += PSDy;
                SumOffsetX += OffsetX;
                SumOffsetY += OffsetY;
                SumLaserDist += LaserDist;
                SumAngle1 += pWnd->M1.CurrentAngle*360.0;
                SumAngle2 += pWnd->M2.CurrentAngle*720.0;
                NoOfPoints++;
            }
            else
            {
                AvePSDx = SumPSDx / (NoOfPoints);
                AvePSDy = SumPSDy / (NoOfPoints);
                AveLaserDist = SumLaserDist/ (NoOfPoints);
                AveAngle1 = SumAngle1 / (NoOfPoints);
                AveAngle2 = SumAngle2 / (NoOfPoints);
                AveOffsetX = SumOffsetX / (NoOfPoints);
                AveOffsetY = SumOffsetY / (NoOfPoints);
                ::GetSystemTime(&st);
                pWnd->m_sTime.Format("%d", (st.wMinute * 60000 + st.wSecond * 1000 + st.wMillis
conds));
                ::fprintf(pWnd->fp, "%d \t%s \t%.3lf\t %.3lf\t %.3lf\t %.3lf\t %.3lf\t %.3lf\t
.3lf\t \n",
                        NoOfPoints, pWnd->m_sTime, AvePSDx, AvePSDy, AveOffsetX, AveOffsetY, Av
LaserDist, AveAngle1, AveAngle2);
                NoOfPoints = 0;
                SumPSDx = 0.0;
                SumPSDy = 0.0;
                SumLaserDist = 0.0;
                SumAngle1 = 0.0;
                SumAngle2 = 0.0;
                SumOffsetX = 0.0;
                SumOffsetY = 0.0;
                pWnd->OnButtonPointSample();
            }
        }

        if (pWnd->m_ctrlComboCS.GetCurSel())
        {
            PtRelToCS[0].x = TransMatrix[0][0] * pWnd->m_lfPosX
                           + TransMatrix[0][1] * pWnd->m_lfPosY
                           + TransMatrix[0][2] * pWnd->m_lfPosZ
                           + TransMatrix[0][3];
            PtRelToCS[0].y = TransMatrix[1][0] * pWnd->m_lfPosX
                           + TransMatrix[1][1] * pWnd->m_lfPosY
                           + TransMatrix[1][2] * pWnd->m_lfPosZ
                           + TransMatrix[1][3];
            PtRelToCS[0].z = TransMatrix[2][0] * pWnd->m_lfPosX
                           + TransMatrix[2][1] * pWnd->m_lfPosY
                           + TransMatrix[2][2] * pWnd->m_lfPosZ
                           + TransMatrix[2][3];
        }
        else
        {
            PtRelToCS[0].x = pWnd->m_lfPosX;
            PtRelToCS[0].y = pWnd->m_lfPosY;
            PtRelToCS[0].z = pWnd->m_lfPosZ;
        }

        pWnd->m_lfPosX2 = PtRelToCS[0].x;
        pWnd->m_lfPosY2 = PtRelToCS[0].y;
        pWnd->m_lfPosZ2 = PtRelToCS[0].z;

        pWnd->m_uiCompThreadFreq++;
        Sleep(50);
    }
    return 0;

oid CTrackingClassDlg::OnRadioGpib()

    // TODO: Add your control notification handler code here
    m_ctrlRadioGPIB.SetCheck(1);
    m_ctrlRadioRS232.SetCheck(0);
    m_bGPIBorRS232 = true;


oid CTrackingClassDlg::OnRadioRs232()

    // TODO: Add your control notification handler code here
    m_ctrlRadioGPIB.SetCheck(0);
    m_ctrlRadioRS232.SetCheck(1);
    m_bGPIBorRS232 = false;


oid CTrackingClassDlg::OnOK()

    // TODO: Add extra validation here
    if (m_bDataThread)
    {
        m_bDataThread = false;
        CloseHandle(g_pComputationThread->m_hThread);
    }
    if (m_bFileFlag)
        fclose(fp);
    if (m_bPsdThread)
        Psd.StopThread();
    if (m_bZygoThread)
        Zygo.StopThread();
    //MyRobot->UpdateStatus();
    //if (MyRobot->IsServoOn())
    //  MyRobot->SetServo(OFF);
    CDialog::OnOK();


oid CTrackingClassDlg::OnButtonTrack()

    // TODO: Add your control notification handler code here
    if (!m_bTrack)
    {
        m_sStatus = "TRACKING...";
        m_ctrlTrack.SetBitmap(hImageStop);
        m_bTrack = true;
        AcroSendString("RUN", 0);
    }
    else
    {
        m_sStatus = "POLLING...";
        m_ctrlTrack.SetBitmap(hImageStart);
        m_bTrack = false;
        AcroSendString("JOG OFF X Y", 0);
        AcroSendString("HALT", 0);
    }


oid CTrackingClassDlg::OnButtonSetBb()

    // TODO: Add your control notification handler code here
    AcroSendString("res x", 0);
    AcroSendString("res y", 0);
    if (m_bAPRorCAT)
        AcroSendString("P6160 = 1373", 0); //for APR on poles
        //AcroSendString("P6160 = 1186", 0); // for APR on Leica Adatper
    else
        AcroSendString("P6160 = 1008", 0);
    m_ctrlGoBB.EnableWindow(1);


oid CTrackingClassDlg::OnButtonGoBb()

    // TODO: Add your control notification handler code here
    AcroSendString("JOG VEL X0.1 Y0.1", 0);
    AcroSendString("JOG ABS X0 Y0",0);


oid CTrackingClassDlg::OnRadioApr()

    // TODO: Add your control notification handler code here
    m_ctrlRadioAPR.SetCheck(1);
    m_ctrlRadioCAT.SetCheck(0);
    m_bAPRorCAT = true;


oid CTrackingClassDlg::OnRadioCat()

    // TODO: Add your control notification handler code here
    m_ctrlRadioAPR.SetCheck(0);
    m_ctrlRadioCAT.SetCheck(1);
    m_bAPRorCAT = false;


oid CTrackingClassDlg::OnButtonStaticSample()

    // TODO: Add your control notification handler code here

    CStaticPt StaticPt;
    char buffer[100];
    char temp[100];

    if (StaticPt.DoModal() == IDOK)
    {
        if (bReplace)
        {
            PtRelToLISMCartesian[CurrLocationPointer].x = m_lfPosX;
            PtRelToLISMCartesian[CurrLocationPointer].y = m_lfPosY;
            PtRelToLISMCartesian[CurrLocationPointer].z = m_lfPosZ;

            sprintf(buffer, "RefPoint%d", CurrLocationPointer+1);
            WritePrivateProfileString(buffer, "Name", PtRelToLISMCartesian[CurrLocationPointer]
name, "RefPtLISM.ini");
            sprintf(temp, "%lf", PtRelToLISMCartesian[CurrLocationPointer].x);
            WritePrivateProfileString(buffer, "X", temp, "RefPtLISM.ini");
            sprintf(temp, "%lf", PtRelToLISMCartesian[CurrLocationPointer].y);
            WritePrivateProfileString(buffer, "Y", temp, "RefPtLISM.ini");
            sprintf(temp, "%lf", PtRelToLISMCartesian[CurrLocationPointer].z);
            WritePrivateProfileString(buffer, "Z", temp, "RefPtLISM.ini");
            bReplace = false;
        }
        else
        {
            PtRelToLISMCartesian[LocationPointer].x = m_lfPosX;
            PtRelToLISMCartesian[LocationPointer].y = m_lfPosY;
            PtRelToLISMCartesian[LocationPointer].z = m_lfPosZ;

            sprintf(buffer, "%d", LocationPointer+1);
            WritePrivateProfileString("No", "Total Pt Num", buffer, "RefPtLISM.ini");
            sprintf(buffer, "RefPoint%d", LocationPointer+1);
            WritePrivateProfileString(buffer, "Name", PtRelToLISMCartesian[LocationPointer].nam
, "RefPtLISM.ini");
            sprintf(temp, "%lf", PtRelToLISMCartesian[LocationPointer].x);
            WritePrivateProfileString(buffer, "X", temp, "RefPtLISM.ini");
            sprintf(temp, "%lf", PtRelToLISMCartesian[LocationPointer].y);
            WritePrivateProfileString(buffer, "Y", temp, "RefPtLISM.ini");
            sprintf(temp, "%lf", PtRelToLISMCartesian[LocationPointer].z);
            WritePrivateProfileString(buffer, "Z", temp, "RefPtLISM.ini");
            LocationPointer++;
        }
    }

oid CTrackingClassDlg::OnButtonDynamicSample()

    // TODO: Add your control notification handler code here
    if (!m_bDynamicSample)
    {
        m_bDynamicSample = true;
        m_ctrlDynamicSample.SetWindowText("STOP SAMPLE");
    }
    else
    {
        m_bDynamicSample = false;
        m_ctrlDynamicSample.SetWindowText("DYNAMIC SAMPLE");
    }

oid CTrackingClassDlg::OnButtonOpenPort()

    // TODO: Add your control notification handler code here
    if(!serial.IsOpened())
    {
        if (serial.Open(2,19200))
        {
            m_ctrlOpenPort.SetWindowText("CLOSE PORT");
            m_sStatus = "Port Opened";
        }
        else
            m_sStatus = "Port Open FAILED";
    }
    else
    {
        serial.Close();
        m_sStatus = "Port Closed";
        m_ctrlOpenPort.SetWindowText("OPEN PORT");
    }

    UpdateData(false);

UINT CTrackingClassDlg::TurboTryThread(LPVOID pParam)

    THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
    CTrackingClassDlg* pTurbo = (CTrackingClassDlg*) pThreadParams->lParam;
    delete pThreadParams;
```

```
// ** Set the number of charaters to send and receive
const int nNumSendChar = 12;
const int nNumRecvChar = 0;

// ** For sending array of short data  - 2 Bytes x 6 = 12 Bytes ** //
char chSendData[nNumSendChar];
// ** Special sequence of characters to send to tell Turbo App to stop
const char chSendEnd[12] = {127,-128,0,0,0,0,0,0,0,0,127,-128 };
// ** For storing the movement data
short nMoveData[6];
short Tx, Ty, Tz;
// ** For sending one char * //
//char chSendData[nNumSendChar] = {'A'};

// ** For receiving data ** //
char chRecvData[nNumRecvChar+1];

// For display of received values
CString szRecvData;
pTurbo->m_lfTurboTryVel = 0.0;

while (pTurbo->m_bTurboTryThread)
{
    //while (!pTurbo->m_bUpdateTurbo)
    //  ;
    pTurbo->m_lfTurboTryVel += pTurbo->m_lfTurboTryAcc * 0.015;
    if (pTurbo->m_lfTurboTryVel > 10.0)
        pTurbo->m_lfTurboTryVel = 10.0;
    double m_dDist  = pTurbo->m_lfTurboTryVel * 1000 * 0.015;

    // ** Calculate which axis to move
    //Tx = short(-1*pTurbo->m_bTx * m_dDist);
    //Ty = short(pTurbo->m_bTy * m_dDist);
    //Tz = short(-1*pTurbo->m_bTz * m_dDist);

    Tx = short(pTurbo->m_bTy * m_dDist);
    Ty = short(-1*pTurbo->m_bTx * m_dDist);
    Tz = short(pTurbo->m_bTz * m_dDist);

    // ** Get current Movement data and copy to Send Buffer
    nMoveData[0] = Tx; //pTurbo->m_nTx;
    nMoveData[1] = Ty; //pTurbo->m_nTy;
    nMoveData[2] = Tz; //pTurbo->m_nTz;
    nMoveData[3] = (short) 0.0; //pTurbo->m_nRx;
    nMoveData[4] = (short) 0.0; //pTurbo->m_nRy;
    nMoveData[5] = (short) 0.0; //pTurbo->m_nRz;
    memcpy(chSendData, nMoveData, nNumSendChar);

    // ** Send Data
    pTurbo->serial.SendData(chSendData, nNumSendChar);
    //pTurbo->m_bUpdateTurbo = false;

    // ** Wait for a confirmation character ** //
    while (pTurbo->serial.ReadDataWaiting() < 1)
        ;
    pTurbo->serial.ReadData(chRecvData, 1);

    // ** If receiving data from MRC ** //
    if(nNumRecvChar > 0)
    {
        // ** Wait for data from MRC to arrive
        while (pTurbo->serial.ReadDataWaiting() < nNumRecvChar)
        ;
        // ** Read the received data
        pTurbo->serial.ReadData(chRecvData, nNumRecvChar);

        // ** Mem copy the data to required data type
        // ** For receiving XYZ Data - 3 x long (4 bytes) = 12 Bytes
        //memcpy(pTurbo->m_nXYZData, chRecvData, 12);
        // ** For receiving XYZ Data - 3 x short (2 bytes) = 6 Bytes
        //memcpy(pTurbo->m_nRotData, (chRecvData + 12), 6);
    }
    // ** Increment the Frequency counter
    pTurbo->m_uiTurboFreq++;

    // ** Update the current pitch of the tool
    //pTurbo->m_dRotX += (nMoveData[3]/100.0);
    //pTurbo->m_bTurboTryThread = false;
    }
    return 0;
}

void CTrackingClassDlg::OnButtonDriveRobot()
{
    // TODO: Add your control notification handler code here
    char buffer[256];
    if (m_bPathDrive)
    {
        if (!m_bDriveRobot)
        {
            m_bDriveRobot = true;
            m_ctrlDriveRobot.SetWindowText("STOP ROBOT");
            UpdateData(false);

            AfxMessageBox("Make sure the other tracker start recording", MB_OK);

            THREADPARAMS* pThreadParamsDriveRobot = new THREADPARAMS;
            pThreadParamsDriveRobot->lParam = (LPARAM) this;
            g_pDriveRobotThread = AfxBeginThread(DriveRobotThread, pThreadParamsDriveRobot,
                                                 THREAD_PRIORITY_NORMAL, NULL);
        }
        else
        {
            m_bDriveRobot = false;
            m_ctrlDriveRobot.SetWindowText("DRIVE ROBOT");
            CloseHandle(g_pDriveRobotThread->m_hThread);
        }
    }
    if (m_bMotocomDrive)
    {
        UpdateData(true);
        double dPos[12] = {0,0,0,0,0,180,0,0,0,0,0,0};

        //Send points to robot controller
        //Send Move to Robot
        CString szMovType = "MOVL";
        CString szSpeedType = "V";
        double dMovSpeed = 10;
        CString szFrame = "UF5";
        WORD wForm = 0;
        int nTool = 1;

        dPos[0] = m_fDx;
        dPos[1] = m_fDy;
        dPos[2] = m_fDz;
        MyRobot->SetMode(PLAY);
        MyRobot->SetServo(ON);
        /*MyRobot->GetCurrentPos(1, &pRobotCurrentPos);
        m_lfRobotX = pRobotCurrentPos.X;
        m_lfRobotY = pRobotCurrentPos.Y;
        m_lfRobotZ = pRobotCurrentPos.Z;
        UpdateData(false);*/
        MyRobot->Move( szMovType, szSpeedType, dMovSpeed, szFrame, wForm, nTool, dPos);
        MyRobot->UpdateStatus();
        while (MyRobot->IsRobotOperating())
            MyRobot->UpdateStatus();
        MyRobot->GetCurrentPos(1, &pRobotCurrentPos);
        m_lfRobotX = pRobotCurrentPos.X;
        m_lfRobotY = pRobotCurrentPos.Y;
        m_lfRobotZ = pRobotCurrentPos.Z;
        if (m_bDriveRobotWrite)
        {
            sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t\n", m_lfRobotX, m_lfRob
otY, m_lfRobotZ, PtRelToCS[0].x, PtRelToCS[0].y, PtRelToCS[0].z);
            fprintf(FileRobotDr, buffer);
        }

        UpdateData(false);

    }
}

void CTrackingClassDlg::OnCheckTx()
{
    // TODO: Add your control notification handler code here
    if (!m_bTx)
        m_bTx = true;
```

```
   else
       m_bTx = false;


oid CTrackingClassDlg::OnCheckTy()

   // TODO: Add your control notification handler code here
   if (!m_bTy)
       m_bTy = true;
   else
       m_bTy = false;


oid CTrackingClassDlg::OnCheckTz()

   // TODO: Add your control notification handler code here
   if (!m_bTz)
       m_bTz = true;
   else
       m_bTz = false;


oid CTrackingClassDlg::OnButtonReset()

   // TODO: Add your control notification handler code here
   m_lfTurboTryVel = 0.0;
   m_lfTurboTryAcc = 0.0;


oid CTrackingClassDlg::OnChangeEditAcc()

   // TODO: If this is a RICHEDIT control, the control will not
   // send this notification unless you override the CDialog::OnInitDialog()
   // function and call CRichEditCtrl().SetEventMask()
   // with the ENM_CHANGE flag ORed into the mask.

   // TODO: Add your control notification handler code here
   UpdateData(true);


oid CTrackingClassDlg::OnButtonTransformation()

   // TODO: Add your control notification handler code here
   CTransformation Transformation;
   char buffer[100];
   if (Transformation.DoModal() == IDOK)
   {
       RefRelToLISMCartesian[0] = PtRelToLISMCartesian[OriginSelect];
       RefRelToLISMCartesian[1] = PtRelToLISMCartesian[PosXSelect];
       RefRelToLISMCartesian[2] = PtRelToLISMCartesian[PosYSelect];
       m_sStatus = "Performing Transformation...";
       UpdateData(false);
       CTrackingClassDlg::Transformation();
       m_ctrlComboCS.AddString(CS[CS_Counter-1]);
       sprintf(buffer, "CS%d", CS_Counter);
       WritePrivateProfileString(buffer, "Name", CS[CS_Counter-1], "RefPtLISM.ini");
       //sprintf(buffer, "%d", CS_Counter);
       WritePrivateProfileString("CSName", "Name", CS[CS_Counter-1], "TransMatrixLISM.ini");
   }


oid CTrackingClassDlg::Transformation()

   double distx, disty, distz, singamma, sinbeta, cosdelta, sindelta, dot23;
   double R1X3, R1Y3;
   REC_DATA_CARTESIAN RefRelToTrackerCartesian[3];
   double alpha, beta, gamma;                          //Rotation angles for transformation
                                                       //calculation LISM->CS
   double dist12, dist13;                              //distance data used for transformation

                                                       //calculation LISM->CS
   char buffer[256], temp[256];

   for (int i =0; i<3; i++)
   {
       RefRelToTrackerCartesian[i] = RefRelToLISMCartesian[i];
   }

   dist12 = sqrt(pow(RefRelToTrackerCartesian[0].x - RefRelToTrackerCartesian[1].x, 2) +
                 pow(RefRelToTrackerCartesian[0].y - RefRelToTrackerCartesian[1].y, 2) +
                 pow(RefRelToTrackerCartesian[0].z - RefRelToTrackerCartesian[1].z, 2)) ;
   dist13 = sqrt(pow(RefRelToTrackerCartesian[0].x - RefRelToTrackerCartesian[2].x, 2) +
                 pow(RefRelToTrackerCartesian[0].y - RefRelToTrackerCartesian[2].y, 2) +
                 pow(RefRelToTrackerCartesian[0].z - RefRelToTrackerCartesian[2].z, 2)) ;
   sinbeta = (RefRelToTrackerCartesian[0].z - RefRelToTrackerCartesian[1].z) / dist12;
   if (sinbeta > 1.0)
       sinbeta = 1.0;
   if (sinbeta < -1.0)
       sinbeta = -1.0;
   beta = asin(sinbeta);
   distx = RefRelToTrackerCartesian[1].x - RefRelToTrackerCartesian[0].x;
   disty = RefRelToTrackerCartesian[1].y - RefRelToTrackerCartesian[0].y;
   alpha = atan2(disty, distx);
   distz = RefRelToTrackerCartesian[2].z - RefRelToTrackerCartesian[0].z;

   dot23 = (RefRelToTrackerCartesian[1].x - RefRelToTrackerCartesian[0].x) * (RefRelToTrackerC
rtesian[2].x - RefRelToTrackerCartesian[0].x)
           + (RefRelToTrackerCartesian[1].y - RefRelToTrackerCartesian[0].y) * (RefRelToTracke
Cartesian[2].y - RefRelToTrackerCartesian[0].y)
           + (RefRelToTrackerCartesian[1].z - RefRelToTrackerCartesian[0].z) * (RefRelToTracke
Cartesian[2].z - RefRelToTrackerCartesian[0].z);
   cosdelta = dot23 / dist12 / dist13;
   sindelta = sqrt(1-pow(cosdelta,2));
   R1X3 = dist13 * cosdelta;
   R1Y3 = dist13 * sindelta;
   singamma = (distz + sinbeta * R1X3) / (cos(beta) * R1Y3);
   //singamma = distz / dist13 / cos(beta);
   if (singamma > 1.0)
       singamma = 1.0;
   if (singamma < -1.0)
       singamma = -1.0;
   gamma = asin(singamma);
   TransMatrix[0][0] = cos(alpha) * cos(beta);
   TransMatrix[0][1] = sin(alpha) * cos(beta);
   TransMatrix[0][2] = -1.0 * sin(beta);
   TransMatrix[0][3] = -1.0 * (TransMatrix[0][0] * RefRelToTrackerCartesian[0].x
                       + TransMatrix[0][1] * RefRelToTrackerCartesian[0].y
                       + TransMatrix[0][2] * RefRelToTrackerCartesian[0].z);

   TransMatrix[1][0] = cos(alpha) * sin(beta) * sin(gamma) - sin(alpha) * cos(gamma);
   TransMatrix[1][1] = sin(alpha) * sin(beta) * sin(gamma) + cos(alpha) * cos(gamma);
   TransMatrix[1][2] = cos(beta) * sin(gamma);
   TransMatrix[1][3] = -1.0 * (TransMatrix[1][0] * RefRelToTrackerCartesian[0].x
                       + TransMatrix[1][1] * RefRelToTrackerCartesian[0].y
                       + TransMatrix[1][2] * RefRelToTrackerCartesian[0].z);

   TransMatrix[2][0] = cos(alpha) * sin(beta) * cos(gamma) + sin(alpha) * sin(gamma);
   TransMatrix[2][1] = sin(alpha) * sin(beta) * cos(gamma) - cos(alpha) * sin(gamma);
   TransMatrix[2][2] = cos(beta) * cos(gamma);
   TransMatrix[2][3] = -1.0 * (TransMatrix[2][0] * RefRelToTrackerCartesian[0].x
                       + TransMatrix[2][1] * RefRelToTrackerCartesian[0].y
                       + TransMatrix[2][2] * RefRelToTrackerCartesian[0].z);

   InvTMatrix[0][0] = TransMatrix[0][0];
   InvTMatrix[0][1] = TransMatrix[1][0];
   InvTMatrix[0][2] = TransMatrix[2][0];
   InvTMatrix[0][3] = RefRelToLISMCartesian[0].x;

   InvTMatrix[1][0] = TransMatrix[0][1];
   InvTMatrix[1][1] = TransMatrix[1][1];
   InvTMatrix[1][2] = TransMatrix[2][1];
   InvTMatrix[1][3] = RefRelToLISMCartesian[0].y;

   InvTMatrix[2][0] = TransMatrix[0][2];
   InvTMatrix[2][1] = TransMatrix[1][2];
   InvTMatrix[2][2] = TransMatrix[2][2];
   InvTMatrix[2][3] = RefRelToLISMCartesian[0].z;

   for (int j=0; j<3; j++)
   {
       sprintf(buffer, "Row%d", j);
       sprintf(temp, "%lf", TransMatrix[j][0]);
       WritePrivateProfileString(buffer, "Col0",temp, "TransMatrixLISM.ini");
       sprintf(temp, "%lf", TransMatrix[j][1]);
       WritePrivateProfileString(buffer, "Col1",temp, "TransMatrixLISM.ini");
       sprintf(temp, "%lf", TransMatrix[j][2]);
       WritePrivateProfileString(buffer, "Col2",temp, "TransMatrixLISM.ini");
       sprintf(temp, "%lf", TransMatrix[j][3]);
```

```
        WritePrivateProfileString(buffer, "Col3",temp, "TransMatrixLISM.ini");
    }
    for (j=0; j<3; j++)
    {
        sprintf(buffer, "InvRow%d", j);
        sprintf(temp, "%lf", InvTMatrix[j][0]);
        WritePrivateProfileString(buffer, "Col0",temp, "TransMatrixLISM.ini");
        sprintf(temp, "%lf", InvTMatrix[j][1]);
        WritePrivateProfileString(buffer, "Col1",temp, "TransMatrixLISM.ini");
        sprintf(temp, "%lf", InvTMatrix[j][2]);
        WritePrivateProfileString(buffer, "Col2",temp, "TransMatrixLISM.ini");
        sprintf(temp, "%lf", InvTMatrix[j][3]);
        WritePrivateProfileString(buffer, "Col3",temp, "TransMatrixLISM.ini");
    }
    UpdateData(false);

 oid CTrackingClassDlg::OnButtonPathGeneration()

    // TODO: Add your control notification handler code here

    CPath PathGen;
    if ((FilePath = fopen("Path.txt", "w")) == NULL)
    {
        puts("cannot open file");
        exit(1);
    }
    rewind(FilePath);            // set the cursor to the beginning of file
    fprintf(FilePath, "PathX\t PathY\t PathZ\t\n");

    if (PathGen.DoModal() == IDOK)
    {
        fclose(FilePath);
    }

 oid CTrackingClassDlg::OnButtonEgtfs()

    // TODO: Add your control notification handler code here
    if (!m_bEGTFS && !m_bEGTFSTurbo)
    {
        if ((FileEgtfs = fopen("Egtfs.txt", "w")) == NULL)
        {
            puts("cannot open file");
            exit(1);
        }
        rewind(FileEgtfs);           // set the cursor to the beginning of file
        fprintf(FileEgtfs, "RobotCurMX\t RobotCurMY\t RobotCurMZ\t\n");

        if (m_ctrlRadioMotocom2.GetCheck())
        {
            m_bEGTFS = true;
            THREADPARAMS* pThreadParamsEGTFS = new THREADPARAMS;
            pThreadParamsEGTFS->lParam = (LPARAM) this;
            g_pEGTFSThread = AfxBeginThread(EGTFSThread, pThreadParamsEGTFS,
                                         THREAD_PRIORITY_NORMAL, NULL);
        }
        if (m_ctrlRadioTurbo2.GetCheck())
        {
            m_bTurboCorrThread = true;
            THREADPARAMS* pThreadParamsTurboCorr = new THREADPARAMS;
            pThreadParamsTurboCorr->lParam = (LPARAM) this;
            g_pTurboCorrThread = AfxBeginThread(TurboCorrectionThread, pThreadParamsTurboCorr,
                                         THREAD_PRIORITY_NORMAL, NULL);
            m_bEGTFSTurbo = true;
            THREADPARAMS* pThreadParamsEGTFSTurbo = new THREADPARAMS;
            pThreadParamsEGTFSTurbo->lParam = (LPARAM) this;
            g_pEGTFSTurboThread = AfxBeginThread(EGTFSTurboThread, pThreadParamsEGTFSTurbo,
                                         THREAD_PRIORITY_NORMAL, NULL);
        }
        m_ctrlButtonEGTFS.SetWindowText("STOP EGTFS");
    }
    else
    {
        fclose(FileEgtfs);
        if (m_bEGTFS)
        {
            m_bEGTFS = false;
            CloseHandle(g_pEGTFSThread->m_hThread);
        }
        if (m_bEGTFSTurbo)
        {
            m_bTurboCorrThread = false;
            CloseHandle(g_pTurboCorrThread->m_hThread);
            m_bEGTFSTurbo = false;
            CloseHandle(g_pEGTFSTurboThread->m_hThread);
            CurrPt = 0;
            CurrLine = 0;
        }
        m_ctrlButtonEGTFS.SetWindowText("START EGTFS");
    }


oid CTrackingClassDlg::OnButtonGuidance()

    // TODO: Add your control notification handler code here
    if (!m_bGuide)
    {
        if ((FileGuidance = fopen("Guidance.txt", "w")) == NULL)
        {
            puts("cannot open file");
            exit(1);
        }
        rewind(FileGuidance);            // set the cursor to the beginning of file
        fprintf(FileGuidance, "PathX\tPathY\tPathZ\tRobotCurX\t RobotCurY\t RobotCurZ\t\n");

        //AcroSendString("HALT", 0);
        AcroSendString("JOG VEL X0.5 Y4", 0);
        AcroSendString("JOG ACC X0.5 Y5", 0);
        AcroSendString("JOG DEC X0.5 Y5", 0);
        OnButtonTrack();

        if (m_ctrlRadioTurbo.GetCheck())
        {
            m_bTurboCorrThread = true;
            THREADPARAMS* pThreadParamsTurboCorr = new THREADPARAMS;
            pThreadParamsTurboCorr->lParam = (LPARAM) this;
            g_pTurboCorrThread = AfxBeginThread(TurboCorrectionThread, pThreadParamsTurboCorr,
                                         THREAD_PRIORITY_NORMAL, NULL);
        }
        m_bGuide = true;
        THREADPARAMS* pThreadParamsGuidance = new THREADPARAMS;
        pThreadParamsGuidance->lParam = (LPARAM) this;
        g_pGuidanceThread = AfxBeginThread(GuidanceThread, pThreadParamsGuidance,
                                         THREAD_PRIORITY_NORMAL, NULL);
        m_ctrlButtonGuidance.SetWindowText("STOP GUIDANCE");
    }
    else
    {
        fclose(FileGuidance);
        //AcroSendString("RUN", 0);
        AcroSendString("JOG VEL X2 Y4", 0);
        AcroSendString("JOG ACC X3 Y5", 0);
        AcroSendString("JOG DEC X3 Y5", 0);
        OnButtonTrack();

        if (m_bTurboCorrThread)
        {
            m_bTurboCorrThread = false;
            CloseHandle(g_pTurboCorrThread->m_hThread);
        }

        m_bGuide = false;
        CloseHandle(g_pGuidanceThread->m_hThread);
        m_ctrlButtonGuidance.SetWindowText("START GUIDANCE");
    }


INT CTrackingClassDlg::GuidanceThread(LPVOID pParam)

    THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
    CTrackingClassDlg* pGuide = (CTrackingClassDlg*) pThreadParams->lParam;
    delete pThreadParams;
    double DiffX, DiffY, DiffZ;
    char buffer[256];
    bool first = true;
    while (pGuide->m_bGuide)
    {
```

```
    pGuide->m_uiGuidanceFreq++;
    if (CurrLine < NumOfLine)
    {
        if (CurrPt < NumOfPt[CurrLine])
        {
            ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t", p
ide->m_lfPSDX, pGuide->m_lfPSDY, PointList[CurrLine][CurrPt].x, PointList[CurrLine][CurrPt].y
PointList[CurrLine][CurrPt].z, PtRelToCS[0].x, PtRelToCS[0].y, PtRelToCS[0].z);
            ::fprintf(FileGuidance, buffer);

            pGuide->DriveBeam();
            //if (first)
            //{
                Sleep(300);
            //  first = false;
            //}
            //else Sleep(100);
            ::sprintf(buffer, "%.3lf\t%.3lf\t \n", pGuide->m_lfPSDX, pGuide->m_lfPSDY);
            ::fprintf(FileGuidance, buffer);
            //::AfxMessageBox(buffer, IDOK, NULL);
            if (pGuide->m_ctrlRadioMotocom.GetCheck())
                pGuide->MotocomCorrection();
            //else
            //  pGuide->TurboCorrection();
            //Sleep(100);
            CurrPt++;
        }
        else
        {
            ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t \n"
pGuide->m_lfPSDX, pGuide->m_lfPSDY, PointList[CurrLine][NumOfPt[CurrLine]-1].x, PointList[Cur
Line][NumOfPt[CurrLine]-1].y, PointList[CurrLine][NumOfPt[CurrLine]-1].z, PtRelToCS[0].x, PtRe
ToCS[0].y, PtRelToCS[0].z);
            ::fprintf(FileGuidance, buffer);
            DiffX = fabs(PointList[CurrLine][NumOfPt[CurrLine]-1].x - PtRelToCS[0].x);
            DiffY = fabs(PointList[CurrLine][NumOfPt[CurrLine]-1].y - PtRelToCS[0].y);
            DiffZ = fabs(PointList[CurrLine][NumOfPt[CurrLine]-1].z - PtRelToCS[0].z);
Check*/     if ((DiffX>0.1) || (DiffY>0.1) || (DiffZ>0.1))
            {
                if (pGuide->m_ctrlRadioMotocom.GetCheck())
                    pGuide->MotocomCorrection();
            //  else
            //      pGuide->TurboCorrection();
                //Sleep(100);
            }
            else
            {
                CurrPt = 0;
                CurrLine++;
            }
        }
    }
    else
    {
        ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t \n", pG
ide->m_lfPSDX, pGuide->m_lfPSDY, PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x, PointList[Cur
Line-1][NumOfPt[CurrLine-1]-1].y, PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z, PtRelToCS[0]
x, PtRelToCS[0].y, PtRelToCS[0].z);
        ::fprintf(FileGuidance, buffer);
        DiffX = fabs(PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x - PtRelToCS[0].x);
        DiffY = fabs(PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y - PtRelToCS[0].y);
        DiffZ = fabs(PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z - PtRelToCS[0].z);
*Check*/ if ((DiffX>0.1) || (DiffY>0.1) || (DiffZ>0.1))
        {
            if (pGuide->m_ctrlRadioMotocom.GetCheck())
                pGuide->MotocomCorrection();
            //else
            //  pGuide->TurboCorrection();
            //Sleep(100);
        }
        else
        {
            ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t \n"
pGuide->m_lfPSDX, pGuide->m_lfPSDY, PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x, PointList
CurrLine-1][NumOfPt[CurrLine-1]-1].y, PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z, PtRelToC
[0].x, PtRelToCS[0].y, PtRelToCS[0].z);
            ::fprintf(FileGuidance, buffer);
            pGuide->OnButtonGuidance();
            CurrPt = 0;
```

```
              CurrLine = 0;
          }
      }
  }
  return 0;

NT CTrackingClassDlg::EGTFSThread(LPVOID pParam)

  //double RobotCurrMX, RobotCurrMY, RobotCurrMZ; //Robot current position recorded by contro
er
  double RobotMovX, RobotMovY, RobotMovZ;      //Robot increment to move to next control point
  char buffer[256];

  THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
  CTrackingClassDlg* pE = (CTrackingClassDlg*) pThreadParams->lParam;
  delete pThreadParams;

  while (pE->m_bEGTFS)
  {
      if (!pE->MyRobot->IsRobotOperating())
      {
          pE->MyRobot->GetCurrentPos(1, &pE->pRobotCurrentPos);
          pE->m_lfRobotX = pE->pRobotCurrentPos.X;
          pE->m_lfRobotY = pE->pRobotCurrentPos.Y;
          pE->m_lfRobotZ = pE->pRobotCurrentPos.Z;
          //RobotCurX = PtRelToCS[0].x;
          //RobotCurY = PtRelToCS[0].y;
          //RobotCurZ = PtRelToCS[0].z;
          ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t \n", pE->m_lfRobotX,
->m_lfRobotY, pE->m_lfRobotZ, PtRelToCS[0].x, PtRelToCS[0].y, PtRelToCS[0].z);
          ::fprintf(FileEgtfs, buffer);

          if (CurrLine < NumOfLine)
          {
              if (CurrPt < NumOfPt[CurrLine])
              {
                  RobotMovX = (float) (PointList[CurrLine][CurrPt].x - PtRelToCS[0].x);
                  RobotMovY = (float) (PointList[CurrLine][CurrPt].y - PtRelToCS[0].y);
                  RobotMovZ = (float) (PointList[CurrLine][CurrPt].z - PtRelToCS[0].z);
                  CurrPt++;
              }
              else
              {
                  RobotMovX = (float) (PointList[CurrLine][CurrPt-1].x - PtRelToCS[0].x);
                  RobotMovY = (float) (PointList[CurrLine][CurrPt-1].y - PtRelToCS[0].y);
                  RobotMovZ = (float) (PointList[CurrLine][CurrPt-1].z - PtRelToCS[0].z);
                  if (fabs(RobotMovX) < 0.1 && fabs(RobotMovY) < 0.1 && fabs(RobotMovZ) < 0.1

                  {
                      CurrPt = 0;
                      CurrLine++;
                  }
              }
          }
          else
          {
              RobotMovX = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x - PtRelToCS
0].x);
              RobotMovY = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y - PtRelToCS
0].y);
              RobotMovZ = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z - PtRelToCS
0].z);
              if (fabs(RobotMovX) < 0.1 && fabs(RobotMovY) < 0.1 && fabs(RobotMovZ) < 0.1)
              {
                  CurrPt = 0;
                  CurrLine = 0;
                  pE->m_bEGTFS = false;
              }
          }
          double dPos[12] = {0,0,0,0,0,0,0,0,0,0,0,0};

          CString szMovType = "IMOV";
          CString szSpeedType = "V";
          double dMovSpeed = pE->m_lfVel;
          CString szFrame = "UF5";
          WORD wForm = 0;
          int nTool = 1;
          dPos[0] = RobotMovX;
          dPos[1] = RobotMovY;
          dPos[2] = RobotMovZ;

          pE->MyRobot->Move(szMovType, szSpeedType, dMovSpeed, szFrame, wForm, nTool, dPos);
          pE->MyRobot->UpdateStatus();
      }
      else
      {
          pE->MyRobot->UpdateStatus();
      }
      ::Sleep(100); //wait for tracker to update positions
  }
  pE->m_ctrlButtonEGTFS.SetWindowText("START EGTFS");
  for (int i =0; i<10; i++)
  {
      ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t \n", pE->m_lfRobotX, pE->
lfRobotY, pE->m_lfRobotZ, PtRelToCS[0].x, PtRelToCS[0].y, PtRelToCS[0].z);
      ::fprintf(FileEgtfs, buffer);
      ::Sleep(100);
  }
  ::fclose(FileEgtfs);
  return 0;


NT CTrackingClassDlg::EGTFSTurboThread(LPVOID pParam)

  THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
  CTrackingClassDlg* pET = (CTrackingClassDlg*) pThreadParams->lParam;
  delete pThreadParams;

  char buffer[256];
  //double RobotMovX, RobotMovY, RobotMovZ;

  while (pET->m_bEGTFSTurbo)
  {
      if (CurrLine < NumOfLine)
      {
          if (CurrPt < NumOfPt[CurrLine])
          {
              ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t \n", PointList[Cu
Line][CurrPt].x, PointList[CurrLine][CurrPt].y, PointList[CurrLine][CurrPt].z, PtRelToCS[0].x
 PtRelToCS[0].y, PtRelToCS[0].z);
              ::fprintf(FileEgtfs, buffer);
              RobotMovXEGTFSTurbo = (float) (PointList[CurrLine][CurrPt].x - PtRelToCS[0].x);
              RobotMovYEGTFSTurbo = (float) (PointList[CurrLine][CurrPt].y - PtRelToCS[0].y);
              RobotMovZEGTFSTurbo = (float) (PointList[CurrLine][CurrPt].z - PtRelToCS[0].z);
              CurrPt++;
          }
          else
          {
              ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t \n", PointList[Cu
Line][CurrPt-1].x, PointList[CurrLine][CurrPt-1].y, PointList[CurrLine][CurrPt-1].z, PtRelToC
0].x, PtRelToCS[0].y, PtRelToCS[0].z);
              ::fprintf(FileEgtfs, buffer);
              RobotMovXEGTFSTurbo = (float) (PointList[CurrLine][CurrPt-1].x - PtRelToCS[0].x

              RobotMovYEGTFSTurbo = (float) (PointList[CurrLine][CurrPt-1].y - PtRelToCS[0].y

              RobotMovZEGTFSTurbo = (float) (PointList[CurrLine][CurrPt-1].z - PtRelToCS[0].z

              if (fabs(RobotMovXEGTFSTurbo) < 0.1 && fabs(RobotMovYEGTFSTurbo) < 0.1 && fabs(
botMovZEGTFSTurbo) < 0.1)
              {
                  CurrPt = 0;
                  CurrLine++;
              }
          }
      }
      else
      {
          ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t \n", PointList[CurrLi
e-1][NumOfPt[CurrLine-1]-1].x, PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y, PointList[CurrL
ne-1][NumOfPt[CurrLine-1]-1].z, PtRelToCS[0].x, PtRelToCS[0].y, PtRelToCS[0].z);
          ::fprintf(FileEgtfs, buffer);
          RobotMovXEGTFSTurbo = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x - PtR
lToCS[0].x);
          RobotMovYEGTFSTurbo = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y - PtR
lToCS[0].y);
```

```
          RobotMovZEGTFSTurbo = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z - PtR
ToCS[0].z);
          if (fabs(RobotMovXEGTFSTurbo) < 0.1 && fabs(RobotMovYEGTFSTurbo) < 0.1 && fabs(Robo
ovZEGTFSTurbo) < 0.1)
          {
              RobotMovXEGTFSTurbo = 0.0;
              RobotMovYEGTFSTurbo = 0.0;
              RobotMovZEGTFSTurbo = 0.0;

              for (int i =0; i<10; i++)
              {
              ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t \n", PointList[Cu
Line-1][NumOfPt[CurrLine-1]-1].x, PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y, PointList[C
rLine-1][NumOfPt[CurrLine-1]-1].z, PtRelToCS[0].x, PtRelToCS[0].y, PtRelToCS[0].z);
              ::fprintf(FileEgtfs, buffer);
              ::Sleep(100);
              }
              //::fclose(FileEgtfs);
              pET->OnButtonEgtfs();
          }
      }
      ::Sleep(15); //wait for tracker to update positions
  }
  return 0;

oid CTrackingClassDlg::DriveBeam()

  double XYLength;
  //char buffer[256];

  //RobotCurX = PtRelToCS[0].x;
  //RobotCurY = PtRelToCS[0].y;
  //RobotCurZ = PtRelToCS[0].z;
  RobotDesX = PointList[CurrLine][CurrPt].x;
  RobotDesY = PointList[CurrLine][CurrPt].y;
  RobotDesZ = PointList[CurrLine][CurrPt].z;

  LISMDesX = InvTMatrix[0][0] * RobotDesX
           + InvTMatrix[0][1] * RobotDesY
           + InvTMatrix[0][2] * RobotDesZ
           + InvTMatrix[0][3];
  LISMDesY = InvTMatrix[1][0] * RobotDesX
           + InvTMatrix[1][1] * RobotDesY
           + InvTMatrix[1][2] * RobotDesZ
           + InvTMatrix[1][3];
  LISMDesZ = InvTMatrix[2][0] * RobotDesX
           + InvTMatrix[2][1] * RobotDesY
           + InvTMatrix[2][2] * RobotDesZ
           + InvTMatrix[2][3];
  XYLength = pow((pow(LISMDesX,2) + pow(LISMDesY,2)), 0.5);
  M1.DriveAngle = (float) RadianToDegree(atan2(-1*LISMDesX, LISMDesY))/360.0 - M1.CurrentAngl
;
  M2.DriveAngle = (float) 0.5 * RadianToDegree(atan2(LISMDesZ, XYLength))/360.0 - M2.CurrentA
gle;
  //sprintf(buffer, "%.5lf\t%.5lf\t\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t\n", M1.DriveA
gle*360.0, M2.DriveAngle*360.0, PointList[CurrLine][CurrPt].x, PointList[CurrLine][CurrPt].y,
ointList[CurrLine][CurrPt].z, PtRelToCS[0].x, PtRelToCS[0].y, PtRelToCS[0].z);
  //m_sStatus = buffer;
  //UpdateData(false);
  //fprintf(FileGuidance, buffer);
  DriveMotor(M1, M2);

oid CTrackingClassDlg::MotocomCorrection()

  double RobotMovX, RobotMovY, RobotMovZ;
  if (CurrLine < NumOfLine)
  {
      if (CurrPt < NumOfPt[CurrLine])
      {
          RobotMovX = (float) (PointList[CurrLine][CurrPt].x - PtRelToCS[0].x);
          RobotMovY = (float) (PointList[CurrLine][CurrPt].y - PtRelToCS[0].y);
          RobotMovZ = (float) (PointList[CurrLine][CurrPt].z - PtRelToCS[0].z);
      }
      else
      {
          RobotMovX = (float) (PointList[CurrLine][CurrPt-1].x - PtRelToCS[0].x);
          RobotMovY = (float) (PointList[CurrLine][CurrPt-1].y - PtRelToCS[0].y);
```

```
            RobotMovZ = (float) (PointList[CurrLine][CurrPt-1].z - PtRelToCS[0].z);
        }
    }
    else
    {
        RobotMovX = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x - PtRelToCS[0].x);
        RobotMovY = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y - PtRelToCS[0].y);
        RobotMovZ = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z - PtRelToCS[0].z);
    }

    double dPos[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
    CString szMovType = "IMOV";
    CString szSpeedType = "V";
    double dMovSpeed = 1;
    CString szFrame = "UF5";
    WORD wForm = 0;
    int nTool = 1;

    dPos[0] = RobotMovX;
    dPos[1] = RobotMovY;
    dPos[2] = RobotMovZ;

    MyRobot->Move(szMovType, szSpeedType, dMovSpeed, szFrame, wForm, nTool, dPos);
    do
    {
        MyRobot->UpdateStatus();
    }
    while (MyRobot->IsRobotOperating());


id CTrackingClassDlg::TurboCorrection()

    double RobotMovX, RobotMovY, RobotMovZ;
    //double RobotCurX, RobotCurY, RobotCurZ;
    //char buffer[256];
    // ** Set the number of charaters to send and receive
    const int nNumSendChar = 12;
    const int nNumRecvChar = 0;

    // ** For sending array of short data  - 2 Bytes x 6 = 12 Bytes ** //
    char chSendData[nNumSendChar];
    // ** Special sequence of characters to send to tell Turbo App to stop
    const char chSendEnd[12] = {127,-128,0,0,0,0,0,0,0,0,127,-128 };
    // ** For storing the movement data
    short nMoveData[6];

    // ** For receiving data ** //
    char chRecvData[nNumRecvChar+1];

    // For display of received values
    CString szRecvData;

    if (CurrLine < NumOfLine)
    {
        if (CurrPt < NumOfPt[CurrLine])
        {
            RobotMovX = (float) (PointList[CurrLine][CurrPt].x - PtRelToCS[0].x);
            RobotMovY = (float) (PointList[CurrLine][CurrPt].y - PtRelToCS[0].y);
            RobotMovZ = (float) (PointList[CurrLine][CurrPt].z - PtRelToCS[0].z);
            RobotDesX = PointList[CurrLine][CurrPt].x;
            RobotDesY = PointList[CurrLine][CurrPt].y;
            RobotDesZ = PointList[CurrLine][CurrPt].z;
        }
        else
        {
            RobotMovX = (float) (PointList[CurrLine][CurrPt-1].x - PtRelToCS[0].x);
            RobotMovY = (float) (PointList[CurrLine][CurrPt-1].y - PtRelToCS[0].y);
            RobotMovZ = (float) (PointList[CurrLine][CurrPt-1].z - PtRelToCS[0].z);
            RobotDesX = PointList[CurrLine][CurrPt-1].x;
            RobotDesY = PointList[CurrLine][CurrPt-1].y;
            RobotDesZ = PointList[CurrLine][CurrPt-1].z;
        }
    }
    else
    {
        RobotMovX = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x - PtRelToCS[0].x);
        RobotMovY = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y - PtRelToCS[0].y);
        RobotMovZ = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z - PtRelToCS[0].z);
        RobotDesX = PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x;
        RobotDesY = PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y;
        RobotDesZ = PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z;
    }

    //Check this
    RobotX = -1.0 * RobotMovX * 400.0;
    RobotY = RobotMovY * 400.0;
    RobotZ = -1.0 * RobotMovZ * 400.0;

    if (fabs(RobotX) > 1000.0)
        RobotX = 1000.0;
    if (fabs(RobotY) > 1000.0)
        RobotY = 1000.0;
    if (fabs(RobotZ) > 1000.0)
        RobotZ = 1000.0;
    // ** Get current Movement data and copy to Send Buffer
    nMoveData[0] = (short) RobotX; //pTurbo->m_nTx;
    nMoveData[1] = (short) RobotY; //pTurbo->m_nTy;
    nMoveData[2] = (short) RobotZ; //pTurbo->m_nTz;
    nMoveData[3] = (short) 0.0; //pTurbo->m_nRx;
    nMoveData[4] = (short) 0.0; //pTurbo->m_nRy;
    nMoveData[5] = (short) 0.0; //pTurbo->m_nRz;
    memcpy(chSendData, nMoveData, nNumSendChar);

    // ** Send Data
    serial.SendData(chSendData, nNumSendChar);

    // ** Wait for a confirmation character ** //
    while (serial.ReadDataWaiting() < 1)
            ;
    serial.ReadData(chRecvData, 1);

    // ** If receiving data from MRC ** //
    if(nNumRecvChar > 0)
    {
        // ** Wait for data from MRC to arrive
        while (serial.ReadDataWaiting() < nNumRecvChar)
        ;
        // ** Read the received data
        serial.ReadData(chRecvData, nNumRecvChar);

        // ** Mem copy the data to required data type
        // ** For receiving XYZ Data - 3 x long (4 bytes) = 12 Bytes
        //memcpy(pTurbo->m_nXYZData, chRecvData, 12);
        // ** For receiving XYZ Data - 3 x short (2 bytes) = 6 Bytes
        //memcpy(pTurbo->m_nRotData, (chRecvData + 12), 6);
    }
    // ** Increment the Frequency counter
    //RobotCurX = PtRelToCS[0].x;
    //RobotCurY = PtRelToCS[0].y;
    //RobotCurZ = PtRelToCS[0].z;
    //sprintf(buffer, "%.51f\t%.51f\t%.51f\t\n", RobotCurX, RobotCurY, RobotCurZ);
    //fprintf(FileGuidance, buffer);
    m_uiTurboFreq++;


INT CTrackingClassDlg::TurboCorrectionThread(LPVOID pParam)

    THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
    CTrackingClassDlg* pT = (CTrackingClassDlg*) pThreadParams->lParam;
    delete pThreadParams;

    double RobotMovX, RobotMovY, RobotMovZ;
    //double RobotCurX, RobotCurY, RobotCurZ;
    //char buffer[256];
    // ** Set the number of charaters to send and receive
    const int nNumSendChar = 12;
    const int nNumRecvChar = 0;

    // ** For sending array of short data  - 2 Bytes x 6 = 12 Bytes ** //
    char chSendData[nNumSendChar];
    // ** Special sequence of characters to send to tell Turbo App to stop
    const char chSendEnd[12] = {127,-128,0,0,0,0,0,0,0,0,127,-128 };
    // ** For storing the movement data
    short nMoveData[6];

    // ** For receiving data ** //
    char chRecvData[nNumRecvChar+1];
```

```
// For display of received values
CString szRecvData;
char buffer[256];

while (pT->m_bTurboCorrThread)
{
    if (CurrLine < NumOfLine)
    {
        if (CurrPt < NumOfPt[CurrLine])
        {
            RobotMovX = (float) (PointList[CurrLine][CurrPt].x - PtRelToCS[0].x);
            RobotMovY = (float) (PointList[CurrLine][CurrPt].y - PtRelToCS[0].y);
            RobotMovZ = (float) (PointList[CurrLine][CurrPt].z - PtRelToCS[0].z);
            RobotDesX = PointList[CurrLine][CurrPt].x;
            RobotDesY = PointList[CurrLine][CurrPt].y;
            RobotDesZ = PointList[CurrLine][CurrPt].z;
        }
        else
        {
            RobotMovX = (float) (PointList[CurrLine][CurrPt-1].x - PtRelToCS[0].x);
            RobotMovY = (float) (PointList[CurrLine][CurrPt-1].y - PtRelToCS[0].y);
            RobotMovZ = (float) (PointList[CurrLine][CurrPt-1].z - PtRelToCS[0].z);
            RobotDesX = PointList[CurrLine][CurrPt-1].x;
            RobotDesY = PointList[CurrLine][CurrPt-1].y;
            RobotDesZ = PointList[CurrLine][CurrPt-1].z;
        }
    }
    else
    {
        RobotMovX = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x - PtRelToCS[0].
;
        RobotMovY = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y - PtRelToCS[0].
;
        RobotMovZ = (float) (PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z - PtRelToCS[0].
;
        RobotDesX = PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].x;
        RobotDesY = PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].y;
        RobotDesZ = PointList[CurrLine-1][NumOfPt[CurrLine-1]-1].z;
    }

    if (pT->m_bGuide)
    {
        //Check this
        RobotX = -1.0 * RobotMovX / 0.4 * 0.015 * 1000.0;
        RobotY = RobotMovY / 0.4 * 0.015 * 1000.0;
        RobotZ = -1.0 * RobotMovZ /0.4 * 0.015 * 1000.0;
    }
    if (pT->m_bEGTFSTurbo)
    {
        RobotX = -1.0 * RobotMovXEGTFSTurbo *0.015 * 1000.0;
        RobotY = RobotMovYEGTFSTurbo * 0.015 * 1000.0;
        RobotZ = -1.0 * RobotMovZEGTFSTurbo * 0.015 * 1000.0;
        //RobotX = RobotMovYEGTFSTurbo / 0.4 * 0.015 * 1000.0;
        //RobotY = -1.0 * RobotMovXEGTFSTurbo / 0.4 * 0.015 * 1000.0;
        //RobotZ = RobotMovZEGTFSTurbo /0.4 * 0.015 * 1000.0;
    }

    if (fabs(RobotX) > 1000.0)
    {
        if (RobotX > 0.0)
            RobotX = 1000.0;
        if (RobotX < 0.0)
            RobotX = -1000.0;
    }
    if (fabs(RobotY) > 1000.0)
    {
        if (RobotY > 0.0)
            RobotY = 1000.0;
        if (RobotY < 0.0)
            RobotY = -1000.0;
    }
    if (fabs(RobotZ) > 1000.0)
    {
        if (RobotZ > 0.0)
            RobotZ = 1000.0;
        if (RobotZ < 0.0)
            RobotZ = -1000.0;
    }
    // ** Get current Movement data and copy to Send Buffer
```

```
        nMoveData[0] = (short) RobotX; //pTurbo->m_nTx;
        nMoveData[1] = (short) RobotY; //pTurbo->m_nTy;
        nMoveData[2] = (short) RobotZ; //pTurbo->m_nTz;
        nMoveData[3] = (short) 0.0; //pTurbo->m_nRx;
        nMoveData[4] = (short) 0.0; //pTurbo->m_nRy;
        nMoveData[5] = (short) 0.0; //pTurbo->m_nRz;
        memcpy(chSendData, nMoveData, nNumSendChar);

        // ** Send Data
        pT->serial.SendData(chSendData, nNumSendChar);

        // ** Wait for a confirmation character ** //
        while (pT->serial.ReadDataWaiting() < 1)
                ;
        pT->serial.ReadData(chRecvData, 1);

        // ** If receiving data from MRC ** //
        if(nNumRecvChar > 0)
        {
            // ** Wait for data from MRC to arrive
            while (pT->serial.ReadDataWaiting() < nNumRecvChar)
            ;
            // ** Read the received data
            pT->serial.ReadData(chRecvData, nNumRecvChar);

            // ** Mem copy the data to required data type
            // ** For receiving XYZ Data - 3 x long (4 bytes) = 12 Bytes
            //memcpy(pTurbo->m_nXYZData, chRecvData, 12);
            // ** For receiving XYZ Data - 3 x short (2 bytes) = 6 Bytes
            //memcpy(pTurbo->m_nRotData, (chRecvData + 12), 6);
        }
        // ** Increment the Frequency counter
        //RobocCurX = PtRelToCS[0].x;
        //RobotCurY = PtRelToCS[0].y;
        //RobotCurZ = PtRelToCS[0].z;
        //sprintf(buffer, "%.5lf\t%.5lf\t%.5lf\t\n", RobotCurX, RobotCurY, RobotCurZ);
        //fprintf(FileGuidance, buffer);
        pT->m_uiTurboFreq++;
    }
    return 0;

id CTrackingClassDlg::OnRadioTurboCorr()

    // TODO: Add your control notification handler code here
    m_ctrlRadioTurbo.SetCheck(1);
    m_ctrlRadicMotocom.SetCheck(0);


id CTrackingClassDlg::OnRadioMotocomCorr()

    // TODO: Add your control notification handler code here
    m_ctrlRadioTurbo.SetCheck(0);
    m_ctrlRadioMotocom.SetCheck(1);


NT CTrackingClassDlg::DriveRobotThread(LPVOID pParam)

    double RobotMovX, RobotMovY, RobotMovZ;      //Robot increment to move to next control point
    char buffer[256];

    THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
    CTrackingClassDlg* pDR = (CTrackingClassDlg*) pThreadParams->lParam;
    delete pThreadParams;

    while (pDR->m_bDriveRobot)
    {
        if (CurrLine < NumOfLine)
        {
            if (CurrPt < NumOfPt[CurrLine])
            {
                RobotMovX = PointList[CurrLine][CurrPt].x;
                RobotMovY = PointList[CurrLine][CurrPt].y;
                RobotMovZ = PointList[CurrLine][CurrPt].z;

                double dPos[12] = {0,0,0,0,0,180,0,0,0,0,0,0};

                //Send points to robot controller
                //Send Move to Robot
```

```
              CString szMovType = "MOVL";
              CString szSpeedType = "V";
              double dMovSpeed = 10.0;
              CString szFrame = "UF5";
              WORD wForm = 0;
              int nTool = 1;

              dPos[0] = RobotMovX;
              dPos[1] = RobotMovY;
              dPos[2] = RobotMovZ;
              pDR->MyRobot->SetMode(PLAY);
              pDR->MyRobot->SetServo(ON);
              pDR->MyRobot->Move( szMovType, szSpeedType, dMovSpeed, szFrame, wForm, nTool, d
s);
              do
              {
                  pDR->MyRobot->UpdateStatus();
              }
              while (pDR->MyRobot->IsRobotOperating());
              pDR->MyRobot->GetCurrentPos(1, &pDR->pRobotCurrentPos);
              pDR->m_lfRobotX = pDR->pRobotCurrentPos.X;
              pDR->m_lfRobotY = pDR->pRobotCurrentPos.Y;
              pDR->m_lfRobotZ = pDR->pRobotCurrentPos.Z;
              ::Sleep(100);
              if (pDR->m_bDriveRobotWrite)
              {
                  ::sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\t\n", pDR->m_lf
botX, pDR->m_lfRobotY, pDR->m_lfRobotZ, PtRelToCS[0].x, PtRelToCS[0].y, PtRelToCS[0].z);
                  ::fprintf(FileRobotDr, buffer);
              }
              CurrPt++;
          }
          else
          {
              CurrPt = 0;
              CurrLine++;
          }
      }
      else
      {
          CurrPt = 0;
          CurrLine = 0;
          pDR->OnButtonDriveRobot();
      }
  }
  return 0;

id CTrackingClassDlg::OnButtonTurbotry()

  // TODO: Add your control notification handler code here
  if (!m_bTurboTryThread)
  {
      m_bTurboTryThread = true;
      THREADPARAMS* pThreadTurboTry = new THREADPARAMS;
      pThreadTurboTry->lParam = (LPARAM) this;
      g_pTurboTryThread = AfxBeginThread(TurboTryThread, pThreadTurboTry,
                                       THREAD_PRIORITY_NORMAL, NULL);
      m_ctrlTurboTry.SetWindowText("STOP TURBO");
  }
  else
  {
      m_bTurboTryThread = false;
      CloseHandle(g_pTurboTryThread->m_hThread);
      m_ctrlTurboTry.SetWindowText("TURBO TRY");

  }

id CTrackingClassDlg::OnRadioMotocomDrive()

  // TODO: Add your control notification handler code here
  m_bPathDrive = false;
  m_bMotocomDrive = true;
  m_ctrlRadioMotocomDrive.SetCheck(1);
  m_ctrlRadioPathDrive.SetCheck(0);
  m_ctrlDx.EnableWindow(1);
  m_ctrlDy.EnableWindow(1);
  m_ctrlDz.EnableWindow(1);
```

```
id CTrackingClassDlg::OnRadioPathDrive()

  // TODO: Add your control notification handler code here
  m_bPathDrive = true;
  m_bMotocomDrive = false;
  m_ctrlRadioMotocomDrive.SetCheck(0);
  m_ctrlRadioPathDrive.SetCheck(1);
  m_ctrlDx.EnableWindow(0);
  m_ctrlDy.EnableWindow(0);
  m_ctrlDz.EnableWindow(0);


id CTrackingClassDlg::OnChangeDx()

  // TODO: if this is a RICHEDIT control, the control will not
  // send this notification unless you override the CDialog::OnInitDialog()
  // function and call CRichEditCtrl().SetEventMask()
  // with the ENM_CHANGE flag ORed into the mask.

  // TODO: Add your control notification handler code here
  UpdateData(true);


id CTrackingClassDlg::OnChangeDy()

  // TODO: If this is a RICHEDIT control, the control will not
  // send this notification unless you override the CDialog::OnInitDialog()
  // function and call CRichEditCtrl().SetEventMask()
  // with the ENM_CHANGE flag ORed into the mask.

  // TODO: Add your control notification handler code here
  UpdateData(true);


id CTrackingClassDlg::OnChangeDz()

  // TODO: If this is a RICHEDIT control, the control will not
  // send this notification unless you override the CDialog::OnInitDialog()
  // function and call CRichEditCtrl().SetEventMask()
  // with the ENM_CHANGE flag ORed into the mask.

  // TODO: Add your control notification handler code here
  UpdateData(true);



id CTrackingClassDlg::OnSelchangeComboCs()

  // TODO: Add your control notification handler code here
      char buffer[256], temp[256];
  int j;
  switch (m_ctrlComboCS.GetCurSel())
  {
  case 0:
      for (j=0; j<3; j++)
      {
          for (int k=0; k<3; k++)
          {
              if (j==k)
              {
                  TransMatrix[j][k] = 1.0;
                  InvTMatrix[j][k] = 1.0;
              }
              else
              {
                  TransMatrix[j][k] = 0.0;
                  InvTMatrix[j][k] = 0.0;
              }
          }
      };
      break;
  case 1:
      for (j=0; j<3; j++)
      {
          sprintf(buffer, "Row%d", j);
          GetPrivateProfileString(buffer, "Col0", NullString, temp, (int)sizeof(temp), "Trans
trixLISM.ini");
```

```
          TransMatrix[j][0] = atof(temp);
          GetPrivateProfileString(buffer, "Col1", NullString, temp, (int)sizeof(temp), "Trans
trixLISM.ini");
          TransMatrix[j][1] = atof(temp);
          GetPrivateProfileString(buffer, "Col2", NullString, temp, (int)sizeof(temp), "Trans
trixLISM.ini");
          TransMatrix[j][2] = atof(temp);
          GetPrivateProfileString(buffer, "Col3", NullString, temp, (int)sizeof(temp), "Trans
trixLISM.ini");
          TransMatrix[j][3] = atof(temp);
      }
      for (j=0; j<3; j++)
      {
          sprintf(buffer, "InvRow%d", j);
          GetPrivateProfileString(buffer, "Col0", NullString, temp, (int)sizeof(temp), "Trans
trixLISM.ini");
          InvTMatrix[j][0] = atof(temp);
          GetPrivateProfileString(buffer, "Col1", NullString, temp, (int)sizeof(temp), "Trans
trixLISM.ini");
          InvTMatrix[j][1] = atof(temp);
          GetPrivateProfileString(buffer, "Col2", NullString, temp, (int)sizeof(temp), "Trans
trixLISM.ini");
          InvTMatrix[j][2] = atof(temp);
          GetPrivateProfileString(buffer, "Col3", NullString, temp, (int)sizeof(temp), "Trans
trixLISM.ini");
          InvTMatrix[j][3] = atof(temp);
      }
      break;
  }
  UpdateData(false);

id CTrackingClassDlg::OnButtonWriteFile()

  // TODO: Add your control notification handler code here
  if (!m_bDriveRobotWrite)
  {
      if ((FileRobotDr = fopen("RobotDrive.txt", "w")) == NULL)
      {
          puts("cannot open file");
          exit(1);
      }
      rewind(FileRobotDr);            // set the cursor to the beginning of file
      fprintf(FileRobotDr, "RobotCurMX\t RobotCurMY\t RobotCurMZ\tRobotCSX\tRobotCSY\tRobotCS
t\n");
      m_ctrlWriteFile.SetWindowText("Close File");
      m_bDriveRobotWrite = true;
  }
  else
  {
      fclose(FileRobotDr);
      m_ctrlWriteFile.SetWindowText("WriteToFile");
      m_bDriveRobotWrite = false;
  }

id CTrackingClassDlg::OnChangeEditVel()

  // TODO: If this is a RICHEDIT control, the control will not
  // send this notification unless you override the CDialog::OnInitDialog()
  // function and call CRichEditCtrl().SetEventMask()
  // with the ENM_CHANGE flag ORed into the mask.

  // TODO: Add your control notification handler code here
  UpdateData(true);


id CTrackingClassDlg::OnButtonDriveBeam()

  // TODO: Add your control notification handler code here
  DriveBeam();


id CTrackingClassDlg::OnButtonCorrect()

  // TODO: Add your control notification handler code here
  if (CurrLine < NumOfLine)
  {
        if (CurrPt < NumOfPt[CurrLine])
        {
            if (m_ctrlRadioMotocom.GetCheck())
                MotocomCorrection();
            else
                TurboCorrection();
            CurrPt++;
        }
        else
        {
            if (m_ctrlRadioMotocom.GetCheck())
                MotocomCorrection();
            else
                TurboCorrection();
            CurrPt = 0;
            CurrLine++;
        }
    }
    else
    {
        if (m_ctrlRadioMotocom.GetCheck())
                MotocomCorrection();
            else
                TurboCorrection();
        CurrPt = 0;
        CurrLine = 0;
    }

  id CTrackingClassDlg::OnButtonZero()

    // TODO: Add your control notification handler code here
    MotocomCorrection();


  id CTrackingClassDlg::kinematics(CMotor Mot1, CMotor Mot2, double psdx, double psdy, double l
  MATRIX *R)

    float a1, a2;//, a3;

    MATRIX *L1, *L2, *L0;
    MATRIX *M;
    MATRIX *A1, *A2, *Scratch1, *Scratch2;
    float q1, q2;

    q1 = Mot1.CurrentAngle * 2 * pi;
    q2 = Mot2.CurrentAngle * 2 * pi;
    L0 = matrix_allocate(4, 4);
    L1 = matrix_allocate(4, 4);
    L2 = matrix_allocate(4, 4);
    M = matrix_allocate(4, 4);
    A1 = matrix_allocate(4, 4);
    A2 = matrix_allocate(4, 4);
    Scratch1 = matrix_allocate(4, 4);
    Scratch2 = matrix_allocate(4, 4);

    transRPY(xL, yL, zL, alphaL, betaL, gammaL, L0);

    /*World CS to M1 Transform*/

    mirr(alpha1, beta1, zed1, A1);
    trans(xm1, ym1, zm1, A2);
    matrix_mult(A2, A1, M); // Mirror1 Transform

    refl(M, L0, A1);
    GETVAL_MAT(2,3,a1,A1);
    matrix_mult(L0, A1, L1);

    //M1 to M2 Transform
    mirr(alpha3, beta3, zed3, A1);
    rotx(q2, A2);
    matrix_mult(A2, A1, Scratch1);
    trans(dxm3, dym3, dzm3, A1);
    matrix_mult(A1, Scratch1, Scratch2);
    rotz(q1, A1);
    matrix_mult(A1, Scratch2, Scratch1);
    trans(xm2, ym2, zm2, A1);
    matrix_mult(A1, Scratch1, M); //Mirror2 Transform
  refl(M, L1, A1);
  GETVAL_MAT(2,3,a2,A1);
  matrix_mult(L1, A1, L2);

  //M2 to Retroreflector Transform
  trans(psdx, psdy, 1, A2);
  matrix_mult(L2, A2, R);


d CTrackingClassDlg::transRPY(float px, float py, float pz, float rx, float ry, float rz, MA
IX *TRPY)

  MATRIX *RX, *RY, *RZ, *T, *M1, *M2;

  M1 = matrix_allocate(4, 4);
  M2 = matrix_allocate(4, 4);
  RX = matrix_allocate(4, 4);
  RY = matrix_allocate(4, 4);
  RZ = matrix_allocate(4, 4);
  T = matrix_allocate(4, 4);

  rotx(rx, RX);
  roty(ry, RY);
  rotz(rz, RZ);
  trans(px, py, pz, T);
  matrix_mult(RY, RX, M1);
  matrix_mult(RZ, M1, M2);
  matrix_mult(T, M2, TRPY);

  matrix_free(M1);
  matrix_free(M2);
  matrix_free(RX);
  matrix_free(RY);
  matrix_free(RZ);
  matrix_free(T);


TRIX *CTrackingClassDlg::matrix_allocate(int rows, int cols)

  int i;
  MATRIX *A;

  A = (MATRIX*) calloc(1, sizeof(MATRIX));
  if (!A)
  {
      AfxMessageBox("Error in Matrix Allocation", IDOK, NULL);
      exit(1);
  }
  A->rows = rows;
  A->cols = cols;

  float **double_matrix;
  double_matrix = (float **) calloc(rows, sizeof(float *));
  for (i=0; i<rows; i++)
  {
      double_matrix[i] = (float*) calloc(cols, sizeof(float));
  }
  A->ptr = (char**)double_matrix;
  return A;


id CTrackingClassDlg::matrix_eye(MATRIX *I)

  unsigned int i, j;
  float **Iptr;

  Iptr = (float **)I->ptr;
  for (i=0; i<I->rows; i++)
  {
      for (j=0; j<I->cols; j++)
          Iptr[i][j] = (i==j) ? 1.0 : 0.0;
  }


id CTrackingClassDlg::rotx(float rx, MATRIX* RX)

  float **RXptr;
  RXptr = (float **)RX->ptr;
  matrix_eye(RX);
  RXptr[1][1] = cos(rx);
  RXptr[1][2] = -sin(rx);
  RXptr[2][1] = -RXptr[1][2];
  RXptr[2][2] = RXptr[1][1];


id CTrackingClassDlg::roty(float ry, MATRIX* RY)

  float **RYptr;
  RYptr = (float **)RY->ptr;
  matrix_eye(RY);
  RYptr[0][0] = cos(ry);
  RYptr[0][2] = sin(ry);
  RYptr[2][0] = -RYptr[0][2];
  RYptr[2][2] = RYptr[0][0];


id CTrackingClassDlg::rotz(float rz, MATRIX* RZ)

  float **RZptr;
  RZptr = (float **)RZ->ptr;
  matrix_eye(RZ);
  RZptr[0][0] = cos(rz);
  RZptr[0][1] = -sin(rz);
  RZptr[1][0] = -RZptr[0][1];
  RZptr[1][1] = RZptr[0][0];


id CTrackingClassDlg::trans(float dx, float dy, float dz, MATRIX* TXYZ)

  float **TXYZptr;

  TXYZptr = (float **)TXYZ->ptr;
  matrix_eye(TXYZ);

  TXYZptr[0][3] = dx;
  TXYZptr[1][3] = dy;
  TXYZptr[2][3] = dz;



id CTrackingClassDlg::matrix_free(MATRIX *A)

  unsigned int i;
  char **a;

  if (!A || !A->ptr || !A->rows || !A->cols)
  {
      AfxMessageBox("invalid matrix free", IDOK, 0);
      exit(1);
  }

  a = A->ptr;
  for (i=0; i<A->rows; i++)
      free(a[i]);
  free((char*)a);
  a=NULL;
  free((char *)A);


id CTrackingClassDlg::mirr(float rx, float ry, float dz, MATRIX *H)

  MATRIX *FA1, *FA2, *FScratch1;

  FA1 = matrix_allocate(4, 4);
  FA2 = matrix_allocate(4, 4);
  FScratch1 = matrix_allocate(4, 4);
  roty(ry, FA1);
  trans(0,0,dz,FA2);
  matrix_mult(FA1, FA2, FScratch1);
  rotx(rx, FA1);
  matrix_mult(FA1, FScratch1, H);
  matrix_free(FA1);
  matrix_free(FA2);
  matrix_free(FScratch1);


id CTrackingClassDlg::matrix_mult(MATRIX *A, MATRIX *B, MATRIX *C)
```

```
 int a_r, a_c, b_c;

 if (B->rows != A->cols)
 {
     AfxMessageBox("Error in Matrix Multiplication", IDOK, 0);
     exit(1);
 }

 a_r = A->rows;
 a_c = A->cols;
 b_c = B->cols;

 MULT_MAT(A, B, C, a_r, a_c, b_c);

d CTrackingClassDlg::refl(MATRIX *H, MATRIX *L, MATRIX *B)

 /* H - mirror CS */
 /* L - laser CS */
 /* B - new CS our from mirror*/

 float n, d, a, v, c, v_m, nLC1, nLC2;
 float **Hptr, **Lptr, **Bptr;

 Hptr = (float **) H->ptr;
 Lptr = (float **) L->ptr;
 Bptr = (float **) B->ptr;

 n = Hptr[0][3]*Hptr[0][2] + Hptr[1][3]*Hptr[1][2] + Hptr[2][3]*Hptr[2][2] - Hptr[0][2]*Lptr
[3] - Hptr[1][2]*Lptr[1][3] - Hptr[2][2]*Lptr[2][3];
 d = Hptr[0][2]*Lptr[0][2] + Hptr[1][2]*Lptr[1][2] + Hptr[2][2]*Lptr[2][2];

 a = n/d;
 nLC1 = Lptr[0][0]*Hptr[0][2] + Lptr[1][0]*Hptr[1][2] + Lptr[2][0]*Hptr[2][2];
 nLC2 = Lptr[0][1]*Hptr[0][2] + Lptr[1][1]*Hptr[1][2] + Lptr[2][1]*Hptr[2][2];

 v = 2*d*d;
 c = 1-v;

 v_m = v/(1-d*d);

 matrix_eye(B);

 Bptr[0][0] = nLC2*nLC2*v_m + c;
 Bptr[0][1] = -nLC2*nLC1*v_m;
 Bptr[0][2] = -nLC1*2*d;
 Bptr[1][0] = Bptr[0][1];
 Bptr[1][1] = nLC1*nLC1*v_m + c;
 Bptr[1][2] = -nLC2*2*d;
 Bptr[2][0] = -Bptr[0][2];
 Bptr[2][1] = -Bptr[1][2];
 Bptr[2][2] = c;
 Bptr[2][3] = a;

id CTrackingClassDlg::OnButtonTry()

 // TODO: Add your control notification handler code here
 MATRIX *R;//, *Q, *A;
 char buffer[256];
 float **Aptr;//, **Rptr, **Qptr;

 R = matrix_allocate(4,4);
 //Q = matrix_allocate(4,4);
 //A = matrix_allocate(4,4);
 /*matrix_eye(R);
 rotx(pi/2, R);
 matrix_eye(Q);
 roty(pi/2, Q);
 matrix_eye(A);
 matrix_mult(R, Q, A);
 Aptr = (float**)A->ptr;*/

 //Qptr = (float**)Q->ptr;
 M1.CurrentAngle = 0.0;
 M2.CurrentAngle = -0.034325;
 kinematics(M1, M2, 0.0, 0.0, 332.175, R);
 Aptr = (float**)R->ptr;
 sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t\n%.3lf\t%.3lf\t%.3lf\t%.3lf\t\n%.3lf\t%.3lf\t
lf\t%.3lf\t\n%.3lf\t%.3lf\t%.3lf\t%.3lf\t\n", Aptr[0][0], Aptr[0][1], Aptr[0][2], Aptr[0][3]
ptr[1][0], Aptr[1][1], Aptr[1][2], Aptr[1][3], Aptr[2][0], Aptr[2][1], Aptr[2][2], Aptr[2][3
Aptr[3][0], Aptr[3][1], Aptr[3][2], Aptr[3][3]);
 m_sStatus = buffer;
 sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\t", Aptr[0][3], Aptr[1][3]-465.13, Aptr[2][3]-548.37);
 m_sStatus += buffer;
 UpdateData(false);
 matrix_free(R);
 //matrix_free(Q);
 //matrix_free(A);


d CTrackingClassDlg::OnChangeEditAcc2()

 // TODO: If this is a RICHEDIT control, the control will not
 // send this notification unless you override the CDialog::OnInitDialog()
 // function and call CRichEditCtrl().SetEventMask()
 // with the ENM_CHANGE flag ORed into the mask.

 // TODO: Add your control notification handler code here
 UpdateData(true);


d CTrackingClassDlg::OnButtonPointSample()

 // TODO: Add your control notification handler code here
 if (!m_bSample)
 {
     m_bSample = true;
     m_ctrlPointSample.SetWindowText("STOP SAMPLE");
 }
 else
 {
     m_bSample = false;
     m_ctrlPointSample.SetWindowText("POINT SAMPLE");
 }


d CTrackingClassDlg::OnRadioMotocomCorr2()

 // TODO: Add your control notification handler code here
 m_ctrlRadioTurbo2.SetCheck(0);
 m_ctrlRadioMotocom2.SetCheck(1);


d CTrackingClassDlg::OnRadioTurboCorr2()

 // TODO: Add your control notification handler code here
 m_ctrlRadioTurbo2.SetCheck(1);
 m_ctrlRadioMotocom2.SetCheck(0);
```

```
TrackingClassDlg.h : header file

 !defined(AFX_TRACKINGCLASSDLG_H__403883F7_2D41_11D7_8A8B_000102C2E8E7__INCLUDED_)
fine AFX_TRACKINGCLASSDLG_H__403883F7_2D41_11D7_8A8B_000102C2E8E7__INCLUDED_

 _MSC_VER > 1000
agma once
dif // _MSC_VER > 1000

clude "ZygoClass.h"
clude "PSDClass.h"
clude "EncoderClass.h"
clude "RobotController.h"

edef struct

 CString name;
 double x;
 double y;
 double z;
EC_DATA_CARTESIAN;

edef struct

 //int element_size;
 unsigned int rows;
 unsigned int cols;
 char **ptr;
ATRIX;

fine MULT_MAT(a,b,c,rowsa,colsa,colsb) {\
 float **_AMX = (float **)a->ptr; \
 float **_BMX = (float **)b->ptr; \
 float **_CMX = (float **)c->ptr; \
 float *_PTA; \
 float *_PTB; \
 float *_PTC; \
 int _IX, _JX, _KX; \
 for (_IX=0; _IX<rowsa; _IX++) {\
     _PTC = _CMX[_IX]; \
     _PTB = _BMX[0]; \
     for (_JX = 0; _JX < colsb; _JX++) {\
         _PTA = _AMX[_IX]; \
         *_PTC = (*_PTA++) * (*_PTB++); \
         for (_KX = 1; _KX < colsa; _KX++) \
             *_PTC += (*_PTA++) * _BMX[_KX][_JX]; \
          _PTC++; \
     } \
 }\


efine GETVAL_MAT(row, col, value, A) { value = ((float**)A->ptr)[row][col];}

nst float xL = 0.0;
nst float yL = 0.0;
nst float zL = 162.0;
nst float alpha1 = (float)-2.3562;
nst float beta1 = 0.0;
nst float zed1 = 0.0;
nst float alpha3 = (float)0.7854;
nst float beta3 = 0.0;
nst float zed3 = 0.0;
nst float xm1 = 0.0;
nst float ym1 = (float)465.13;
nst float zm1 = 162.0;
nst float xm2 = 0.0;
nst float ym2 = (float)465.13;
nst float zm2 = (float)548.37;
nst float dxm3 = 0.0;
nst float dym3 = 0.0;
nst float dzm3 = 0.0;
nst float alphaL = (float)-1.5708;
nst float betaL = 0.0;
nst float gammaL = 0.0;

///////////////////////////////////////////////////////////////////////////
 CTrackingClassDlg dialog
```

```
ss CTrackingClassDlg : public CDialog

Construction
lic:
 CTrackingClassDlg(CWnd* pParent = NULL);     // standard constructor
 void Transformation();
 void DriveMotor(CMotor, CMotor);
 bool m_bTrack;
 bool m_bInitFlag;
 bool m_bSubSystemRun;
 bool m_bDataThread;
 bool m_bFileFlag;
 bool m_bSample;
 bool m_bDynamicSample;
 bool m_bTurboTryThread;
 bool m_bUpdateTurbo;


 FILE* fp;
 CEncoder Encoder;
 CPSD Psd;
 CZygo Zygo;
 CBitmap hImageStart, hImageStop;
 CMotor M1, M2;

 CMotor Motor1, Motor2;
 bool m_bEncoderThread;
 bool m_bPsdThread;
 bool m_bZygoThread;

 bool m_bGPIBorRS232;

 CString m_sTime;
 CRobotController* MyRobot;
 CSerial serial;
 static UINT TurboTryThread(LPVOID);

 bool m_bTx, m_bTy, m_bTz;
 bool m_bGuide;
 bool m_bEGTFS;
 bool m_bEGTFSTurbo;
 bool m_bDriveRobot;
 bool m_bMotocomDrive;
 bool m_bPathDrive;
 bool m_bDriveRobotWrite;
 bool m_bTurboCorrThread;
 float m_fAzimuthTarget;
 float m_fElevationTarget;
 static UINT GuidanceThread(LPVOID);
 static UINT EGTFSThread(LPVOID);
 static UINT EGTFSTurboThread(LPVOID);
 static UINT DriveRobotThread(LPVOID);
 static UINT TurboCorrectionThread(LPVOID);
 void DriveBeam();
 void MotocomCorrection();
 void TurboCorrection();
 RobotPos pRobotCurrentPos;

 void kinematics(CMotor, CMotor, double, double, double, MATRIX*);
 void rotx(float, MATRIX*);
 void roty(float, MATRIX*);
 void rotz(float, MATRIX*);
 void mirr(float, float, float, MATRIX*);
 void trans(float, float, float, MATRIX*);
 void matrix_eye(MATRIX*);
 void refl (MATRIX*, MATRIX*, MATRIX*);
 MATRIX *matrix_allocate(int, int);
 void matrix_free(MATRIX *);
 void matrix_mult(MATRIX*, MATRIX*, MATRIX*);
 void transRPY(float, float, float, float, float, float, MATRIX*);
Dialog Data
 //{{AFX_DATA(CTrackingClassDlg)
 enum { IDD = IDD_TRACKINGCLASS_DIALOG };
 CButton m_ctrlPointSample;
 CButton m_ctrlRadioTurbo2;
 CButton m_ctrlRadioMotocom2;
 CButton m_ctrlWriteFile;
 CEdit   m_ctrlDz;
 CEdit   m_ctrlDy;
```

```
    CEdit   m_ctrlDx;
    CButton m_ctrlRadioPathDrive;
    CButton m_ctrlRadioMotocomDrive;
    CButton m_ctrlTurboTry;
    CButton m_ctrlRadioTurbo;
    CButton m_ctrlRadioMotocom;
    CButton m_ctrlButtonGuidance;
    CButton m_ctrlButtonEGTFS;
    CComboBox    m_ctrlComboCS;
    CButton m_ctrlDriveRobot;
    CButton m_ctrlOpenPort;
    CButton m_ctrlDynamicSample;
    CButton m_ctrlSample;
    CButton m_ctrlRadioCAT;
    CButton m_ctrlRadioAPR;
    CButton m_ctrlGoBB;
    CButton m_ctrlTrack;
    CButton m_ctrlCheckDisplay;
    CButton m_ctrlRadioRS232;
    CButton m_ctrlRadioGPIB;
    CButton m_ctrlUp;
    CButton m_ctrlInit;
    CEdit   m_ctrlStepSize;
    CButton m_ctrlSubSystem;
    CButton m_ctrlDecStepSize;
    CButton m_ctrlIncStepSize;
    CButton m_ctrlRight;
    CButton m_ctrlLeft;
    CButton m_ctrlDown;
    CString m_sStatus;
    UINT    m_uiPSDFreq;
    UINT    m_uiENCFreq;
    int     m_uiZYGOFreq;
    long    m_lEnc1;
    long    m_lEnc2;
    double  m_lfPSDX;
    double  m_lfPSDY;
    double  m_lfLaserDist;
    UINT    m_uiStepSize;
    CString m_sBeam;
    UINT    m_uiCompThreadFreq;
    double  m_lfPosX;
    double  m_lfPosY;
    double  m_lfPosZ;
    double  m_lfPosX2;
    double  m_lfPosY2;
    double  m_lfPosZ2;
    UINT    m_uiTurboFreq;
    UINT    m_uiEGTFSFreq;
    UINT    m_uiGuidanceFreq;
    double  m_lfVel;
    double  m_lfRobotX;
    double  m_lfRobotY;
    double  m_lfRobotZ;
    double  m_lfTurboTryAcc;
    double  m_lfTurboTryVel;
    float   m_fDx;
    float   m_fDy;
    float   m_fDz;
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTrackingClassDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

   Implementation
otected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CTrackingClassDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnButtonInit();
    afx_msg void OnTimer(UINT nIDEvent);
```

```
  afx_msg void OnButtonSubsystem();
  afx_msg void OnButtonUp();
  afx_msg void OnButtonDown();
  afx_msg void OnButtonLeft();
  afx_msg void OnButtonRight();
  afx_msg void OnChangeEditStepSize();
  afx_msg void OnBUTTONReINITZYGO();
  afx_msg void OnRadioGpib();
  afx_msg void OnRadioRs232();
  virtual void OnOK();
  afx_msg void OnButtonTrack();
  afx_msg void OnButtonSetBb();
  afx_msg void OnButtonGoBb();
  afx_msg void OnRadioApr();
  afx_msg void OnRadioCat();
  afx_msg void OnButtonStaticSample();
  afx_msg void OnButtonDynamicSample();
  afx_msg void OnButtonOpenPort();
  afx_msg void OnButtonDriveRobot();
  afx_msg void OnCheckTx();
  afx_msg void OnCheckTy();
  afx_msg void OnCheckTz();
  afx_msg void OnButtonReset();
  afx_msg void OnChangeEditAcc();
  afx_msg void OnButtonTransformation();
  afx_msg void OnButtonPathGeneration();
  afx_msg void OnButtonEgtfs();
  afx_msg void OnButtonGuidance();
  afx_msg void OnRadioTurboCorr();
  afx_msg void OnRadioMotocomCorr();
  afx_msg void OnButtonTurbotry();
  afx_msg void OnRadioMotocomDrive();
  afx_msg void OnRadioPathDrive();
  afx_msg void OnChangeDx();
  afx_msg void OnChangeDy();
  afx_msg void OnChangeDz();
  afx_msg void OnSelchangeComboCs();
  afx_msg void OnButtonWriteFile();
  afx_msg void OnChangeEditVel();
  afx_msg void OnButtonDriveBeam();
  afx_msg void OnButtonCorrect();
  afx_msg void OnButtonZero();
  afx_msg void OnButtonTry();
  afx_msg void OnChangeEditAcc2();
  afx_msg void OnButtonPointSample();
  afx_msg void OnRadioMotocomCorr2();
  afx_msg void OnRadioTurboCorr2();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
ivate:
  static UINT ComputationThread(LPVOID);

{{AFX_INSERT_LOCATION}}
 Microsoft Visual C++ will insert additional declarations immediately before the previous lin

ndif // !defined(AFX_TRACKINGCLASSDLG_H__403883F7_2D41_11D7_8A8B_000102C2E8E7__INCLUDED_)
```

```
TrackingClass.cpp : Defines the class behaviors for the application.


clude "stdafx.h"
clude "TrackingClass.h"
clude "TrackingClassDlg.h"

def _DEBUG
fine new DEBUG_NEW
def THIS_FILE
tic char THIS_FILE[] = __FILE__;
dif

///////////////////////////////////////////////////////////////////////
 CTrackingClassApp

IN_MESSAGE_MAP(CTrackingClassApp, CWinApp)
 //{{AFX_MSG_MAP(CTrackingClassApp)
     // NOTE - the ClassWizard will add and remove mapping macros here.
     //    DO NOT EDIT what you see in these blocks of generated code!
 //}}AFX_MSG
 ON_COMMAND(ID_HELP, CWinApp::OnHelp)
_MESSAGE_MAP()

///////////////////////////////////////////////////////////////////////
 CTrackingClassApp construction

ackingClassApp::CTrackingClassApp()

 // TODO: add construction code here,
 // Place all significant initialization in InitInstance


///////////////////////////////////////////////////////////////////////
 The one and only CTrackingClassApp object

ackingClassApp theApp;

///////////////////////////////////////////////////////////////////////
 CTrackingClassApp initialization

L CTrackingClassApp::InitInstance()

 AfxEnableControlContainer();

 // Standard initialization
 // If you are not using these features and wish to reduce the size
 //  of your final executable, you should remove from the following
 //  the specific initialization routines you do not need.

def _AFXDLL
 Enable3dControls();          // Call this when using MFC in a shared DLL
se
 Enable3dControlsStatic();    // Call this when linking to MFC statically
dif

 CTrackingClassDlg dlg;
 m_pMainWnd = &dlg;
 int nResponse = dlg.DoModal();
 if (nResponse == IDOK)
 {
     // TODO: Place code here to handle when the dialog is
     //  dismissed with OK
 }
 else if (nResponse == IDCANCEL)
 {
     // TODO: Place code here to handle when the dialog is
     //  dismissed with Cancel
 }

 // Since the dialog has been closed, return FALSE so that we exit the
 //  application, rather than start the application's message pump.
 return FALSE;
```

```
TrackingClass.h : main header file for the TRACKINGCLASS application

 !defined(AFX_TRACKINGCLASS_H__403883F5_2D41_11D7_8A8B_000102C2E8E7__INCLUDED_)
fine AFX_TRACKINGCLASS_H__403883F5_2D41_11D7_8A8B_000102C2E8E7__INCLUDED_

 _MSC_VER > 1000
agma once
dif // _MSC_VER > 1000

ndef __AFXWIN_H__
 #error include 'stdafx.h' before including this file for PCH
dif

clude "resource.h"       // main symbols
clude "serial.h"
///////////////////////////////////////////////////////////////////////////
CTrackingClassApp:
See TrackingClass.cpp for the implementation of this class


ss CTrackingClassApp : public CWinApp

lic:
 CTrackingClassApp();

Overrides
 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CTrackingClassApp)
 public:
 virtual BOOL InitInstance();
 //}}AFX_VIRTUAL

Implementation

 //{{AFX_MSG(CTrackingClassApp)
     // NOTE - the ClassWizard will add and remove member functions here.
     //    DO NOT EDIT what you see in these blocks of generated code !
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()



///////////////////////////////////////////////////////////////////////////

{{AFX_INSERT_LOCATION}}
 Microsoft Visual C++ will insert additional declarations immediately before the previous lin


ndif // !defined(AFX_TRACKINGCLASS_H__403883F5_2D41_11D7_8A8B_000102C2E8E7__INCLUDED_)
```

```
ZygoClass.cpp: implementation of the CZygo class.

//////////////////////////////////////////////////////////////////////

nclude "stdafx.h"
nclude "ZygoClass.h"
nclude "SSClass.h"
nclude "decl-32.h"
nclude "acrolib.h"

fdef _DEBUG
ndef THIS_FILE
atic char THIS_FILE[]=__FILE__;
efine new DEBUG_NEW
ndif

ort Zmi1000;
ort GpibBoard;
 Device = 0;
tern unsigned long pACRAddress[1];
tern bool m_bAPRorCAT;
//////////////////////////////////////////////////////////////////////
 Construction/Destruction
//////////////////////////////////////////////////////////////////////

ygo::CZygo()

  // Initialise Laser parameters
  m_uiLaserFreq = 0;
  if (m_bAPRorCAT)
      LaserDist = 332.175; //323.994; /* 332.175 for APR on poles
                                      /* 323.994 for APR on Leica adapter*/
  else
      LaserDist = 317.79;
  hCommPort = NULL;
  m_bThread = false;
  g_pZygoThread = NULL;


ygo::~CZygo()

  if (g_pZygoThread)
  {
      m_bThread = false;
      CloseHandle(g_pZygoThread->m_hThread);
  }


id CZygo::InitialiseLaser(bool ZygoInterface, CString* Status)

  char buffer[256];
  char Wbuffer[80];
  char Lbuffer[80];
  char portcmd[256];

  m_bGPIB_RS232 = ZygoInterface;
  //LaserDist = BBLaserDist;
  if (m_bAPRorCAT)
      LaserDist = 332.175; //323.994;
  else
      LaserDist = 317.79;
  if (m_bGPIB_RS232)
  {
      Device = ibfind("DEV1");
      sprintf(buffer, "Zygo descriptor = %d\n",Device);
      m_sStatus = buffer;
      GpibBoard = ibfind("GPIB0");
      sprintf(buffer, "GPIB board descriptor = %d\n",GpibBoard);
      m_sStatus.Insert(m_sStatus.GetLength(), buffer);
      ibonl(Device, 1);
      SendIFC(0);
      ibconfig(Device, IbcEOSrd, 1);
      ibclr(Device);
      ibwrt(Device, "FORMAT A\n", 9);
      ibwrt(Device, "*ESE 60\n", 8);
      ibwrt(Device, "*SRE 32\n", 8);
      ibwrt(Device, "*SRE 1\n", 7);
      ibtmo(Device, T10s);
      ibtmo(GpibBoard, T10s);
      ibwrt(Device, "*IDN?\n", 6);
      ibrd(Device, ZygoID, 80);
      ibwrt(Device, "*OPT?\n", 5);
      ibrd(Device, ZygoOp, 80);
      ibwrt(Device, "WGET? #H2D,#H100\n", 17);
      ibrd(Device, Wbuffer, 80);
      ibwrt(Device, "LGET? #H2D,#H10C\n", 17);
      ibrd(Device, Lbuffer, 80);
      ibwrt(Device, "WPUT #H2D,#H104,#H2\n", 20);
      ibwrt(Device, "WGET? #H2D,#H100\n", 17);
      ibrd(Device, Wbuffer, 80);
      ibwrt(Device, "LGET? #H2D,#H10C\n", 17);
      ibrd(Device, Lbuffer, 80);
  }
  else
  {
      if (hCommPort != NULL)
      {
          ClosePort();
          OpenPort();
      }
      else
          OpenPort();
      sprintf(portcmd, "*CLS\n");
      WriteToPort(portcmd, strlen(portcmd)).
      sprintf(portcmd, "*RST\n");
      WriteToPort(portcmd, strlen(portcmd));
      sprintf(portcmd, "FORMAT A\n");
      WriteToPort(portcmd, strlen(portcmd));
      sprintf(portcmd, "*ESE 60\n");
      WriteToPort(portcmd, strlen(portcmd));
      sprintf(portcmd, "*SRE 32\n");
      WriteToPort(portcmd, strlen(portcmd));
      sprintf(portcmd, "*SRE 1\n");
      WriteToPort(portcmd, strlen(portcmd));
      sprintf(buffer, "RS_232 COM PORT 1 \n");
      m_sStatus = buffer;
  }
  *Status = m_sStatus;


NDLE CZygo::OpenPort()

  DCB dcb;
  char lpPort[10]="COM1";
  hCommPort = CreateFile(lpPort,
                           GENERIC_READ | GENERIC_WRITE,
                           0,
                           NULL,
                           OPEN_EXISTING,
                           NULL,
                           NULL);

  if (hCommPort == INVALID_HANDLE_VALUE)
  {
      AfxMessageBox("Cannot open port.", MB_OK);
      return 0;
  }
  FillMemory(&dcb, sizeof(dcb), 0);
  dcb.DCBlength=sizeof(dcb);
  if (!BuildCommDCB("19200, n, 8, 1", &dcb))
  {
      AfxMessageBox("Cannot build dcb.", MB_OK);
      return 0;
  }
  GetCommState(hCommPort, &dcb);
  dcb.fOutxCtsFlow = 0;
  dcb.fOutxDsrFlow = 0;
  //dcb.BaudRate = CBR_9600;
  //dcb.ByteSize = 8;
  //dcb.Parity = NOPARITY;
  //dcb.StopBits = ONESTOPBIT;
  SetCommState(hCommPort, &dcb);
  return (hCommPort);


oid CZygo::ClosePort()

  CloseHandle(hCommPort);
```

```
t CZygo::ReadFromPort(char *lpReadBuf, DWORD BytesToRead)

  DWORD dwRead;
  //char *lpReadBuf;

  if (!ReadFile(hCommPort, lpReadBuf, BytesToRead, &dwRead, NULL))
  {
      AfxMessageBox("ReadFile ERROR", MB_OK);
      return 0;
  }
  else
  {
      //AfxMessageBox("ReadFile OK", ML_OK);
      return (dwRead);
  }


t CZygo::WriteToPort(char *lpWriteBuf, DWORD BytesToWrite)

  DWORD dwWrite;
  //char *lpWriteBuf;

  if (!WriteFile(hCommPort, lpWriteBuf, BytesToWrite, &dwWrite, NULL))
  {
      AfxMessageBox("WriteFile Error", MB_OK);
      return 0;
  }
  else
  {
      //AfxMessageBox("WriteFile OK", MB_OK);
      return (dwWrite);
  }


NT CZygo::SampleZygoThread(LPVOID pParam)

  THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
  CZygo* pZygoWnd = (CZygo*) pThreadParams->lParam;
  delete pThreadParams;

  char portcmd[256];
  char portdata[256] = {0};
  double ZValue;
  unsigned long byteread;
  char *pPortData;
  char Lbuffer[80];
  long L;
  unsigned long L2;
  char* pChar;
  DWORD dwErrorFlags;
  float pSendIEEEZygo[1];

  //InitializeCriticalSection(&ZygoCS);
  while (pZygoWnd->m_bThread)
  {
      //GPIB Sampling
      if (pZygoWnd->m_bGPIB_RS232)
      {
          ibwrt(Device, "LGET? #H2D,#H10C\n", 17);
          ibrd(Device, Lbuffer, 80);
          Lbuffer[0] = '0';
          Lbuffer[1] = 'x';
          if (Lbuffer[2] >= '8')
          {
              L2 = strtoul(Lbuffer, &pChar, 16);
              L = L2 - 2147483648 - 2147483648;
          }
          else
          {
              L = strtol(Lbuffer, &pChar, 16);
          }
      }
      //RS232 Sampling
      else
      {

          // Send command to read Interferometer data
```

```
            ::sprintf(portcmd, "READ? 1,M\n");
            pZygoWnd->WriteToPort(portcmd, strlen(portcmd));
            //ClearCommError(pZygoWnd->hCommPort, &dwErrorFlags, &pZygoWnd->ComStat);
            //byteread = pZygoWnd->ComStat.cbInQue;
            //pZygoWnd->ReadFromPort(Lbuffer, byteread);

            // Use pointer to point to the array of char, improve efficiency
            pPortData = portdata;
            // Start polling until receiving character 10
            while (true)
            {
                byteread += pZygoWnd->ReadFromPort(pPortData, 1);
                if (*pPortData == 10)
                    break;
                if (byteread)
                    pPortData++;
            }
            L = atoi(portdata);

        }
        //Data Conversion
        ZValue = L * Resolution * 1000;
        if (m_bAPRorCAT)
            pZygoWnd->LaserDist = BBLaserDistAPR + ZValue;
        else
            pZygoWnd->LaserDist = BBLaserDistCAT + ZValue;
        if (pZygoWnd->LaserDist < 0.0)
        {
            AcroSendString("JOG OFF X Y", 0);
            AcroSendString("HALT",0);
            return 0;
        }
        pSendIEEEZygo[0] = (float) pZygoWnd->LaserDist;
        A8_BIN_POKE_IEEE(0x01, 1, pACRAddress[0] + 5, pSendIEEEZygo, 0);

        byteread = 0;
        pZygoWnd->m_uiLaserFreq++;
    }
    return 0;

  id CZygo::StartThread()

    m_bThread = true;
    THREADPARAMS* pZygoThread = new THREADPARAMS;
    pZygoThread->lParam = (LPARAM) this;
    g_pZygoThread = AfxBeginThread(SampleZygoThread, pZygoThread, THREAD_PRIORITY_HIGHEST, NULL


  id CZygo::StopThread()

    m_bThread = false;
    CloseHandle(g_pZygoThread->m_hThread);


  uble CZygo::GetSample()

    return LaserDist;


  NT CZygo::GetFreq()

    UINT temp;
    temp = m_uiLaserFreq;
    m_uiLaserFreq = 0;
    return temp;
```

```
ZygoClass.h: interface for the CZygo class.

///////////////////////////////////////////////////////////////////////

f !defined(AFX_ZYGOCLASS_H__FB37866F_DF2E_11D6_8A3A_000102C2E8E7__INCLUDED_)
efine AFX_ZYGOCLASS_H__FB37866F_DF2E_11D6_8A3A_000102C2E8E7__INCLUDED_

f _MSC_VER > 1000
ragma once
ndif // _MSC_VER > 1000

atic CWinThread*   g_pZygoThread;

nst double N = 1.000271296;
nst double Lambda = 0.000000632991528;
nst double Resolution = Lambda / (N * 256);
nst double BBLaserDistAPR = 332.175;;
nst double BBLaserDistCAT = 317.79;// 333.8995;


ass CZygo

ivate:
  unsigned int m_uiLaserFreq;
  int WriteToPort(char *lpReadBuf, DWORD BytesToRead);
  int ReadFromPort(char *lpReadBuf, DWORD BytesToRead);
  void ClosePort();
  HANDLE OpenPort();

  HANDLE hCommPort;
  COMSTAT ComStat;
  BYTE bPort, bByteSize, bParity, bStopBits;
  DWORD dwBaudRate;
  COMMTIMEOUTS timeout;
  static UINT SampleZygoThread(LPVOID);

  //double InterferometerDist;
  double LaserDist;
  //double BBLaserDist;
  //unsigned int freq;
  double time;
  bool m_bThread;

  char ZygoID[80];
  char ZygoOp[80];
blic:
  CString m_sStatus;
  bool m_bGPIB_RS232;
  UINT GetFreq();
  double GetSample();
  void StopThread();
  void StartThread();
  void InitialiseLaser(bool, CString*);

  CZygo();
  virtual ~CZygo();

  //double GetSample();
  //void StartThread();
  //void StopThread();


ndif // !defined(AFX_ZYGOCLASS_H__FB37866F_DF2E_11D6_8A3A_000102C2E8E7__INCLUDED_)
```

```
PSDClass.cpp: implementation of the CPSD class.

//////////////////////////////////////////////////////////////////////

nclude "stdafx.h"
nclude "PSDClass.h"
nclude "SSClass.h"
nclude "acrolib.h"
nclude "math.h"

fdef _DEBUG
ndef THIS_FILE
atic char THIS_FILE[]=__FILE__;
efine new DEBUG_NEW
ndif

tern unsigned long pACRAddress[1];
//////////////////////////////////////////////////////////////////////
 Construction/Destruction
//////////////////////////////////////////////////////////////////////

SD::CPSD()

  // Initialise PSD parameters
  m_uiPSDFreq = 0;
  x_PSD = 0.0;
  y_PSD = 0.0;
  channel = 12;

  psd = new PSDStruct;
  psd->x = 0.0;
  psd->y = 0.0;

  p_Begin = psd;
  p_Current = psd;
  Mov_Count = 1;
  x_MovAvg = 0.0;
  y_MovAvg = 0.0;

  m_bThread = false;
  g_pPSDSvThread = NULL;



SD::~CPSD()

  if (g_pPSDSvThread)
  {
      m_bThread = false;
      CloseHandle(g_pPSDSvThread->m_hThread);
  }


OL CALLBACK GetDriver(LPSTR lpszName, LPSTR lpszEntry, LPARAM lParam)

  LPBOARD lpboard = (LPBOARD)(LPVOID)lParam;
  /* fill in board strings */
  lstrcpyn(lpboard->name,lpszName,STRLEN);
  lstrcpyn(lpboard->entry,lpszEntry,STRLEN);
  /* try to open board */
  lpboard->status = olDaInitialize(lpszName,(LPHDEV)&lpboard->hdrvr);
  if   (lpboard->hdrvr != NULL)
      return FALSE;           /* false to stop enumerating */
  else
      return TRUE;            /* true to continue          */


ol CPSD::InitialisePSDSv(CString* Status)


is is a callback function of olDaEnumBoards, it gets the strings of the Open the board.
 successful, enumeration is halted.
yers board and attempts to initialize

  char buffer[80];

  /* Get first available Open Layers board */
  board.hdrvr = NULL;
  CHECKERROR (olDaEnumBoards(GetDriver,(LPARAM)(LPBOARD)&board));
  /* check for error within callback function */
  CHECKERROR (board.status);

  /* check for NULL driver handle-means no boards */
  if (board.hdrvr == NULL)
  {
      AfxMessageBox("No Open Layer boards!!!",
          MB_ICONEXCLAMATION | MB_OK);
      return ((UINT) NULL);
  }

  /* get handle to ADC sub system */
  CHECKERROR (olDaGetDASS((HDEV)board.hdrvr,OLSS_AD,0,&board.hdass));

  /*Set subsystem for single value operation */
  CHECKERROR (olDaSetDataFlow(board.hdass, OL_DF_SINGLEVALUE));
  CHECKERROR (olDaSetChannelType(board.hdass, OL_CHNT_SINGLEENDED));
  olDaSetRange(board.hdass,10.0,-10.0);
  olDaSetEncoding(board.hdass, OL_ENC_BINARY);
  CHECKERROR (olDaConfig(board.hdass));

  /* get sub system information for code/volts conversion */
  olDaGetRange(board.hdass,&max,&min);
  olDaGetEncoding(board.hdass,&encoding);
  olDaGetResolution(board.hdass,&resolution);
  sprintf(buffer, "PSD Initialised\n");
  *Status = buffer;
  return true;



id CPSD::StartThread()

  m_bThread = true;
  THREADPARAMS* pPSDThread = new THREADPARAMS;
  pPSDThread->lParam = (LPARAM) this;
  g_pPSDSvThread = AfxBeginThread(SamplePSDSvThread, pPSDThread, THREAD_PRIORITY_NORMAL, NULL



id CPSD::StopThread()

  m_bThread = false;
  CloseHandle(g_pPSDSvThread->m_hThread);


NT CPSD::SamplePSDSvThread(LPVOID pParam)

  THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
  CPSD* pPSDWnd = (CPSD*) pThreadParams->lParam;
  delete pThreadParams;

  float volts15, volts14, volts13, volts12;
  //float volts7, volts6, volts5, volts4;
  //float volts3, volts2, volts1, volts0;
  long value15, value14, value13, value12;
  //long value7, value6, value5, value4;
  //long value3, value2, value1, value0;
  double RatioX, RatioY;
  float DevXy, DevYx;
  float pSendIEEEPSD[2];

  while (pPSDWnd->m_bThread)
  {
      /* get single value */
      olDaGetSingleValue(board.hdass,&value15, 15, gain);
      olDaGetSingleValue(board.hdass,&value14, 14, gain);
      olDaGetSingleValue(board.hdass,&value13, 13, gain);
      olDaGetSingleValue(board.hdass,&value12, 12, gain);

      //PSD 2 on Gimbal
      /*olDaGetSingleValue(board.hdass,&value3, 3, gain);
      olDaGetSingleValue(board.hdass,&value2, 2, gain);
      olDaGetSingleValue(board.hdass,&value1, 1, gain);
      olDaGetSingleValue(board.hdass,&value0, 0, gain);

      //PSD 3 on Gimbal
      olDaGetSingleValue(board.hdass,&value7, 7, gain);
```

```
    olDaGetSingleValue(board.hdass,&value6, 6, gain);
    olDaGetSingleValue(board.hdass,&value5, 5, gain);
    olDaGetSingleValue(board.hdass,&value4, 4, gain);
    */

    //volts1 = (10.0)/(4096) * value1 + 0.01;
    //volts2 = (10.0)/(4096) * value2 + 0.01;
    //volts3 = (10.0)/(4096) * value3 + 0.01;
    //volts4 = (10.0)/(4096) * value4 + 0.01;

    //With interference filter
    volts15 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value15
(float)pPSDWnd->min +0.234-0.257+0.0695;
    volts14 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value14
(float)pPSDWnd->min -0.299-0.205+0.0444;
    volts13 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value13
(float)pPSDWnd->min -0.098-0.203+0.0260;
    volts12 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value12
(float)pPSDWnd->min +0.081-0.266+0.0583;
    /*
    //PSD 2 on Gimbal
    volts7 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value7 +
float)pPSDWnd->min +0.234-0.257+0.0695;
    volts6 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value6 +
float)pPSDWnd->min -0.299-0.205+0.0444;
    volts5 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value5 +
float)pPSDWnd->min -0.098-0.203+0.0260;
    volts4 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value4 +
float)pPSDWnd->min +0.081-0.266+0.0583;

    //PSD 3 on Gimbal
    volts3 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value3 +
float)pPSDWnd->min +0.234-0.257+0.0695;
    volts2 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value2 +
float)pPSDWnd->min -0.299-0.205+0.0444;
    volts1 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value1 +
float)pPSDWnd->min -0.098-0.203+0.0260;
    volts0 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value0 +
float)pPSDWnd->min +0.081-0.266+0.0583;
    */

    if (pPSDWnd->Mov_Count < 5)                  //Moving Average period 5
    {
        //Lateral Effect Detector
        if ((volts13 == 0.0) && (volts15 == 0.0))
        {
            RatioX = 0.0;
        }
        else
        {
            RatioX = ((volts13 - volts15) / (volts13 + volts15));
        }

        if ((volts12 == 0.0) && (volts14 == 0.0))
        {
            RatioY = 0.0;
        }
        else
        {
            RatioY = ((volts14 - volts12) / (volts14 + volts12));
        }

        //Second Order using Corner Cube
        if (RatioX >=0.0)
            pPSDWnd->p_Current->x = 1.519*RatioX*RatioX + 7.4715*RatioX;
        else
            pPSDWnd->p_Current->x = -1.2825*RatioX*RatioX + 7.3906*RatioX;
        if (RatioY >=0.0)
            pPSDWnd->p_Current->y = 1.7685*RatioY*RatioY + 7.3441*RatioY;
        else
            pPSDWnd->p_Current->y = -1.3512*RatioY*RatioY + 7.834*RatioY;

        pPSDWnd->x_MovAvg += pPSDWnd->p_Current->x;
        pPSDWnd->y_MovAvg += pPSDWnd->p_Current->y;
        pPSDWnd->p_Current->next = new PSDStruct;
        pPSDWnd->p_Current = pPSDWnd->p_Current->next;
        pPSDWnd->Mov_Count++;
    }
    else
```

```
        {
            if ((volts13 == 0.0) && (volts15 == 0.0))
            {
                RatioX = 0.0;
            }
            else
            {
                RatioX = ((volts13 - volts15) / (volts13 + volts15));
            }

            if ((volts12 == 0.0) && (volts14 == 0.0))
            {
                RatioY = 0.0;
            }
            else
            {
                RatioY = ((volts14 - volts12) / (volts14 + volts12));
            }

            //Second Order using Corner Cube
            if (RatioX >=0.0)
                pPSDWnd->p_Current->x = 1.519*RatioX*RatioX + 7.4715*RatioX;
            else
                pPSDWnd->p_Current->x = -1.2825*RatioX*RatioX + 7.3906*RatioX;
            if (RatioY >=0.0)
                pPSDWnd->p_Current->y = 1.7685*RatioY*RatioY + 7.3441*RatioY;
            else
                pPSDWnd->p_Current->y = -1.3512*RatioY*RatioY + 7.834*RatioY;


            pPSDWnd->x_MovAvg += pPSDWnd->p_Current->x;
            pPSDWnd->y_MovAvg += pPSDWnd->p_Current->y;
            pPSDWnd->x_PSD = pPSDWnd->x_MovAvg / pPSDWnd->Mov_Count;
            pPSDWnd->y_PSD = pPSDWnd->y_MovAvg / pPSDWnd->Mov_Count;

            DevXy = -0.0021 * pPSDWnd->y_PSD * pPSDWnd->y_PSD - 0.0117 * pPSDWnd->y_PSD;
            DevYx = 0.0081 * pPSDWnd->x_PSD * pPSDWnd->x_PSD + 0.0219 * pPSDWnd->x_PSD;

            pPSDWnd->x_PSD -= DevXy;
            pPSDWnd->y_PSD -= DevYx;

            /*if (fabs(pPSDWnd->x_PSD) < 0.1)
                pPSDWnd->x_PSD = 0.0;
            if (fabs(pPSDWnd->y_PSD) < 0.1)
                pPSDWnd->y_PSD = 0.0;*/
            pSendIEEEPSD[0] = pPSDWnd->x_PSD;
            pSendIEEEPSD[1] = pPSDWnd->y_PSD;
            A8_BIN_POKE_IEEE(0x01, 2, pACRAddress[0] + 1, pSendIEEEPSD, 0);

            pPSDWnd->x_MovAvg -= pPSDWnd->p_Begin->x;
            pPSDWnd->y_MovAvg -= pPSDWnd->p_Begin->y;
            pPSDWnd->p_delete = pPSDWnd->p_Begin;
            pPSDWnd->p_Begin = pPSDWnd->p_Begin->next;
            delete pPSDWnd->p_delete;
            pPSDWnd->p_Current->next = new PSDStruct;
            pPSDWnd->p_Current = pPSDWnd->p_Current->next;
        }
        //x_PSD = ((V4 - V2) / (V4 + V2)) * 7.3874;
        //y_PSD = ((V3 - V1) / (V3 + V1)) * 6.4005;
        //pPSDWnd->channel = 0;
        pPSDWnd->m_uiPSDFreq++;
        //::Sleep(1);
    }
    //Sleep(1);

    return 0;

id CPSD::GetSample(double *pX, double *pY)

    *pX = x_PSD;
    *pY = y_PSD;


NT CPSD::GetFreq()

    UINT temp;
    temp = m_uiPSDFreq;
    m_uiPSDFreq = 0;
```

```
return temp;
```

```
PSDClass.h: interface for the CPSD class.

//////////////////////////////////////////////////////////////////////

 !defined(AFX_PSDCLASS_H__FB378680_DF2E_11D6_8A3A_000102C2E8E7__INCLUDED_)
fine AFX_PSDCLASS_H__FB378680_DF2E_11D6_8A3A_000102C2E8E7__INCLUDED_

 _MSC_VER > 1000
agma once
dif // _MSC_VER > 1000
clude <olmem.h>
clude <olerrors.h>
clude <oldaapi.h>


Error handling macros */
fine STRLEN 80          /* string size for general text manipulation   */
tic char str[STRLEN];          /* global string for general text manipulation */

fine SHOW_ERROR(ecode) AfxMessageBox(olDaGetErrorString(ecode,\
               str,STRLEN), MB_ICONEXCLAMATION | MB_OK);

fine CHECKERROR(ecode) if ((board.status = (ecode)) != OLNOERROR)\
               {\
               SHOW_ERROR(board.status);\
               olDaReleaseDASS(board.hdass);\
               olDaTerminate((HDEV)board.hdrvr);\
               return ((UINT)NULL);}

fine CLOSEONERROR(ecode) if ((board.status = (ecode)) != OLNOERROR)\
               {\
               SHOW_ERROR(board.status);\
               olDaReleaseDASS(board.hdass);\
               olDaTerminate((HDEV)board.hdrvr);\
               EndDialog(hDlg, TRUE);\
               return (TRUE);}

fine CLOSEONERROR1(ecode, h) if ((board.status = (ecode)) != OLNOERROR)\
               {\
               SHOW_ERROR(board.status);\
               olDaReleaseDASS(board.hdass);\
               olDaTerminate((HDEV)board.hdrvr);\
               EndDialog(h, TRUE);\
               return (TRUE);}
fine CHECKERROR1(ecode) if ((board.status = (ecode)) != OLNOERROR)\
               {\
               SHOW_ERROR(board.status);\
               olDaReleaseDASS(board.hdass);\
                 olDaTerminate((HDEV)board.hdrvr);\
                 return ((UINT)NULL);}

pedef struct tag_Board

  HDEV hdrvr;          /* driver handle               */
  HDASS hdass;         /* sub system handle           */
  ECODE status;        /* board error status          */
  HBUF  hbuf;          /* sub system buffer handle */
  WORD FAR* lpbuf;     /* buffer pointer              */
  char name[STRLEN];   /* string for board name       */
  char entry[STRLEN];  /* string for board name       */
OARD;

pedef BOARD FAR* LPBOARD;
atic BOARD board;

nst DBL gain = 1.0;

atic CWinThread*   g_pPSDSvThread;

ass PSDStruct

blic:
  double x;
  double y;
  PSDStruct* next;


ass CPSD
```

```
lic:
  UINT GetFreq();
  void GetSample(double*, double*);
  CPSD();
  virtual ~CPSD();
  void StopThread();
  void StartThread();
  bool InitialisePSDSv(CString*);

vate:
  unsigned int m_uiPSDFreq;
  float m_fV1;
  float m_fV2;
  float m_fV3;
  float m_fV4;
  static UINT SamplePSDSvThread(LPVOID);
  bool m_bThread;
  DBL min, max;
  UINT encoding, resolution;
  UINT channel;
  PSDStruct* psd;
  PSDStruct* p_Begin;
  PSDStruct* p_Current;
  PSDStruct* p_delete;
  double x_MovAvg, y_MovAvg;
  double x_PSD, y_PSD;
  int Mov_Count;
  double temp_x, temp_y;
  //BOOL CALLBACK GetDriver(LPSTR , LPSTR , LPARAM );


ndif // !defined(AFX_PSDCLASS_H__FB378680_DF2E_11D6_8A3A_000102C2E8E7__INCLUDED_)
```

```
EncoderClass.cpp: implementation of the CEncoder class.

///////////////////////////////////////////////////////////////////

clude "stdafx.h"
clude "EncoderClass.h"
clude "acrolib.h"
clude "SSClass.h"

def _DEBUG
def THIS_FILE
tic char THIS_FILE[]=__FILE__;
fine new DEBUG_NEW
dif

///////////////////////////////////////////////////////////////////
Construction/Destruction
///////////////////////////////////////////////////////////////////

coder::CEncoder()

 Motor1.enc = 0;
 Motor1.bb_enc = 0;
 Motor1.CurrentAngle = 0.0;

 Motor2.enc = 0;
 Motor2.bb_enc = 0;
 Motor2.CurrentAngle = 0.0;

 m_uiEncFreq = 0;
 g_pMotorStatusThread = NULL;
 m_bThread = false;


coder::~CEncoder()

 if (g_pMotorStatusThread)
 {
     m_bThread = false;
     CloseHandle(g_pMotorStatusThread->m_hThread);
 }


id CEncoder::InitialiseMotor(CString* Status, unsigned long* pACRAdd, unsigned long* pACRAdd2


 char buffer[80];

 AcroInitialize(0);
 if (AcroGetError()!=ACRO_SUCCESS)
 {
     MessageBeep(0);
     AfxMessageBox("Card Driver Not Started", MB_ICONEXCLAMATION);
     exit(0);
 }
 A8_BIN_ADD ESS(0, 0x00, pACRAdd, 0);
 A8_BIN_ADDRESS(0, 0x01, pACRAdd2, 0);
 sprintf(buffer, "%ld\t%ld\n", pACRAdd, pACRAdd2);
 *Status = buffer;



id CEncoder::StartThread()

 m_bThread = true;
 THREADPARAMS* pMotorStatusThread = new THREADPARAMS;
 pMotorStatusThread->lParam = (LPARAM) this;
 g_pMotorStatusThread = AfxBeginThread(ReadMotorStatusThread, pMotorStatusThread, THREAD_PRI
ITY_NORMAL, NULL);


id CEncoder::StopThread()

 m_bThread = false;
 CloseHandle(g_pMotorStatusThread->m_hThread);


NT CEncoder::ReadMotorStatusThread(LPVOID pParam)
```

```
THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
CEncoder* pMotorStatusWnd = (CEncoder*) pThreadParams->lParam;
delete pThreadParams;
long pPosParameter[2];

while (pMotorStatusWnd->m_bThread)
{
    A8_BIN_GROUP_GETLONG(0x18, 0x00, 0x03, pPosParameter, 0);
    pMotorStatusWnd->Motor1.enc = pPosParameter[0];
    pMotorStatusWnd->Motor2.enc = -1 * pPosParameter[1];
    pMotorStatusWnd->Motor1.CurrentAngle = (float(pMotorStatusWnd->Motor1.enc) /614400.0);
 //Angle in Revolution
    pMotorStatusWnd->Motor2.CurrentAngle = (float(pMotorStatusWnd->Motor2.enc) /40000.0);
 //Angle in Revolution
    pMotorStatusWnd->m_uiEncFreq++;
    //::Sleep(10);
}
return 0;


d CEncoder::GetSample(CMotor *M1, CMotor *M2)

 *M1 = Motor1;
 *M2 = Motor2;


T CEncoder::GetFreq()

 UINT Temp;
 Temp = m_uiEncFreq;
 m_uiEncFreq = 0;
 return Temp;
```

```
EncoderClass.h: interface for the CEncoder class.

//////////////////////////////////////////////////////////////////////

 !defined(AFX_ENCODERCLASS_H__C893BC80_E00F_11D6_8A3A_000102C2E8E7__INCLUDED_)
fine AFX_ENCODERCLASS_H__C893BC80_E00F_11D6_8A3A_000102C2E8E7__INCLUDED_

 _MSC_VER > 1000
agma once
dif // _MSC_VER > 1000

tic CWinThread*   g_pMotorStatusThread;

ss CMotor

lic:
 double DriveAngle;
 double CurrentAngle;
 long enc;
 long bb_enc;


ss CEncoder

lic:
 //unsigned long pAddress[3];
 UINT GetFreq();
 void GetSample(CMotor*, CMotor*);
 void StopThread();
 void StartThread();
 void InitialiseMotor(CString*, unsigned long*, unsigned long*);

 CEncoder();
 virtual ~CEncoder();

vate:
 UINT m_uiEncFreq;
 static UINT ReadMotorStatusThread(LPVOID);
 bool m_bThread;
 CMotor Motor1;
 CMotor Motor2;


dif // !defined(AFX_ENCODERCLASS_H__C893BC80_E00F_11D6_8A3A_^00102C2E8E7__INCLUDED_)
```

```
Path.cpp : implementation file


nclude "stdafx.h"
nclude "TrackingClass.h"
nclude "Path.h"
nclude "TrackingClassDlg.h"
nclude "math.h"

fdef _DEBUG
efine new DEBUG_NEW
ndef THIS_FILE
atic char THIS_FILE[] = __FILE__;
ndif

tern int SelPathType;
extern REC_DATA_CARTESIAN StartPoint[1];
extern REC_DATA_CARTESIAN LastPoint[1];
tern REC_DATA_CARTESIAN PointList[Max_Line][Max_ViaPoint];
tern int NumOfPt[4];
tern int NumOfLine;
t LineNum;
tern FILE* FilePath;
tern REC_DATA_CARTESIAN PtRelToCS[1];
tern float aveVel;
//////////////////////////////////////////////////////////////////////////////
 CPath dialog


ath::CPath(CWnd* pParent /*=NULL*/)
  : CDialog(CPath::IDD, pParent)

  //{{AFX_DATA_INIT(CPath)
  m_sStaticViaPtDistNum = _T("");
  m_sViaPtDistNum = _T("");
  m_fPathStartX = 0.0f;
  m_fPathStartY = 0.0f;
  m_fPathStartZ = 0.0f;
  m_fPathLastX = 0.0f;
  m_fPathLastY = 0.0f;
  m_fPathLastZ = 0.0f;
  m_fVel = 0.0f;
  //}}AFX_DATA_INIT



id CPath::DoDataExchange(CDataExchange* pDX)

  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CPath)
  DDX_Control(pDX, IDC_RADIO_VIA_PT_VEL, m_ctrlRadioViaPtVel);
  DDX_Control(pDX, IDC_COMBO_LINE_NUMBER, m_ctrlComboLineNum);
  DDX_Control(pDX, IDC_RADIO_VIA_PT_NUM, m_ctrlRadioViaPtNum);
  DDX_Control(pDX, IDC_RADIO_VIA_PT_DIST, m_ctrlRadioViaPtDist);
  DDX_Control(pDX, IDC_STATIC_LP, m_ctrlLastPtText);
  DDX_Control(pDX, IDC_LIST_VIA_POINTS, m_ctrlListViaPoints);
  DDX_Control(pDX, IDC_COMBO_PATHTYPE, m_ctrlComboPathType);
  DDX_Control(pDX, IDC_EDIT_VIA_PT_DISTNUM, m_ctrlViaPtDistNum);
  DDX_Control(pDX, IDC_BUTTON_GENERATE_VIA_POINTS, m_ctrlGenerateViaPt);
  DDX_Control(pDX, IDC_EDIT_PATH_RADIUS, m_ctrlPathRadius);
  DDX_Control(pDX, IDC_STATIC_PATH_RADIUS, m_ctrlStaticPathRadius);
  DDX_Control(pDX, IDC_BUTTON_FILE_BROWSE, m_ctrlPathFileBrowse);
  DDX_Text(pDX, IDC_STATIC_VIA_PT_DISTNUM, m_sStaticViaPtDistNum);
  DDX_Text(pDX, IDC_EDIT_VIA_PT_DISTNUM, m_sViaPtDistNum);
  DDX_Text(pDX, IDC_EDIT_PATH_START_X, m_fPathStartX);
  DDX_Text(pDX, IDC_EDIT_PATH_START_Y, m_fPathStartY);
  DDX_Text(pDX, IDC_EDIT_PATH_START_Z, m_fPathStartZ);
  DDX_Text(pDX, IDC_EDIT_PATH_LAST_X, m_fPathLastX);
  DDX_Text(pDX, IDC_EDIT_PATH_LAST_Y, m_fPathLastY);
  DDX_Text(pDX, IDC_EDIT_PATH_LAST_Z, m_fPathLastZ);
  DDX_Text(pDX, IDC_EDIT_PATH_VEL, m_fVel);
  //}}AFX_DATA_MAP



GIN_MESSAGE_MAP(CPath, CDialog)
  //{{AFX_MSG_MAP(CPath)
  ON_CBN_SELCHANGE(IDC_COMBO_PATHTYPE, OnSelchangeComboPathtype)
  ON_BN_CLICKED(IDC_RADIO_VIA_PT_NUM, OnRadioViaPtNum)
 ON_BN_CLICKED(IDC_RADIO_VIA_PT_DIST, OnRadioViaPtDist)
 ON_BN_CLICKED(IDC_BUTTON_GENERATE_VIA_POINTS, OnButtonGenerateViaPoints)
 ON_CBN_SELCHANGE(IDC_COMBO_LINE_NUMBER, OnSelchangeComboLineNumber)
 ON_BN_CLICKED(IDC_BUTTON_GET_CUR_POS, OnButtonGetCurPos)
 ON_BN_CLICKED(IDC_RADIO_VIA_PT_VEL, OnRadioViaPtVel)
 //}}AFX_MSG_MAP
_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
CPath message handlers

d CPath::OnSelchangeComboPathtype()

 // TODO: Add your control notification handler code here
 switch (m_ctrlComboPathType.GetCurSel())
 {
 case 0:
     m_ctrlPathFileBrowse.EnableWindow(0);
     m_ctrlPathRadius.ShowWindow(0);
     m_ctrlStaticPathRadius.ShowWindow(0);
     m_ctrlRadioViaPtNum.EnableWindow(1);
     m_ctrlRadioViaPtDist.EnableWindow(1);
     m_ctrlGenerateViaPt.EnableWindow(1);
     m_ctrlViaPtDistNum.EnableWindow(1);
     break;
 case 1:
     m_ctrlPathFileBrowse.EnableWindow(0);
     m_ctrlPathRadius.ShowWindow(1);
     m_ctrlStaticPathRadius.ShowWindow(1);
     m_ctrlRadioViaPtNum.EnableWindow(1);
     m_ctrlRadioViaPtDist.EnableWindow(1);
     m_ctrlGenerateViaPt.EnableWindow(1);
     m_ctrlViaPtDistNum.EnableWindow(1);
     break;
 case 2:
     m_ctrlPathFileBrowse.EnableWindow(1);
     m_ctrlPathRadius.ShowWindow(0);
     m_ctrlStaticPathRadius.ShowWindow(0);
     m_ctrlRadioViaPtNum.EnableWindow(0);
     m_ctrlRadioViaPtDist.EnableWindow(0);
     m_ctrlGenerateViaPt.EnableWindow(0);
     m_ctrlViaPtDistNum.EnableWindow(0);
     break;
 }


OL CPath::OnInitDialog()

 CDialog::OnInitDialog();

 // TODO: Add extra initialization here
 m_ctrlRadioViaPtNum.SetCheck(1);
 m_ctrlRadioViaPtDist.SetCheck(0);
 m_sStaticViaPtDistNum = "No. of Via Points";
 LineNum = 1;

 m_ctrlComboPathType.SetCurSel(SelPathType);
 if (SelPathType == 2)
 {
     m_ctrlPathFileBrowse.EnableWindow(1);
     m_ctrlPathRadius.ShowWindow(1);
     m_ctrlStaticPathRadius.ShowWindow(1);
     m_ctrlRadioViaPtNum.EnableWindow(0);
     m_ctrlRadioViaPtDist.EnableWindow(0);
     m_ctrlGenerateViaPt.EnableWindow(0);
     m_ctrlViaPtDistNum.EnableWindow(0);
 }
 else
 {
     m_ctrlPathFileBrowse.EnableWindow(0);
     m_ctrlPathRadius.ShowWindow(0);
     m_ctrlStaticPathRadius.ShowWindow(0);
     m_ctrlRadioViaPtNum.EnableWindow(1);
     m_ctrlRadioViaPtDist.EnableWindow(1);
     m_ctrlGenerateViaPt.EnableWindow(1);
     m_ctrlViaPtDistNum.EnableWindow(1);
 }

 UpdateData(false);
```

```
  return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE


d CPath::OnRadioViaPtNum()

  // TODO: Add your control notification handler code here
  m_ctrlRadioViaPtVel.SetCheck(0);
  m_ctrlRadioViaPtNum.SetCheck(1);
  m_ctrlRadioViaPtDist.SetCheck(0);
  m_sStaticViaPtDistNum = "No. of Via Points";
  UpdateData(false);


d CPath::OnRadioViaPtDist()

  // TODO: Add your control notification handler code here
  m_ctrlRadioViaPtVel.SetCheck(0);
  m_ctrlRadioViaPtNum.SetCheck(0);
  m_ctrlRadioViaPtDist.SetCheck(1);
  m_sStaticViaPtDistNum = "Via Points Distance Interval";
  UpdateData(false);


d CPath::OnOK()

  // TODO: Add extra validation here
  SelPathType = m_ctrlComboPathType.GetCurSel();
  CDialog::OnOK();


d CPath::OnButtonGenerateViaPoints()

  // TODO: Add your control notification handler code here
  double xInterval, yInterval, zInterval;
  char buffer[256];
  double TotalDist;
  float FacY, FacZ;
  double DeltaX, DeltaY, DeltaZ;
  int Num, Den;
  float remainder;

  UpdateData(true);

  if (m_ctrlRadioViaPtNum.GetCheck())
  {
      NumOfPt[LineNum-1] = atoi(m_sViaPtDistNum) + 2;
      PointList[LineNum-1][0].x = m_fPathStartX;
      PointList[LineNum-1][0].y = m_fPathStartY;
      PointList[LineNum-1][0].z = m_fPathStartZ;
      PointList[LineNum-1][NumOfPt[LineNum-1]-1].x = m_fPathLastX;
      PointList[LineNum-1][NumOfPt[LineNum-1]-1].y = m_fPathLastY;
      PointList[LineNum-1][NumOfPt[LineNum-1]-1].z = m_fPathLastZ;
      xInterval = sqrt(pow((m_fPathStartX - m_fPathLastX),2)) / (atof(m_sViaPtDistNum) + 1.0)

      yInterval = sqrt(pow((m_fPathStartY - m_fPathLastY),2)) / (atof(m_sViaPtDistNum) + 1.0)

      zInterval = sqrt(pow((m_fPathStartZ - m_fPathLastZ),2)) / (atof(m_sViaPtDistNum) + 1.0)


      for (int i = 0; i < NumOfPt[LineNum-1]; i++)
      {
          sprintf(buffer, "%.3lf\t%.3lf\t%.3lf", PointList[LineNum-1][i].x, PointList[LineNum
][i].y, PointList[LineNum-1][i].z);
          m_ctrlListViaPoints.AddString(buffer);
          PointList[LineNum-1][i+1].x = PointList[LineNum-1][i].x + xInterval;
          PointList[LineNum-1][i+1].y = PointList[LineNum-1][i].y + yInterval;
          PointList[LineNum-1][i+1].z = PointList[LineNum-1][i].z + zInterval;
      }
      //Get distance btw start and end points
      //divide distance by no of points, get interval
      //square interval, divide by 3, square root again-> get interval for individual x y and

      //interative add intervals to points
  }

  if (m_ctrlRadioViaPtDist.GetCheck() || m_ctrlRadioViaPtVel.GetCheck())
  {
      //square interval, divide by 3, square root again-> get interval for individual x y and
```

```
    //interative add intervals to points
    //Check if current point greater than end point, if yes->last point is end point
    //  if not-> keep adding
    if (m_ctrlRadioViaPtVel.GetCheck())
    {
        m_sViaPtDistNum.Format("%.3f", m_fVel * 0.4);
        UpdateData(false);
        aveVel = m_fVel;
    }

    TotalDist = sqrt(pow((m_fPathStartX - m_fPathLastX),2) + pow((m_fPathStartY - m_fPathLa
),2) + pow((m_fPathStartZ - m_fPathLastZ),2));
    Num = TotalDist * 1000.0;
    Den = atof(m_sViaPtDistNum) * 1000.0;
    remainder = Num % Den;
    if (remainder == 0)
        NumOfPt[LineNum-1] = TotalDist / atof(m_sViaPtDistNum) + 1;
    else
        NumOfPt[LineNum-1] = TotalDist / atof(m_sViaPtDistNum) + 2;
    PointList[LineNum-1][0].x = m_fPathStartX;
    PointList[LineNum-1][0].y = m_fPathStartY;
    PointList[LineNum-1][0].z = m_fPathStartZ;
    PointList[LineNum-1][NumOfPt[LineNum-1]-1].x = m_fPathLastX;
    PointList[LineNum-1][NumOfPt[LineNum-1]-1].y = m_fPathLastY;
    PointList[LineNum-1][NumOfPt[LineNum-1]-1].z = m_fPathLastZ;
    DeltaX = m_fPathLastX - m_fPathStartX;
    DeltaY = m_fPathLastY - m_fPathStartY;
    DeltaZ = m_fPathLastZ - m_fPathStartZ;
    if (DeltaX !=0.0)
    {
        FacY = fabs(DeltaY / DeltaX);
        FacZ = fabs(DeltaZ / DeltaX);
        if (DeltaX >= 0.0)
            xInterval = sqrt(pow(atof(m_sViaPtDistNum),2) / (pow(FacY, 2) + pow(FacZ, 2) +
0));
        else
            xInterval = -1.0 * sqrt(pow(atof(m_sViaPtDistNum),2) / (pow(FacY, 2) + pow(FacZ
2) + 1.0));
        if (DeltaY >= 0.0)
            yInterval = FacY * fabs(xInterval);
        else
            yInterval = -1.0 * FacY * fabs(xInterval);
        if (DeltaZ >= 0.0)
            zInterval = FacZ * fabs(xInterval);
        else
            zInterval = -1.0 * FacZ * fabs(xInterval);
    }
    else if (DeltaY == 0)
    {
        //FacY = 0.0;
        //FacZ = 1.0;
        xInterval = 0.0;
        yInterval = 0.0;
        if (DeltaZ >=0.0)
            zInterval = atof(m_sViaPtDistNum);
        else
            zInterval = -1.0 * atof(m_sViaPtDistNum);
    }
    else
    {
        xInterval = 0.0;
        if (DeltaY >=0.0)
            yInterval = atof(m_sViaPtDistNum);
        else
            yInterval = -1.0 * atof(m_sViaPtDistNum);
        zInterval = 0.0;
    }

    sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\n", PointList[LineNum-1][0].x, PointList[LineNum-1
0].y, PointList[LineNum-1][0].z);
    m_ctrlListViaPoints.AddString(buffer);
    fprintf(FilePath, buffer);
    for (int i = 0; i < NumOfPt[LineNum-1]-2; i++)
    {
        PointList[LineNum-1][i+1].x = PointList[LineNum-1][i].x + xInterval;
        PointList[LineNum-1][i+1].y = PointList[LineNum-1][i].y + yInterval;
        PointList[LineNum-1][i+1].z = PointList[LineNum-1][i].z + zInterval;
        sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\n", PointList[LineNum-1][i+1].x, PointList[Lin
m-1][i+1].y, PointList[LineNum-1][i+1].z);
            m_ctrlListViaPoints.AddString(buffer);
            fprintf(FilePath, buffer);
        }
        sprintf(buffer, "%.3lf\t%.3lf\t%.3lf\n", PointList[LineNum-1][NumOfPt[LineNum-1]-1].x,
ntList[LineNum-1][NumOfPt[LineNum-1]-1].y, PointList[LineNum-1][NumOfPt[LineNum-1]-1].z);
        m_ctrlListViaPoints.AddString(buffer);
        fprintf(FilePath, buffer);
    }

    if (LineNum < NumOfLine)
    {
        m_fPathStartX = PointList[LineNum-1][NumOfPt[LineNum-1] - 1].x;
        m_fPathStartY = PointList[LineNum-1][NumOfPt[LineNum-1] - 1].y;
        m_fPathStartZ = PointList[LineNum-1][NumOfPt[LineNum-1] - 1].z;
        LineNum ++;
        sprintf(buffer, "Line %d !! -> Enter the last point and press GENERATE", LineNum);
        m_ctrlLastPtText.SetWindowText(buffer);
    }
    else
        m_ctrlGenerateViaPt.EnableWindow(0);
    UpdateData(false);


d CPath::OnSelchangeComboLineNumber()

    // TODO: Add your control notification handler code here
    switch (m_ctrlComboLineNum.GetCurSel())
    {
    case 0:
        NumOfLine = 1;
        break;
    case 1:
        NumOfLine = 2;
        break;
    case 2:
        NumOfLine = 3;
        break;
    case 3:
        NumOfLine = 4;
        break;
    }


d CPath::OnButtonGetCurPos()

    // TODO: Add your control notification handler code here
    m_fPathStartX = PtRelToCS[0].x;
    m_fPathStartY = PtRelToCS[0].y;
    m_fPathStartZ = PtRelToCS[0].z;
    UpdateData(false);


d CPath::OnRadioViaPtVel()

    // TODO: Add your control notification handler code here
    m_ctrlRadioViaPtNum.SetCheck(0);
    m_ctrlRadioViaPtDist.SetCheck(0);
    m_ctrlRadioViaPtVel.SetCheck(1);
    m_sStaticViaPtDistNum = "Via Points Distance Interval";
    UpdateData(false);
```

```
Path.h : header file

 !defined(AFX_PATH_H__D2F54A54_2367_45AC_B876_88EA34C58E7D__INCLUDED_)
fine AFX_PATH_H__D2F54A54_2367_45AC_B876_88EA34C58E7D__INCLUDED_

 _MSC_VER > 1000
agma once
dif // _MSC_VER > 1000

//////////////////////////////////////////////////////////////////////////////
CPath dialog

ss CPath : public CDialog

Construction
lic:
 CPath(CWnd* pParent = NULL);   // standard constructor

Dialog Data
 //{{AFX_DATA(CPath)
 enum { IDD = IDD_DIALOG_PATH_GENERATION };
 CButton m_ctrlRadioViaPtVel;
 CComboBox   m_ctrlComboLineNum;
 CButton m_ctrlRadioViaPtNum;
 CButton m_ctrlRadioViaPtDist;
 CStatic m_ctrlLastPtText;
 CListBox    m_ctrlListViaPoints;
 CComboBox   m_ctrlComboPathType;
 CEdit   m_ctrlViaPtDistNum;
 CButton m_ctrlGenerateViaPt;
 CEdit   m_ctrlPathRadius;
 CStatic m_ctrlStaticPathRadius;
 CButton m_ctrlPathFileBrowse;
 CString m_sStaticViaPtDistNum;
 CString m_sViaPtDistNum;
 float   m_fPathStartX;
 float   m_fPathStartY;
 float   m_fPathStartZ;
 float   m_fPathLastX;
 float   m_fPathLastY;
 float   m_fPathLastZ;
 float   m_fVel;
 //}}AFX_DATA


Overrides
 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CPath)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

Implementation
tected:

 // Generated message map functions
 //{{AFX_MSG(CPath)
 afx_msg void OnSelchangeComboPathtype();
 virtual BOOL OnInitDialog();
 afx_msg void OnRadioViaPtNum();
 afx_msg void OnRadioViaPtDist();
 virtual void OnOK();
 afx_msg void OnButtonGenerateViaPoints();
 afx_msg void OnSelchangeComboLineNumber();
 afx_msg void OnButtonGetCurPos();
 afx_msg void OnRadioViaPtVel();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()


{{AFX_INSERT_LOCATION}}
Microsoft Visual C++ will insert additional declarations immediately before the previous lin


dif // !defined(AFX_PATH_H__D2F54A54_2367_45AC_B876_88EA34C58E7D__INCLUDED_)
```

```
RobotController.cpp: implementation of the CRobotController class.

//////////////////////////////////////////////////////////////////////

clude "stdafx.h"
clude "RobotController.h"
clude "motocom.h"
clude <direct.h>

def _DEBUG
def THIS_FILE
tic char THIS_FILE[]=__FILE__;
fine new DEBUG_NEW
dif

//////////////////////////////////////////////////////////////////////
Construction/Destruction
//////////////////////////////////////////////////////////////////////

botController::CRobotController(int ControllerType, LPCTSTR FilePath) :
 m_nControllerType(ControllerType)

 // ** Setup intial default parameters
 if( FilePath == "" )
 {
     _getcwd( m_szFilePath.GetBuffer(100), 100 );
     m_szFilePath.ReleaseBuffer();
 }

 m_nCommType = RS232;
 m_nCommPort = 2;
 m_nBaudRate = 9600;
 m_nParity = 2;
 m_nDataLength = 8;
 m_nStopBits = 0;

 m_szIPAddress = "196.168.10.10";

 m_nControlGroup = m_nMode = m_nCycle =
     m_nHold = m_nErrorCode = m_nAlarmCode = 0;

 m_bCommOpen = m_bConnected = m_bServoOn = m_bAlarmed =
     m_bErrored = m_bRemoteOn = m_bOperating = m_bSafeOperating = FALSE;

 // ** Set SK120 Robot's joints PPR
 long lRobotPPR[12] = {669570, 564000, 564000, 473140, 469330, 285120, 1,1,1,1,1,1};
 SetRobotPPR(lRobotPPR);

 m_szCurrentJobName = "";



botController::CRobotController()




botController::~CRobotController()

 CloseComm();



 CRobotController::SendFile(CString szFileName)


 int rc = -1;

 // File name check
 if( szFileName.IsEmpty() || (szFileName.GetLength() > 12) )
 {
     AfxMessageBox ( "Error with File Name - Empty or too long!");
     return -1;
 }

 // Make Job Name UpperCase
 szFileName.MakeUpper();

 // Connect to Controller
```

```
if( Connect() != 1)
    return -1;

if( m_bConnected )
{
    // Download File
    char* pchTemp = (char *)(LPCTSTR)szFileName;

    rc = BscDownLoad( m_nCommID, pchTemp);
    Disconnect();

    // Check return value
    if( rc == 0 )
    {
        AfxMessageBox ( "File Sent Successfully" );
        return rc;
    }
    else
    {
        AfxMessageBox ( "Error Sending File");
        return -1;
    }
}

return rc;


t CRobotController::ReceiveFile(CString szFileName)


int rc = -1;


// File name check
if( szFileName.IsEmpty() || (szFileName.GetLength() > 12) )
{
    AfxMessageBox ( "Error with File Name - Empty or too long!");
    return -1;
}

//CString szTemp;

//szTemp = szFileName.Right(4);
//szTemp = szFileName.Right(3);
//szTemp = szFileName.Insert(szFileName.GetLength(), ".JBI");

// Add ".jbi" if required
if( (szFileName.Right(4) != ".jbi") || (szFileName.Right(4) != ".JBI"))
    //szFileName.Insert(szFileName.GetLength(), ".JBI");
    szFileName += ".JBI";

// Connect to Controller
if( Connect() != 1)
    return -1;

if( m_bConnected )
{
    // Upload File
    rc = BscUpLoad( m_nCommID, (char *)(LPCTSTR)szFileName);
    Disconnect();
    // Check return value
    if( rc == 0 )
    {
        AfxMessageBox ( "File Received Successfully" );
        return rc;
    }
    else
    {
        AfxMessageBox ( "Error Receiving File");
        return -1;
    }
}

return rc;


t CRobotController::OpenComm()
```

```
// *******************
/*parity    Parity  0: None, 1: Odd, 2: Even
Clen    Data length  7: 7 bits, 8: 8 bits
Stp Stop bit  0: 1 bit, 1: 1.5 bits, 2: 2 bits
*/

int rc = -1;

if(m_bCommOpen) return 1;

if( m_nCommType == RS232 )
{
    // Open the COM port

    m_nCommID = 3scOpen( (char *)(LPCTSTR)m_szFilePath, PACKETCOM );
    if( m_nCommID < 0 )
    {
        m_bCommOpen = FALSE;
        return( m_nCommID );
    }

    // Parameter set for serial communication
    rc = BscSetCom( m_nCommID, m_nCommPort, m_nBaudRate,
        m_nParity, m_nDataLength, m_nStopBits);
    if( rc != 1 )
    {
        rc = BscClose( m_nCommID );
        m_bCommOpen = FALSE;
        return( -1 );
    }
}

// ** Ethernet communcations option:
else
{

    // Open the Ethernet port
    m_nCommID = BscOpen( (char *)(LPCTSTR)m_szFilePath, PACKETETHERNET );
    if( m_nCommID < 0 )
    {
        m_bCommOpen = FALSE;
        return( m_nCommID );
    }

    // Parameter set for Ethernet communication
    rc = BscSetEther( m_nCommID, (char *)(LPCTSTR)m_szIPAddress, 0,
        AfxGetMainWnd()->GetSafeHwnd() ); //Last parameter was garbled in sample code - che
'GetSafeHwnd'.
    if( rc != 1 )
    {
        rc = BscClose( m_nCommID );
        m_bCommOpen = FALSE;
        return( -1 );
    }
}
m_bCommOpen = TRUE;
return rc;


CRobotController::ReadPos()

int  rc;
m_nControlGroup = 1;
CString szFrameName = "UF5"; //"ROBOT";
//short nIsPulse = 1;
int     nIsExt = 1;
WORD    wForm;
WORD    wTool;
double  dPosition[12];


if( Connect() > 0 )
{

    //if(m_nControlGroup = 0)
    //  m_nControlGroup = GetControlGroup();

    //BscIsLoc function generates an error when called with nIsPulse = 0!!!
    //Only seems to work in Pulse mode!!
```

```
        //rc = BscIsLoc(nCid, nIsPulse, &wForm, dPosition);

        rc = BscIsRobotPos(m_nCommID, (char *)(LPCTSTR)szFrameName, nIsExt, &wForm, &wTool, dPo
ion);

        // Disconnect from Controller - Optional
        Disconnect();

        if(rc == 0)
        {
            m_RobotPos.X         = dPosition[0];
            m_RobotPos.Y         = dPosition[1];
            m_RobotPos.Z         = dPosition[2];
            m_RobotPos.Roll = dPosition[3];
            m_RobotPos.Pitch     = dPosition[4];
            m_RobotPos.Yaw  = dPosition[5];
            m_RobotPos.Base = dPosition[6];

            m_RobotPos.Form = wForm;
            m_RobotPos.Tool = wTool;
            m_RobotPos.Frame     = szFrameName;

            if(nIsExt)
            {
                for(int i = 0; i < 6; i++)
                {
                    m_lExtAxis[i] = (long)dPosition[i+6];
                }
            }

            return rc;

        }

        else
        {
            AfxMessageBox("Error Reading Position");
            return -1;
        }
    }
    else
    {
        AfxMessageBox("Not Open!");
        return -1;
    }

    return rc;


ring CRobotController::GetPosAsString()

    CString szPosition;

    if(ReadPos() < 0 )
    {
        szPosition = "Error getting position";
        return szPosition;
    }

    szPosition.Format("X: %.2f, Y: %.2f, Z: %.2f, R: %.2f, P: %.2f, Y: %.2f, B: %.2f",
        m_RobotPos.X,
        m_RobotPos.Y,
        m_RobotPos.Z,
        m_RobotPos.Roll,
        m_RobotPos.Pitch,
        m_RobotPos.Yaw,
        m_RobotPos.Base );

    return szPosition;


d CRobotController::SetComm(int nCommType, int nCommPort, int nBaudRate,
                                int nParity, int nDataLength, int nStopBits)

    m_nCommType = nCommType;
    m_nCommPort = nCommPort;
    m_nBaudRate = nBaudRate;
    m_nParity   = nParity;
    m_nDataLength = nDataLength;
```

```
m_nStopBits = nStopBits;


d CRobotController::SetComm(int nCommType, LPCTSTR szIPAddress )

m_nCommType = nCommType;
m_szIPAddress = szIPAddress;


CRobotController::GetCommType()

return m_nCommType;


d CRobotController::SetControllerType(int nControllerType)

m_nControllerType = nControllerType;


CRobotController::GetControllerType()

return m_nControllerType;



CRobotController::GetControlGroup()

unsigned short nControlGroup, nTaskInfo;

if(Connect() < 0)
    return -1;

BscGetCtrlGroup(m_nCommID, &nControlGroup, &nTaskInfo);

Disconnect();

m_nControlGroup = int(nControlGroup);
m_nTaskInfo = int(nTaskInfo);

return m_nControlGroup;


d CRobotController::CloseComm()


if(m_bConnected == TRUE)
    Disconnect();
if(m_bCommOpen = TRUE)
{
    BscClose(m_nCommID);
    m_bCommOpen = FALSE;
}


** Connect makes a connection to the controller.
** It will open the comm port if neccessary.
** Returns 1 if successful, 0 if Not successful
** If there is already a connection then it will return
** Unsuccessful (0) so that two commands are not sent
** to the controller at once which would result in a freeze
t CRobotController::Connect()

int rc = 0;

if(m_bConnected == TRUE)
    return 0;

// Check Comm is Open if not Open and return error if unsuccessful
if(!m_bCommOpen)
    if(OpenComm() < 0)
        return 0;

// Connect to communication port
rc = BscConnect( m_nCommID );

if( rc != 1 )
{
    m_bConnected = FALSE;
    return( rc );
```

```
  }

  m_bConnected = TRUE;

  return( rc );


t cRobotController::Disconnect()

  if(m_bCommOpen = TRUE)
  {
      m_bConnected = FALSE;
      return BscDisConnect(m_nCommID);
  }
  else
  {
      m_bConnected = FALSE;
      return -1;
  }



t CRobotController::SelectJob(CString szJobName)

  int rc = -1;

  // File name check
  if( szJobName.IsEmpty() || (szJobName.GetLength() > 12) )
  {
      AfxMessageBox ( "Error with File Name - Empty or too long!");
      return -1;
  }

  // Make Job Name UpperCase
  szJobName.MakeUpper();

  // Connect to Controller
  if( Connect() != 1)
      return -1;

  if( m_bConnected )
  {
      // Select the Job as Active Job
      rc = BscSelectJob( m_nCommID, (char *)(LPCTSTR)szJobName);
      if(rc == 0)
          rc = BscSetMasterJob( m_nCommID );

      Disconnect();

      // Check return value
      if( rc == 0 )
      {
          AfxMessageBox( "Job Selected");
          return rc;
      }
      else
      {
          AfxMessageBox ( "Couldnt Select Job" );
          return rc;
      }
  }

  return rc;



t CRobotController::SetMode(int nMode)

  int rc = -1;

  if(Connect() < 0)
      return -1;
  rc = BscSelectMode(m_nCommID, nMode);
  Disconnect();
  if(rc != 0)
      return rc;

  m_nMode = nMode;
  return m_nMode;


t CRobotController::GetMode()

  /*
  //Following code gets current value from controller.
  //Code is not needed if Update Status is called before this.

  int rc = -1;

  if(Connect() < 0)
      return -1;

  rc = BscIsTeachMode(m_nCommID);
  Disconnect();

  if( rc < 0)
      return rc;
  else
      m_nMode = rc + 1; // ** TEACH is 1, PLAY is 2 - but BscIsTeachMode return 0 for Teach a
 2 for Play
  */

  return m_nMode;


t CRobotController::GetHold()

  return m_nHold;



t CRobotController::GetCycle()

  return m_nCycle;


t CRobotController::UpdateJobStatus()

  int rc;

  char chName[32];

  if(Connect() == 1)
  {
      rc = BscIsJobName(m_nCommID, chName, 32);
      m_nJobLine = BscIsJobLine(m_nCommID);
      m_nJobStep = BscIsJobStep(m_nCommID);

      m_szCurrentJobName = (CString)chName;
      //BscFindFirst(short nCid,char *fname,short size);
      //BscFindNext(short nCid,char *fname,short size);

      //rc = BscFindFirstMaster(m_nCommID, (const char *)m_szMasterJobName, 32);
      //BscFindNextMaster(short nCid,char *fname,short size);

      Disconnect();
      return rc;
  }
  else
      return -1;


t CRobotController::UpdateStatus()

  /*
  Data 1

      Data 1 are represented by bit data in decimals.

      D0:Step                             D4:Operation at safe speed
      D1:1-cycle                          D5:Teach *
      D2:Auto operation                   D6:Play *
      D3:Operating                        D7:Command remote
      *: Effective only for XRC and MRC.

  Data 2
```

```
    Data 2 are represented by bit data in decimals.

        D0:Hold (XRC/MRC: Playback box hold, ERC:Panel hold)
        D1:Hold (XRC/MRC: Programming pendant hold, ERC: T-BOX hold)
        D2:Hold (External hold)
        D3:Hold(Command hold)
        D4:Alarm occurred
        D5:Error occurred
        D6:Servo ON
*/

WORD wStatus1, wStatus2;
int rc;

if(Connect() == 1)
{
    rc = BscGetStatus(m_nCommID, &wStatus1, &wStatus2);
    Disconnect();
}
else
    return -1;

// ** Interpret response from Controller
if(wStatus1 & 0x01)
    m_nCycle = STEP;
if(wStatus1 & 0x02)
    m_nCycle = CYCLE;
if(wStatus1 & 0x04)
    m_nCycle = AUTO;

m_bOperating = (wStatus1 & 0x08) != 0;
m_bSafeOperating = (wStatus1 & 0x10) != 0;

if(wStatus1 & 0x20)
    m_nMode = TEACH;
if(wStatus1 & 0x40)
    m_nMode = PLAY;

m_bRemoteOn = (wStatus1 & 0x80) != 0;

m_nHold = 0;
if(wStatus2 & 0x01)
    m_nHold = HOLD_PANEL;
if(wStatus2 & 0x02)
    m_nHold = HOLD_PENDANT;
if(wStatus2 & 0x04)
    m_nHold = HOLD_EXTERNAL;
// ** Hold by PC Command does not seems to register in the Status Data returned - it is alw
s zero.
if(wStatus2 & 0x08)
    m_nHold = HOLD_PC;

if(m_nHold != 0)
    m_bHoldOn = TRUE;
else
    m_bHoldOn = FALSE;

m_bAlarmed  = (wStatus2 & 0x10) != 0;
m_bErrored  = (wStatus2 & 0x20) != 0;
m_bServoOn  = (wStatus2 & 0x40) != 0;

if(m_bErrored || m_bAlarmed)
{
    GetErrorAlarm();
}

return 0;


t CRobotController::GetErrorCode()

return m_nErrorCode;


t CRobotController::GetAlarm(int &nAlarmCode, WORD &wAlarmData)

if(m_bAlarmed)
{
    nAlarmCode = m_nAlarmCode;
    wAlarmData = m_wAlarmData;
}
else
    return -1;

return 0;


CRobotController::GetErrorAlarm()

int rc = -1;
WORD wAlarmData;

if(Connect() == 1)
{
    switch (m_nControllerType)
    {
    case MRC:
        rc = BscGetError(m_nCommID);
        break;
    case XRC:
        rc = BscGetError2(m_nCommID);
        break;
    }

    if(m_bErrored)
        m_nErrorCode = rc;

    if(m_bAlarmed)
    {
        rc = BscGetFirstAlarm(m_nCommID, &wAlarmData);
        m_nAlarmCode = rc;
        m_wAlarmData = wAlarmData;
    }

    Disconnect();
}

return rc;


t CRobotController::StartJob(CString szJobName)

int rc = -1;

// ** Select this as master job
if( SelectJob(szJobName) < 0)
{
    return -1;
}

// ** Check current status of Controller
UpdateStatus();

// ** Check Mode and change if necessary
if( m_nMode != PLAY )
{
    rc = SetMode(PLAY);
    if(rc < 0)
        return rc;
}

// ** Turn Servo On if necessary
if( !m_bServoOn )
{
    rc = SetServo(ON);
    if(rc < 0)
        return rc;
}

// ** If any of the following are true then return
if( m_bAlarmed || m_bErrored || m_bOperating || m_nHold )
    return -1;

// ** StartJob
if(Connect() < 0)
    return -1;

rc = BscStartJob(m_nCommID);
```

```
 Disconnect();

 return rc;


OL CRobotController::SetServo(int nState)

 int rc = -1;

 if( Connect() < 0)
     return -1;

 // ** Turn Servo On
 if(nState == ON)
 {
     rc = BscServoOn(m_nCommID);
     Disconnect();
     if( rc  != 0 )
         return rc;
     m_bServoOn = TRUE;
 }

 // ** Turn Servo Off
 else
 {
     rc = BscServoOff(m_nCommID);
     Disconnect();
     if( rc  != 0 )
         return rc;
     m_bServoOn = FALSE;
 }

 return m_bServoOn;


OL CRobotController::SetHold(int nState)

 int rc = -1;

 if( Connect() < 0)
     return -1;

 // ** Turn Hold On
 if(nState == ON)
 {
     rc = BscHoldOn(m_nCommID);
     Disconnect();
     if( rc  != 0 )
         return rc;
     m_bHoldOn = TRUE;
 }

 // ** Turn Servo Off
 else
 {
     rc = BscHoldOff(m_nCommID);
     Disconnect();
     if( rc  != 0 )
         return rc;
     m_bHoldOn = FALSE;
 }

 return m_bHoldOn;


tring CRobotController::ReadJobList()

 int rc = -1;
 char* JobName = new char[30];
 CString szJobList;

 if(Connect() < 0)
     return -1;

 rc = BscFindFirst(m_nCommID, JobName, 30);

 if( rc != 0)
 {
    Disconnect();
    return rc;
}

while( rc == 0 )
{
    szJobList += JobName;
    szJobList += "\r\n";
    rc = BscFindNext(m_nCommID, JobName, 30);
}

Disconnect();

return szJobList;


CRobotController::DeleteJob(CString szJobName)

int rc = -1;

if(Connect() < 0)
    return -1;

if( m_bConnected )
{
    // Select Job First
    rc = BscSelectJob( m_nCommID, (char *)(LPCTSTR)szJobName);
    // Delete if Selection was successful
    if( rc != 0 )
    {
        // Couldnt select Job
        Disconnect();
        return rc;
    }

    rc = BscDeleteJob(m_nCommID);
    Disconnect();

    if( rc == 1 )
    {
        AfxMessageBox ( "Cannot Delete Job!" );
        return rc;
    }
    else if( rc != 0)
    {
        CString szMessage;
        szMessage.Format("Error: %d when trying to Delete File!", rc );
        AfxMessageBox (szMessage);
        return rc;
    }

    AfxMessageBox ( "Job Deleted!!");

    return rc;
}

return rc;


t CRobotController::SetCycle(int nCycle)

int rc = -1;

if(Connect() < 0)
    return -1;
switch( nCycle )
{

case STEP :
    rc = BscSelStepCycle(m_nCommID);
    break;

case CYCLE :
    rc = BscSelOneCycle(m_nCommID);
    break;

case AUTO:
    rc = BscSelLoopCycle(m_nCommID);
    break;
```

```
default:
    return -1;
}

Disconnect();
if( rc == 0 )
{
    m_nCycle = nCycle;
    return m_nCycle;
}

return rc;


CRobotController::ContinueJob()

int rc = -1;

if(Connect() < 0)
    return rc;

rc = BscContinueJob(m_nCommID);
Disconnect();

return rc;



OL CRobotController::IsAlarmed()

return m_bAlarmed;


OL CRobotController::IsErrored()

return m_bErrored;


OL CRobotController::IsRobotOperating()

return m_bOperating;


OL CRobotController::IsSafeOperating()

return m_bSafeOperating;


OL CRobotController::IsServoOn()

return m_bServoOn;


OL CRobotController::IsHoldOn()

return m_bHoldOn;


OL CRobotController::IsRemoteOn()

return m_bRemoteOn;


t CRobotController::GetCurrentPos(int nControlGroup, RobotPos* pRobotPos)


ReadPos();
memcpy(pRobotPos,&m_RobotPos,sizeof(m_RobotPos));

return 0;


t CRobotController::GetRobotAxisPulse(long* plPulse)

int rc = -1;
```

```
if(Connect() < 0)
    return rc;

rc = UpdateRobotPosition("PULSE", "", 1);

if(rc == 0)
    memcpy(plPulse, m_lRobotPulse, sizeof(long) * 6);

Disconnect();

return rc;


CRobotController::GetExtAxisPulse(long* plPulse)

int rc = -1;

if(Connect() < 0)
    return rc;

rc = UpdateRobotPosition("PULSE", "", 1);

if(rc == 0)
    memcpy(plPulse, m_lExtAxis, sizeof(long) * 6);

Disconnect();

return rc;



CRobotController::GetRobotPos(LPCTSTR szPosType, LPCTSTR szFrameName, int nIsExt,
                              RobotPos* pRobotPos)

int rc = -1;

if(Connect() < 0)
    return rc;

rc = UpdateRobotPosition("RECTAN", szFrameName, 1);

if( rc == 0)
    *pRobotPos = m_RobotPos;

Disconnect();

return rc;


CRobotController::MonitorRobotPos(LPCTSTR szPosType, LPCTSTR szFrameName, int nIsExt,
                                  RobotPos* pRobotPos)

int rc = -1;

// ** Connect if not connected
if(!m_bConnected)
    if(Connect() < 0)
        return rc;

rc = UpdateRobotPosition("RECTAN", szFrameName, 1);

if( rc == 0)
    *pRobotPos = m_RobotPos;

// ** No disconnection from controller

return rc;



t CRobotController::UpdateRobotPosition(LPCTSTR szPosType, LPCTSTR szFrameName, int nIsExt)

int rc;
WORD    wForm;
WORD    wTool;
double  dPosition[12];
```

```
//BscIsLoc function generates an error when called with nIsPulse = 0!!!
//Only seems to work in Pulse mode!!

if(szPosType == "PULSE")
{
    rc = BscIsLoc(m_nCommID, 1, &wForm, dPosition);

    if(rc == 0)
    {
        for( int i=0; i < 6; i++)
        {
            m_lRobotPulse[i] = (long)dPosition[i];
            m_lExtAxis[i]    = (long)dPosition[i+6];
        }
    }
}

if(szPosType == "RECTAN")
{
    rc = BscIsRobotPos(m_nCommID, (char *)(LPCTSTR)szFrameName, nIsExt, &wForm, &wTool, dPo
ion);

    if(rc == 0)
    {
        m_RobotPos.X        = dPosition[0];
        m_RobotPos.Y        = dPosition[1];
        m_RobotPos.Z        = dPosition[2];
        m_RobotPos.Roll     = dPosition[3];
        m_RobotPos.Pitch    = dPosition[4];
        m_RobotPos.Yaw      = dPosition[5];
        m_RobotPos.Base     = dPosition[6];

        m_RobotPos.Form     = wForm;
        m_RobotPos.Tool     = wTool;
        //** Crashes here???
        m_RobotPos.Frame    = szFrameName;

        if(nIsExt)
        {
            for(int i = 0; i < 6; i++)
            {
                m_lExtAxis[i] = (long)dPosition[i+6];
            }
        }
    }
}

return rc;


CRobotController::SetControlGroup(int nControlGroup1, int nControlGroup2)

// ** Sets the control croup on the controller - XRC command can set two groups

int rc;

if(Connect() < 0)
    return -1;

switch(m_nControllerType)
{
case MRC:
    {
        rc = BscSetCtrlGroup(m_nCommID, nControlGroup1);
    }
    break;

case XRC:
    {
        rc = BscSetCtrlGroupXrc(m_nCommID, nControlGroup1, nControlGroup2);
    }
    break;
}

if(rc != 0)
{
    return rc;
}
```

```
// ** Need to add stuff for XRC twin groups
m_nControlGroup = nControlGroup1;

return rc;



CRobotController::IsControlGroup()

if(Connect() < 0)
    return -1;

m_nControlGroup = BscIsCtrlGroup(m_nCommID);

Disconnect();

return m_nControlGroup;



CRobotController::Test()

int rc;
short axisno = 1;
long abso;

rc = BscGetAbso(m_nCommID,axisno, &abso);

return abso;


CRobotController::Move( LPCTSTR szMoveType, LPCTSTR szSpeedType, double dSpeed,
        CString szFrame, int nForm, int nTool, double* pdPos)

int rc = -1;
// ** MoveType can be either - MOVJ, MOVL, or IMOV

if( Connect() > 0 )
{
    rc = BscMov(m_nCommID, (char *)(LPCTSTR)szMoveType, (char *)(LPCTSTR)szSpeedType, dSpee

        (char *)(LPCTSTR)szFrame, (short)nForm, (short)nTool, pdPos);

    Disconnect();
}

return rc;



CRobotController::PMove(LPCTSTR szMoveType, LPCTSTR szSpeedType,
        double dSpeed, int nTool, double* pdPos)

int rc = -1;
// ** MoveType can be either - MOVJ, or MOVL

if( Connect() > 0 )
{
    rc = BscPMov(m_nCommID, (char *)(LPCTSTR)szMoveType, (char *)(LPCTSTR)szSpeedType,
        dSpeed, (short)nTool, pdPos);

    Disconnect();
}

return rc;


CRobotController::ReadPosStream()

int  rc;
int count = 0;
WORD    wForm;
WORD    wTool;
double  dPosition[12];
CString szFrameName = "ROBOT";
int nIsExt = 0;

if( Connect() > 0 )
{
    for(int i=0; i<100; i++)
    {

    //BscIsLoc function generates an error when called with nIsPulse = 0!!!
    //Only seems to work in Pulse mode!!
    //rc = BscIsLoc(nCid, nIsPulse, &wForm, dPosition);

    rc = BscIsRobotPos(m_nCommID, (char *)(LPCTSTR)szFrameName, nIsExt, &wForm, &wTool, dPo
ion);

    if(rc == 0)
    {
        /*
        m_RobotPos.X        = dPosition[0];
        m_RobotPos.Y        = dPosition[1];
        m_RobotPos.Z        = dPosition[2];
        m_RobotPos.Roll = dPosition[3];
        m_RobotPos.Pitch    = dPosition[4];
        m_RobotPos.Yaw  = dPosition[5];
        m_RobotPos.Base = dPosition[6];

        m_RobotPos.Form = wForm;
        m_RobotPos.Tool = wTool;
        m_RobotPos.Frame    = szFrameName;

        if(nIsExt)
        {
            for(int i = 0; i < 6; i++)
            {
                m_nExtAxis[i] = (long)dPosition[i+6];
            }
        }
        */
        count++;
    }
    }
}

else
    {
        AfxMessageBox("Error Reading Position");
    }

// Disconnect from Controller - Optional
Disconnect();

return count;


CRobotController::IMove()

//rc = BscImov(m_nCommID, chVtype, dSpeed, chFrameName, (short)nTool, dPosition);

return 1;



CRobotController::MoveJointRelative(double dJointAngles[6])

int rc = -1;
double dNewPulse[12] = {0,0,0,0,0,0,0,0,0,0,0,0};

UpdateRobotPosition("PULSE", "", 1);

for(int i=0; i < 6; i++)
{
    dNewPulse[i] = m_lRobotPulse[i];
    dNewPulse[i] += (dJointAngles[i] * m_lRobotJointPPR[i] / 360);
}

rc = PMove("MOVJ", "V", 5, 0, dNewPulse);

return rc;

d CRobotController::SetRobotPPR(long lRobotPPR[])

for(int i=0; i<12; i++)
    m_lRobotJointPPR[i] = lRobotPPR[i];
```

```
CRobotController::GetRobotAxisDegrees(double dDegrees[12])

int rc = -1;
long lRobotPulse[6], lExtPulse[6], lTemp;

rc = GetRobotAxisPulse(lRobotPulse);
rc = GetExtAxisPulse(lExtPulse);

if(rc != 0)
   return rc;

for(int i=0; i<6; i++)
{
   dDegrees[i]   = (lRobotPulse[i] / (m_lRobotJointPPR[i]/360.0));
   dDegrees[i+6] = (lExtPulse[i] / (m_lRobotJointPPR[i+6]/360.0));
}

// ** Round off the values to the nearest 0.001 degrees
for(i=0; i<12; i++)
{
   lTemp = long(dDegrees[i] * 1000);
   dDegrees[i] = (double)lTemp / 1000.0;
}


return rc;


t CRobotController::ResetAlarm()

int rc;

if(Connect() == 1)
{
   rc = BscReset(m_nCommID);
   Disconnect();

}
else rc = -1;

return rc;


t CRobotController::CancelError()

int rc;

if(Connect() == 1)
{
   rc = BscCancel(m_nCommID);
   Disconnect();

}
else rc = -1;

return rc;


OL CRobotController::IsConnected()

return m_bConnected;


OL CRobotController::IsCommOpen()

return m_bCommOpen;


t CRobotController::GetJobLine()

return m_nJobLine;


t CRobotController::GetJobStep()

return m_nJobStep;
```

```
ring CRobotController::GetCurrentJobName()
 return m_szCurrentJobName;

 CRobotController::GetTaskInfo()
 return m_nTaskInfo;

ring CRobotController::GetFilePath()
 return m_szFilePath;
```

```
RobotController.h: interface for the CRobotController class.

//////////////////////////////////////////////////////////////////////

 !defined(AFX_ROBOTCONTROLLER_H__BE86056E_C7E4_48F2_A51A_DDA1CF4EAD16__INCLUDED_)
fine AFX_ROBOTCONTROLLER_H__BE86056E_C7E4_48F2_A51A_DDA1CF4EAD16__INCLUDED_

 _MSC_VER > 1000
agma once
dif // _MSC_VER > 1000

fine ETHERNET    0
fine RS232       1
fine MRC         0
fine XRC         1

fine OFF         0
fine ON          1

fine TEACH       1
fine PLAY        2

fine STEP        1
fine CYCLE       2
fine AUTO        3

fine HOLD_PANEL          1
fine HOLD_PENDANT        2
fine HOLD_EXTERNAL       3
fine HOLD_PC             4

uct RobotPos

 BOOL     IsPulse;
 CString  Frame;
 int      Form;
 int      Tool;
 double   X;
 double   Y;
 double   Z;
 double   Roll;
 double   Pitch;
 double   Yaw;
 double   Base;



ss CRobotController

lic:
 CString GetFilePath();
 int GetTaskInfo();
 CString GetCurrentJobName();
 int GetJobStep();
 int GetJobLine();
 BOOL IsCommOpen();
 BOOL IsConnected();
 //szJobList szlistMasterJobList;

 // ** Job status variables
 CString m_szCurrentJobName;
 CString m_szMasterJobName;
 CString m_szMasterJobName2;

 int UpdateJobStatus();
 int CancelError();
 int ResetAlarm();
 int GetAlarm(int& nAlarmCode, WORD& wAlarmData);
 int GetErrorCode();
 BOOL IsErrored();
 BOOL IsAlarmed();
 int GetHold();
 BOOL IsSafeOperating();
 BOOL IsRobotOperating();
 BOOL IsRemoteOn();
 // ** Use this to continually monitor Robot Position
 int MonitorRobotPos(LPCTSTR szPosType, LPCTSTR szFrameName, int nIsExt,
                              RobotPos* pRobotPos);
```

```
// ** These are Pulse Per Rev values for the Robot used
long m_lRobotJointPPR[12];
void SetRobotPPR(long lRobotPPR[12]);

int MoveJointRelative(double dJointAngles[12]);

int IMove();
int ReadPosStream();
int Test();
int IsControlGroup();
int SetControlGroup(int nControlGroup, int nControlGroup2 = -1);

// **************************** //
// ** Job Management Actions ** //
// **************************** //

// ** Read from the controller the complete list of jobs
// ** Job names will be returned as a CString with each job on a separate line
CString ReadJobList();

// ** Sends a file to the Controller - will not replace existing files
int SendFile(CString szFileName);
// ** Receives a File to the path directory - Replaces current file if it exists!
int ReceiveFile(CString szFileName);

// ** Selects the specified Job
int SelectJob(CString szJobName);

// ** Delete the specified job from the controller
int DeleteJob(CString szJobName);

// **************************** //
// ** Job Playback Related   ** //
// **************************** //

// ** Starts the currently selected job or the specified job
int StartJob(CString szJobName = "");

// ** Re-Starts job - Execution starts from the current line of the current job.
// ** Use this to continue Playback after Hold, Stop, or Pause.
int ContinueJob();

// **************************** //
// ** Motion Related Actions ** //
// **************************** //

// ** Get the RobotPos info for specified control group
int GetCurrentPos(int nControlGroup, RobotPos* pRobotPos);

int GetRobotPos(LPCTSTR szPosType, LPCTSTR szFrameName, int nIsExt, RobotPos* pRobotPos);
int GetRobotAxisPulse(long *plPulse);
int GetExtAxisPulse(long* plPulse);
int GetRobotAxisDegrees(double dDegrees[12]);

int ReadPos();

CString GetPosAsString();

int Move(LPCTSTR szMoveType, LPCTSTR szSpeedType, double dSpeed,
       CString szFrame, int nForm, int nTool, double* pdPos);

int PMove(LPCTSTR szMoveType, LPCTSTR szSpeedType,
       double dSpeed, int nTool, double* pdPos);

// ** Get the currently selected Control Group
int GetControlGroup();

// ******************************** //
// ** Controller Status Actions ** //
// ******************************** //

// ** Update the current status info for the Controller
int UpdateStatus();

// ** Following Get functions only return values from last controller status update.
// ** Always refresh the values from the controller with - UpdateStatus()

int GetMode();
int GetCycle();
```

```
 BOOL IsHoldOn();
 BOOL IsServoOn();

 // ** Following Set functions act immediately once called

 // ** Change the Playback Cycle mode - Step, Cycle, Auto (1,2,3)
 int SetCycle(int nCycle);
 // ** Set the Controller mode - Teach, Play (1,2)
 int SetMode(int nMode);
 // ** Turn Servo ON or OFF
 BOOL SetServo(int nState = ON);
 // ** Turn Hold ON or OFF - Pauses Playback - Hold off does not restart
 BOOL SetHold(int nState = ON);

 // *********************************** //
 // ** Communication Related Actions ** //
 // *********************************** //

 // ** Open the Communications Port - returns an integer
 int OpenComm();
 // ** Close the Communications Port
 void CloseComm();

 // ** Makes a connection to the Robot controller (Open Port first)
 int Connect();
 // ** Disconnects from the Robot Controller
 int Disconnect();

 // ** Set the Communication Parameters for RS232 type
 // * Communication port number 1: COM1, 2: COM2, 3: COM3, 4: COM4
 // * Baud rate 300, 600, 1200, 2400, 4800, 9600
 // * Parity  0: None, 1: Odd, 2: Even
 // * Data length  7: 7 bits, 8: 8 bits
 // * Stop bit  0: 1 bit, 1: 1.5 bits, 2: 2 bits
 void SetComm(int nCommType, int nCommPort, int nBaudRate = 9600,
    int nParity = 2, int nDataLength = 8, int nStopBits = 0);

 // ** Set the Communication Parameters for Ethernet type
 void SetComm(int nCommType, LPCTSTR szIPAddress);

 // ** Get which Comm type is currently set
 int GetCommType();

 // ** Set/Get the Motoman Controller Type - MRC or XRC (0 or 1)
 void SetControllerType(int nControllerType);
 int GetControllerType();

 // ** Constuctors
 CRobotController(int ControllerType, LPCTSTR FilePath = "");
 CRobotController();

 virtual ~CRobotController();


vate:
 int m_nStopBits;
 int m_nDataLength;
 int m_nParity;

 int m_nJobStep;
 int m_nJobLine;

 WORD m_wAlarmData;
 int m_nTaskInfo;

 long m_lRobotPulse[6];
 // ** Array to store pulse values of external axis - Set to 6 max currently
 long m_lExtAxis[6];

 BOOL m_bHoldOn;
 BOOL m_bServoOn;
 int  m_nHold;
 BOOL m_bAlarmed;
 BOOL m_bErrored;
 int  m_nErrorCode;
 int  m_nAlarmCode;

 // ** IsSafeOperating indicates if "Safety Speed" is set in "Special Play Options"
 BOOL m_bSafeOperating;
```

```
// ** IsOperating indicates if the Robot/Groups are currently moving
BOOL m_bOperating;
// ** Specifies if Remote Mode is On or Off - TRUE or FALSE
BOOL m_bRemoteOn;
// ** The currently selected Mode - TEACH (1) or PLAY (2)
int m_nMode;
// ** The currently selcted Cycle - STEP (1 step at a time), CYCLE (1 Job), AUTO
int m_nCycle;
// ** Indicates if a connection between the PC and Controller is active
BOOL m_bConnected;
// ** Robot Position storage variables
RobotPos m_RobotPos;

// ** Communication port number 1: COM1, 2: COM2, 3: COM3, 4: COM4
int m_nCommPort;
// ** Baud rate 300, 600, 1200, 2400, 4800, 9600
int m_nBaudRate;
// ** The IP Address of the Controller (For Ethernet Option)
LPCTSTR     m_szIPAddress;
// ** Communcications handler ID to identify the controller
int         m_nCommID;
// ** Commmunication type between PC and Controller: RS232 or ETHERNET
int         m_nCommType;
// ** Comm Open: TRUE or FALSE
BOOL        m_bCommOpen;
// ** Controller Type: XRC or MRC
int         m_nControllerType;
// ** Path where Files are loaded to/from
CString     m_szFilePath;
// ** Active Control Group
int m_nControlGroup;

// ********************* //
// ** Private Functions:* //
// ********************* //

// ** Update the robot position info.
// ** Must connect/disconnect before and after calling this
int UpdateRobotPosition(LPCTSTR szPosType, LPCTSTR szFrameName, int nIsExt);
int GetErrorAlarm();



** Motocom32.dll all commands available


File Management
scDownLoad(short nCid,char *fname);
scUpLoad(short nCid,char *fname);
DownLoadEx(short nCid,char *fname, char *srcPath, BOOL nFlg);
UpLoadEx(short nCid,char *fname, char *desPath, BOOL nFlg);
scDeleteJob(short nCid);

Status Queries
scGetStatus(short nCid,WORD *d1,WORD *d2);
IsHold(short nCid);
IsCycle(short nCid);
IsPlayMode(short nCid);
scIsTeachMode(short nCid);
IsRemoteMode(short nCid);
IsServo(short nCid);
scIsCtrlGroup(short nCid);
IsTaskInf(short nCid);
scGetCtrlGroup(short nCid,WORD *groupinf,WORD *taskinf);
--> XRC Œ`'Ô□î•ñ"``—'Î‰ž
IsCtrlGroupXrc(short nCid, WORD *robtask,WORD *stattask);
IsTaskInfXrc(short nCid);
GetCtrlGroupXrc(short nCid,WORD *robtask,WORD *stattask,WORD *taskinf);
<---

Status Setting
scSelectMode(short nCid,short mode);
SelStepCycle(short nCid);
SelOneCycle(short nCid);
SelLoopCycle(short nCid);
scSetCtrlGroup(short nCid,WORD groupno);
--> XRC Œ`'Ô□î•ñ"``—'Î‰ž
scSetCtrlGroupXrc(short nCid,WORD groupno1, WORD groupno2);
```

```
scHoldOn(short nCid);
scHoldOff(short nCid);
scServoOn(short nCid);
scServoOff(short nCid);

Robot Position
IsLoc(short nCid,short ispulse,WORD *rconf,double *p);
scIsRobotPos(short nCid,char *framename,short isex,WORD *rconf,WORD *toolno,double *p);

Robot Move Commands
Movj(short nCid,double spd,char *framename,short rconf,short toolno,double *p);
Movl(short nCid,char *vtype,double spd,char *framename,short rconf,short toolno,double *p);
Imov(short nCid,char *vtype,double spd,char *framename,short toolno,double *p);
scMov(short nCid,char *movtype,char *vtype,double spd,char *framename,short rconf,short tool
double *p);
PMovj(short nCid,double spd,short toolno,double *p);
PMovl(short nCid,char *vtype,double spd,short toolno,double *p);
PMov(short nCid,char *movtype,char *vtype,double spd,short toolno,double *p);

Robot Errors Alarms
IsError(short nCid);
IsAlarm(short nCid);
** BUG ** - IsErrorCode doesnt seen to work - it returns 0 always
IsErrorCode(short nCid);

scGetError(short nCid);
scGetError2(short nCid);//990224 Sakasegawa for XRC
scGetFirstAlarm(short nCid,WORD *data);
GetNextAlarm(short nCid, WORD *data);
scReset(short nCid);
scCancel(short nCid);

Job Queries
IsJobName(short nCid,char *jobname,short size);
IsJobLine(short nCid);
IsJobStep(short nCid);
scFindFirst(short nCid,char *fname,short size);
scFindNext(short nCid,char *fname,short size);
FindFirstMaster(short nCid,char *fname,short size);
FindNextMaster(short nCid,char *fname,short size);

Job Playback
scSelectJob(short nCid,char *name);
scSetMasterJob(short nCid);
SetLineNumber(short nCid,short line);
scStartJob(short nCid);
scContinueJob(short nCid);
JobWait(short nCid,short time);
ChangeTask(short nCid,short task);

Other Job Commands
ConvertJobR2P(short nCid,char *name,short cv_type,char *var_no);
GetVarData(short nCid,short type,short varno,double *pos);
PutVarData(short nCid,short type,short varno,double *dat);
GetUFrame(short nCid,char *ufname,double *p);
PutUFrame(short nCid,char *ufname,double *p);
ConvertJobP2R(short nCid,char *name,char *framename);

IO Commands
ReadIO(short nCid,WORD startadd,WORD ionum,WORD *stat);
WriteIO(short nCid,WORD startadd,WORD ionum,WORD *stat);

Robot Interlocking
OPLock(short nCid);
OPUnLock(short nCid);

Data Transmission Basic Command Sending
Command(short nCid,char *h_buf,char *d_buf,short fforever);
Status(short nCid,char *hpt,char *dpt,unsigned short sz,char *rbuf);

DCI Commands
DCILoadSave(short nCid,short timec);
DCILoadSaveOnce(short nCid);
DCIGetPos(short nCid,WORD *type,WORD *rconf,double *p);
DCIPutPos(short nCid,WORD type,WORD rconf,double *p);

Basic Serial Comm Commands
scOpen(char *path,short mode);
scClose(short nCid);
```

```
scSetCom(short nCid,short port,DWORD baud,short parity,short clen,short stp);
SetHSL(short nCid,LPCTSTR strName,short port);
scConnect(short nCid);
scDisConnect(short nCid);
Puts(short nCid,char *bufptr,WORD length);
Gets(short nCid,char *bufptr,WORD bsize,WORD *plengets);
InBytes(short nCid);
OutBytes(short nCid);
scDelay(short nCid,DWORD a);
SetCondBSC(short nCid,short timerA,short timerB,short rtyR,short rtyW);
SetBreak(short nCid,short flg);
SetEther( short nCid, char FAR *IPaddr, short mode, HWND hWnd);

Misc
MDSP(short nCid,char *ptr);
DiskFreeSizeGet(short nCid,short dno,long *dsize);
**************************************************** //
70909
GetAbso(short nCid,short axisno,long *abso);
SetAbso(short nCid,short axisno,long abso);




dif // !defined(AFX_ROBOTCONTROLLER_H__BE86056E_C7E4_48F2_A51A_DDA1CF4EAD16__INCLUDED_)
```

```
Serial.cpp

clude "stdafx.h"
clude "Serial.h"

rial::CSerial()


 memset( &m_OverlappedRead, 0, sizeof( OVERLAPPED ) );
 memset( &m_OverlappedWrite, 0, sizeof( OVERLAPPED ) );
 m_hIDComDev = NULL;
 m_bOpened = FALSE;



rial::~CSerial()


 Close();



L CSerial::Open( int nPort, int nBaud )


 if( m_bOpened ) return( TRUE );

 char szPort[15];
 char szComParams[50];
 DCB dcb;

 wsprintf( szPort, "COM%d", nPort );
 m_hIDComDev = CreateFile( szPort, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FIL
TTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, NULL );
 if( m_hIDComDev == NULL ) return( FALSE );

 memset( &m_OverlappedRead, 0, sizeof( OVERLAPPED ) );
 memset( &m_OverlappedWrite, 0, sizeof( OVERLAPPED ) );

 COMMTIMEOUTS CommTimeOuts;
 CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF;
 CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
 CommTimeOuts.ReadTotalTimeoutConstant = 0;
 CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
 CommTimeOuts.WriteTotalTimeoutConstant = 5000;
 SetCommTimeouts( m_hIDComDev, &CommTimeOuts );

 wsprintf( szComParams, "COM%d:%d,n,8,1", nPort, nBaud );

 m_OverlappedRead.hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
 m_OverlappedWrite.hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );

 dcb.DCBlength = sizeof( DCB );
 GetCommState( m_hIDComDev, &dcb );
 dcb.BaudRate = nBaud;
 dcb.ByteSize = 8;
 unsigned char ucSet;
 ucSet = (unsigned char) ( ( FC_RTSCTS & FC_DTRDSR ) != 0 );
 ucSet = (unsigned char) ( ( FC_RTSCTS & FC_RTSCTS ) != 0 );
 ucSet = (unsigned char) ( ( FC_RTSCTS & FC_XONXOFF ) != 0 );
 if( !SetCommState( m_hIDComDev, &dcb ) ||
     !SetupComm( m_hIDComDev, 10000, 10000 ) ||
     m_OverlappedRead.hEvent == NULL ||
     m_OverlappedWrite.hEvent == NULL ){
     DWORD dwError = GetLastError();
     if( m_OverlappedRead.hEvent != NULL ) CloseHandle( m_OverlappedRead.hEvent );
     if( m_OverlappedWrite.hEvent != NULL ) CloseHandle( m_OverlappedWrite.hEvent );
     CloseHandle( m_hIDComDev );
     return( FALSE );
     }

 m_bOpened = TRUE;

 return( m_bOpened );



L CSerial::Close( void )
```

```
if( !m_bOpened || m_hIDComDev == NULL ) return( TRUE );

if( m_OverlappedRead.hEvent != NULL ) CloseHandle( m_OverlappedRead.hEvent );
if( m_OverlappedWrite.hEvent != NULL ) CloseHandle( m_OverlappedWrite.hEvent );
CloseHandle( m_hIDComDev );
m_bOpened = FALSE;
m_hIDComDev = NULL;

return( TRUE );



L CSerial::WriteCommByte( unsigned char ucByte )

BOOL bWriteStat;
DWORD dwBytesWritten;

bWriteStat = WriteFile( m_hIDComDev, (LPSTR) &ucByte, 1, &dwBytesWritten, &m_OverlappedWrit
;
if( !bWriteStat && ( GetLastError() == ERROR_IO_PENDING ) ){
    if( WaitForSingleObject( m_OverlappedWrite.hEvent, 1000 ) ) dwBytesWritten = 0;
    else{
        GetOverlappedResult( m_hIDComDev, &m_OverlappedWrite, &dwBytesWritten, FALSE );
        m_OverlappedWrite.Offset += dwBytesWritten;
        }
    }

return( TRUE );



CSerial::SendData( const char *buffer, int size )


if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );

DWORD dwBytesWritten = 0;
int i;
for( i=0; i<size; i++ ){
    WriteCommByte( buffer[i] );
    dwBytesWritten++;
    }

return( (int) dwBytesWritten );



CSerial::ReadDataWaiting( void )


if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );

DWORD dwErrorFlags;
COMSTAT ComStat;

ClearCommError( m_hIDComDev, &dwErrorFlags, &ComStat );

return( (int) ComStat.cbInQue );



CSerial::ReadData( void *buffer, int limit )


if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );

BOOL bReadStatus;
DWORD dwBytesRead, dwErrorFlags;
COMSTAT ComStat;

ClearCommError( m_hIDComDev, &dwErrorFlags, &ComStat );
if( !ComStat.cbInQue ) return( 0 );

dwBytesRead = (DWORD) ComStat.cbInQue;
if( limit < (int) dwBytesRead ) dwBytesRead = (DWORD) limit;

bReadStatus = ReadFile( m_hIDComDev, buffer, dwBytesRead, &dwBytesRead, &m_OverlappedRead )
```

```
if( !bReadStatus ){
    if( GetLastError() == ERROR_IO_PENDING ){
        WaitForSingleObject( m_OverlappedRead.hEvent, 2000 );
        return( (int) dwBytesRead );
        }
    return( 0 );
    }

return( (int) dwBytesRead );
```

```
Serial.h

ndef __SERIAL_H__
fine __SERIAL_H__

fine FC_DTRDSR        0x01
fine FC_RTSCTS        0x02
fine FC_XONXOFF       0x04
fine ASCII_BEL        0x07
fine ASCII_BS         0x08
fine ASCII_LF         0x0A
fine ASCII_CR         0x0D
fine ASCII_XON        0x11
fine ASCII_XOFF       0x13

ss CSerial


lic:
 CSerial();
 ~CSerial();

 BOOL Open( int nPort = 1, int nBaud = 19200 );
 BOOL Close( void );

 int ReadData( void *, int );
 int SendData( const char *, int );
 int ReadDataWaiting( void );

 BOOL IsOpened( void ){ return( m_bOpened ); }

tected:
 BOOL WriteCommByte( unsigned char );

 HANDLE m_hIDComDev;
 OVERLAPPED m_OverlappedRead, m_OverlappedWrite;
 BOOL m_bOpened;



dif
```

```
StaticPt.cpp : implementation file


clude "stdafx.h"
clude "TrackingClass.h"
clude "StaticPt.h"
clude "TrackingClassDlg.h"

def _DEBUG
fine new DEBUG_NEW
def THIS_FILE
tic char THIS_FILE[] = __FILE__;
dif

ern char* pPointLocations[Max_Point];
ern int PointIndex;
ern int LocationPointer;
ern int CurrLocationPointer;
ern REC_DATA_CARTESIAN PtRelToLISMCartesian[Max_Point];
ern bool bReplace;
///////////////////////////////////////////////////////////////////////////
CStaticPt dialog


aticPt::CStaticPt(CWnd* pParent /*=NULL*/)
 : CDialog(CStaticPt::IDD, pParent)

 //{{AFX_DATA_INIT(CStaticPt)
 m_sStaticPtName = _T("");
 //}}AFX_DATA_INIT



d CStaticPt::DoDataExchange(CDataExchange* pDX)

 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CStaticPt)
 DDX_Control(pDX, IDC_LIST_STATIC_POINT, m_ctrlListStaticPt);
 DDX_Text(pDX, IDC_EDIT_STATIC_POINT_NAME, m_sStaticPtName);
 //}}AFX_DATA_MAP



IN_MESSAGE_MAP(CStaticPt, CDialog)
 //{{AFX_MSG_MAP(CStaticPt)
 //}}AFX_MSG_MAP
_MESSAGE_MAP()

///////////////////////////////////////////////////////////////////////////
CStaticPt message handlers

L CStaticPt::OnInitDialog()

 CDialog::OnInitDialog();

 // TODO: Add extra initialization here
 char LocationString[256], buffer[256];
 m_ctrlListStaticPt.SetHorizontalExtent(400);
 ::strcpy(LocationString, "Point Name    X Co-ordinate     Y Co-ordinate         Z Co-ordinate

 m_ctrlListStaticPt.AddString(LocationString);

 for (int index=0; index<Max_Point; index++)
 {
     if (PtRelToLISMCartesian[index].name != "\0")
     {
         ::strcpy(LocationString, PtRelToLISMCartesian[index].name);
         ::strcat(LocationString, "\t");
         ::sprintf(buffer, "%15.3f", PtRelToLISMCartesian[index].x);
         ::strcat(LocationString, buffer);
         ::strcat(LocationString, "\t");
         ::sprintf(buffer, "%15.3f", PtRelToLISMCartesian[index].y);
         ::strcat(LocationString, buffer);
         ::strcat(LocationString, "\t");
         ::sprintf(buffer, "%15.3f", PtRelToLISMCartesian[index].z);
         ::strcat(LocationString, buffer);
         m_ctrlListStaticPt.AddString(LocationString);
     }
 }
```

```
return TRUE;  // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE


d CStaticPt::OnOK()

// TODO: Add extra validation here
int rc;
UpdateData(true);
if (m_sStaticPtName.IsEmpty())
{
   AfxMessageBox("You need to type the name of the point");
   return;
}
for (int i=0; i<LocationPointer; i++)
{
   if (m_sStaticPtName.GetBuffer(m_sStaticPtName.GetLength()) == PtRelToLISMCartesian[i].n
)
       rc = AfxMessageBox("Point exist. Do you want to overwrite?", IDOK);
   switch (rc)
   {
   case IDOK:
       CurrLocationPointer = i;
       bReplace = true;
       CDialog::OnOK();
   case IDCANCEL:
       return;
   }
}
if (LocationPointer == Max_Point)
   LocationPointer = 0;
//if (PtRelToLeica[LocationPointer].name != "\0")
PtRelToLISMCartesian[LocationPointer].name = m_sStaticPtName.GetBuffer(m_sStaticPtName.GetL
th());


CDialog::OnOK();
```

```
StaticPt.h : header file

//////////////////////////////////////////////////////////////////////////

 !defined(AFX_STATICPT_H__91B0747C_B88A_4D12_98F6_0449F02FB4B9__INCLUDED_)
fine AFX_STATICPT_H__91B0747C_B88A_4D12_98F6_0449F02FB4B9__INCLUDED_

 _MSC_VER > 1000
agma once
dif // _MSC_VER > 1000

CStaticPt dialog

ss CStaticPt : public CDialog

Construction
lic:
 CStaticPt(CWnd* pParent = NULL);   // standard constructor

Dialog Data
 //{{AFX_DATA(CStaticPt)
 enum { IDD = IDD_DIALOG_STATIC };
 CListBox    m_ctrlListStaticPt;
 CString m_sStaticPtName;
 //}}AFX_DATA


Overrides
 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CStaticPt)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

Implementation
tected:

 // Generated message map functions
 //{{AFX_MSG(CStaticPt)
 virtual BOOL OnInitDialog();
 virtual void OnOK();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()


{AFX_INSERT_LOCATION}}
Microsoft Visual C++ will insert additional declarations immediately before the previous lin


dif // !defined(AFX_STATICPT_H__91B0747C_B88A_4D12_98F6_0449F02FB4B9__INCLUDED_)
```

```
ransformation.cpp : implementation file

clude "stdafx.h"
clude "TrackingClass.h"
clude "Transformation.h"
clude "TrackingClassDlg.h"

def _DEBUG
fine new DEBUG_NEW
def THIS_FILE
tic char THIS_FILE[] = __FILE__;
dif

ern char* pPointLocations[Max_Point];
ern int PointIndex;
ern int LocationPointer, OriginSelect, PosXSelect, PosYSelect;
ern REC_DATA_CARTESIAN PtRelToLISMCartesian[Max_Point];
ern CString CS[10];
ern int CS_Counter;
//////////////////////////////////////////////////////////////////////////
CTransformation dialog


ansformation::CTransformation(CWnd* pParent /*=NULL*/)
 : CDialog(CTransformation::IDD, pParent)

 //{{AFX_DATA_INIT(CTransformation)
 m_sCSName = _T("");
 //}}AFX_DATA_INIT



d CTransformation::DoDataExchange(CDataExchange* pDX)

 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CTransformation)
 DDX_Control(pDX, IDC_COMBO_AXIS1_RefPt, m_ctrlComboAxis1RefPt);
 DDX_Control(pDX, IDC_COMBO_PLANE12_RefPt, m_ctrlComboPlane12RefPt);
 DDX_Control(pDX, IDC_COMBO_ORIGIN_RefPt, m_ctrlComboOriginRefPt);
 DDX_Text(pDX, IDC_EDIT_CSName, m_sCSName);
 //}}AFX_DATA_MAP



IN_MESSAGE_MAP(CTransformation, CDialog)
 //{{AFX_MSG_MAP(CTransformation)
 //}}AFX_MSG_MAP
_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
CTransformation message handlers

L CTransformation::OnInitDialog()

 CDialog::OnInitDialog();

 // TODO: Add extra initialization here
 char LocationString[256];
 for (int index=0; index<Max_Point; index++)
 {
     if (PtRelToLISMCartesian[index].name != "\0")
     {
         ::strcpy(LocationString, PtRelToLISMCartesian[index].name);
         m_ctrlComboOriginRefPt.AddString(LocationString);
         m_ctrlComboAxis1RefPt.AddString(LocationString);
         m_ctrlComboPlane12RefPt.AddString(LocationString);
     }
 }

 CheckRadioButton(IDC_RADIO_A1PosX, IDC_RADIO_A1NegZ, IDC_RADIO_A1PosX);
 CheckRadioButton(IDC_RADIO_A2PosX, IDC_RADIO_A2NegZ, IDC_RADIO_A2PosY);
 return TRUE;  // return TRUE unless you set the focus to a control
               // EXCEPTION: OCX Property Pages should return FALSE


d CTransformation::OnOK()

 // TODO: Add extra validation here
```

```
UpdateData(true);
if (m_sCSName.IsEmpty())
{
    AfxMessageBox("You need to type the name of the point");
    return;
}
CS[CS_Counter++] = m_sCSName;
OriginSelect = m_ctrlComboOriginRefPt.GetCurSel();
PosXSelect = m_ctrlComboAxis1RefPt.GetCurSel();
PosYSelect = m_ctrlComboPlane12RefPt.GetCurSel();
CDialog::OnOK();
```

```
ransformation.h : header file

/////////////////////////////////////////////////////////////////////////////

 !defined(AFX_TRANSFORMATION_H__95953425_3377_4DB2_A94E_EDF7561673DE__INCLUDED_)
ine AFX_TRANSFORMATION_H__95953425_3377_4DB2_A94E_EDF7561673DE__INCLUDED_

 _MSC_VER > 1000
gma once
if // _MSC_VER > 1000

Transformation dialog

s CTransformation : public CDialog

onstruction
ic:
 CTransformation(CWnd* pParent = NULL);   // standard constructor

ialog Data
 //{{AFX_DATA(CTransformation)
 enum { IDD = IDD_DIALOG_TRANSFORMATION };
 CComboBox   m_ctrlComboAxis1RefPt;
 CComboBox   m_ctrlComboPlane12RefPt;
 CComboBox   m_ctrlComboOriginRefPt;
 CString m_sCSName;
 //}}AFX_DATA


verrides
 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CTransformation)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

mplementation
tected:

 // Generated message map functions
 //{{AFX_MSG(CTransformation)
 virtual BOOL OnInitDialog();
 virtual void OnOK();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()


{AFX_INSERT_LOCATION}}
icrosoft Visual C++ will insert additional declarations immediately before the previous lin


dif // !defined(AFX_TRANSFORMATION_H__95953425_3377_4DB2_A94E_EDF7561673DE__INCLUDED_)
```

```
ssClass.h : header file

////////////////////////////////////////////////////////////////////////////

edef struct tagTHREADPARAMS

 LPARAM lParam;
HREADPARAMS;
```

```
stdafx.cpp : source file that includes just the standard includes
TrackingClass.pch will be the pre-compiled header
stdafx.obj will contain the pre-compiled type information

clude "stdafx.h"
```

```
stdafx.h : include file for standard system include files,
or project specific include files that are used frequently, but
   are changed infrequently


!defined(AFX_STDAFX_H__403883F9_2D41_11D7_8A8B_000102C2E8E7__INCLUDED_)
fine AFX_STDAFX_H__403883F9_2D41_11D7_8A8B_000102C2E8E7__INCLUDED_

_MSC_VER > 1000
agma once
dif // _MSC_VER > 1000

fine VC_EXTRALEAN         // Exclude rarely-used stuff from Windows headers

clude <afxwin.h>          // MFC core and standard components
clude <afxext.h>          // MFC extensions
clude <afxdisp.h>         // MFC Automation classes
clude <afxdtctl.h>        // MFC support for Internet Explorer 4 Common Controls
ndef _AFX_NO_AFXCMN_SUPPORT
clude <afxcmn.h>          // MFC support for Windows Common Controls
dif // _AFX_NO_AFXCMN_SUPPORT

fine RadianToDegree(radian)  (57.29577951308 * radian)
fine DegreeToRadian(degree)  (0.017453292 * degree)
fine pi                   3.141592654
fine Max_Point            128
fine NullString           "\0"
fine Max_Line             4
fine Max_ViaPoint         4000

tic CWinThread* g_pComputationThread;
tic CWinThread* g_pTurboTryThread;
tic CWinThread* g_pGuidanceThread;
tic CWinThread* g_pEGTFSThread;
tic CWinThread* g_pEGTFSTurboThread;
tic CWinThread* g_pDriveRobotThread;
tic CWinThread* g_pTurboCorrThread;

{AFX_INSERT_LOCATION}}
Microsoft Visual C++ will insert additional declarations immediately before the previous lin


dif // !defined(AFX_STDAFX_H__403883F9_2D41_11D7_8A8B_000102C2E8E7__INCLUDED_)
```

```
esource.h

////////////////////////////////////////////////////////////////////////////

{NO_DEPENDENCIES}}
icrosoft Developer Studio generated include file.
sed by TrackingClass.rc

fine IDM_ABOUTBOX                      0x0010
fine IDD_ABOUTBOX                      100
fine IDS_ABOUTBOX                      101
fine IDD_TRACKINGCLASS_DIALOG          102
fine IDR_MAINFRAME                     128
fine IDB_STOP                          129
fine IDB_START                         130
fine IDD_DIALOG_STATIC                 132
fine IDD_DIALOG_TRANSFORMATION         134
fine IDD_DIALOG_PATH_GENERATION        135
fine IDC_CHECK_DISPLAY                 1000
fine IDC_STATIC_STATUS                 1001
fine IDC_EDIT_PSDX                     1002
fine IDC_EDIT_PSDY                     1003
fine IDC_EDIT_PSD_FREQ                 1004
fine IDC_EDIT_ENC1                     1005
fine IDC_EDIT_ENC2                     1006
fine IDC_EDIT_ENC_FREQ                 1007
fine IDC_EDIT_ZYGO                     1008
fine IDC_EDIT_ZYGO_FREQ                1009
fine IDC_EDIT_TIME                     1010
fine IDC_EDIT_COMPTHREAD_FREQ          1011
fine IDC_EDIT_PATH_LAST_X              1011
fine IDC_EDIT_STEP_SIZE                1012
fine IDC_COMBO_ORIGIN_RefPt            1012
fine IDC_BUTTON_INC_STEPSIZE           1013
fine IDC_COMBO_AXIS1_RefPt             1013
fine IDC_BUTTON_DEC_STEPSIZE           1014
fine IDC_COMBO_PLANE12_RefPt           1014
fine IDC_EDIT_PATH_LAST_Y              1014
fine IDC_RADIO_GPIB                    1015
fine IDC_RADIO_A1PosX                  1015
fine IDC_EDIT_PATH_LAST_Z              1015
fine IDC_RADIO_RS232                   1016
fine IDC_RADIO_A1PosY                  1016
fine IDC_RADIO_A1PosZ                  1017
fine IDC_RADIO_APR                     1017
fine IDC_RADIO_A1NegX                  1018
fine IDC_RADIO_CAT                     1018
fine IDC_RADIO_A1NegY                  1019
fine IDC_BUTTON_UP                     1019
fine IDC_RADIO_A1NegZ                  1020
fine IDC_BUTTON_LEFT                   1020
fine IDC_RADIO_A2PosX                  1021
fine IDC_BUTTON_RIGHT                  1021
fine IDC_RADIO_A2PosY                  1022
fine IDC_BUTTON_DOWN                   1022
fine IDC_RADIO_A2PosZ                  1023
fine IDC_BUTTON_SHOW_PSD               1023
fine IDC_RADIO_A2NegX                  1024
fine IDC_BUTTON_SET_HOME               1024
fine IDC_RADIO_A2NegY                  1025
fine IDC_BUTTON_HOME                   1025
fine IDC_RADIO_A2NegZ                  1026
fine IDC_BUTTON_SET_BB                 1026
fine IDC_BUTTON_GO_BB                  1027
fine IDC_EDIT_CSName                   1027
fine IDC_BUTTON_STATIC_SAMPLE          1028
fine IDC_POSX                          1029
fine IDC_EDIT_STATIC_POINT_NAME        1029
fine IDC_POSY                          1030
fine IDC_LIST_STATIC_POINT             1030
fine IDC_BUTTON_ReINIT_ZYGO            1031
fine IDC_BUTTON_INIT                   1032
fine IDC_BUTTON_SUBSYSTEM              1033
fine IDC_POSZ                          1034
fine IDC_EDIT_PATH_START_X             1034
fine IDC_POSX2                         1035
fine IDC_EDIT_PATH_START_Y             1035
fine IDC_POSY2                         1036
fine IDC_EDIT_PATH_START_Z             1036
fine IDC_STATIC_BEAM                    1037
fine IDC_LIST_VIA_POINTS                1037
fine IDC_BUTTON_TRACK                   1038
fine IDC_COMBO_PATHTYPE                 1038
fine IDC_POSZ2                          1039
fine IDC_EDIT_PATH_VEL                  1039
fine IDC_BUTTON_DYNAMIC_SAMPLE          1040
fine IDC_EDIT_PATH_ACC                  1040
fine IDC_EDIT_TURBO_FREQ                1041
fine IDC_EDIT_PATH_DEC                  1041
fine IDC_CHECK_TX                       1042
fine IDC_EDIT_PATH_RADIUS               1042
fine IDC_CHECK_TY                       1043
fine IDC_STATIC_PATH_RADIUS             1043
fine IDC_CHECK_TZ                       1044
fine IDC_BUTTON_FILE_BROWSE             1044
fine IDC_EDIT_VEL                       1045
fine IDC_STATIC_PATH_RADIUS2            1045
fine IDC_EDIT_ACC                       1046
fine IDC_DX                             1046
fine IDC_BUTTON_RESET                   1047
fine IDC_BUTTON_TRANSFORMATION          1048
fine IDC_RADIO_VIA_PT_NUM               1048
fine IDC_COMBO_CS                       1049
fine IDC_RADIO_VIA_PT_DIST              1049
fine IDC_BUTTON_PATH_GENERATION         1050
fine IDC_BUTTON_GENERATE_VIA_POINTS     1050
fine IDC_EDIT_VIA_PT_DISTNUM            1051
fine IDC_BUTTON_EGTFS                   1051
fine IDC_BUTTON_GUIDANCE                1052
fine IDC_RADIO_VIA_PT_VEL               1052
fine IDC_EDIT_GUIDANCE_FREQ             1053
fine IDC_EDIT_EGTFS_FREQ                1054
fine IDC_RADIO_MOTOCOM_CORR             1055
fine IDC_RADIO_TURBO_CORR               1056
fine IDC_EDIT_VEL2                      1057
efine IDC_STATIC_VIA_PT_DISTNUM         1058
efine IDC_EDIT_ACC2                     1058
efine IDC_EDIT_ROBOT_CUR_X              1059
efine IDC_EDIT_ROBOT_CUR_Y              1060
efine IDC_EDIT_ROBOT_CUR_Z              1061
efine IDC_BUTTON_TURBOTRY               1062
efine IDC_RADIO_MOTOCOM_DRIVE           1063
efine IDC_RADIO_PATH_DRIVE              1064
efine IDC_STATIC_DRIVE_ROBOT            1065
efine IDC_DY                            1066
efine IDC_DZ                            1067
efine IDC_BUTTON_WRITE_FILE             1068
efine IDC_BUTTON_DRIVE_BEAM             1069
efine IDC_BUTTON_GET_CUR_POS            1069
efine IDC_BUTTON_CORRECT                1070
efine IDC_BUTTON_ZERO                   1071
efine IDC_BUTTON_TRY                    1072
efine IDC_BUTTON_POINT_SAMPLE           1073
efine IDC_RADIO_MOTOCOM_CORR2           1074
efine IDC_RADIO_TURBO_CORR2             1075
efine IDC_COMBO_LINE_NUMBER             1088
efine IDC_STATIC_LP                     1089
efine IDC_BUTTON_OPEN_PORT              1092
efine IDC_BUTTON_DRIVE_ROBOT            1093

Next default values for new objects

fdef APSTUDIO_INVOKED
fndef APSTUDIO_READONLY_SYMBOLS
efine _APS_NEXT_RESOURCE_VALUE          137
efine _APS_NEXT_COMMAND_VALUE           32771
efine _APS_NEXT_CONTROL_VALUE           1074
efine _APS_NEXT_SYMED_VALUE             101
ndif
ndif
```

```
**Acroloop Program for Target Following Control Algorithm, Position Mode

SYS
HALT ALL
DETACH ALL
CONFIG ENC4 STEPPER4 STEPPER4 NONE

PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ATTACH AXIS0 ENC0 STEPPER0
ATTACH AXIS1 STEPPER1 STEPPER1
AXIS2 OFF
AXIS3 OFF
AXIS4 OFF
AXIS5 OFF
AXIS6 OFF
AXIS7 OFF

AXIS0 PGAIN 0.00017
AXIS0 IGAIN 0.005
AXIS0 ILIMIT 0.01
AXIS0 DGAIN 0.000001

AXIS1 PGAIN 0.0025
AXIS1 IGAIN 0.0
AXIS1 ILIMIT 0.0
PPU X614400 Y80000
ENC0 MULT -4
ENC1 MULT 4
RES X
RES Y
P6160=000000000

PERIOD 0.001
VEL 0.1 ACC 5 DEC 5 STP 5
JOG VEL X2 Y3
JOG ACC X3 Y5
JOG DEC X3 Y5
SET 32
SET 33
10 LV0 = P6916
20 DV0 = DV0
30 DV1 = DV1
40 DV2 = DV2
50 DV3 = P6144
60 DV4 = -1 * P6160
70 DV5 = DV3 / 614400 *360
80 DV6 = DV4 / 40000 *720
90 DV7 = -0.5 * (DV0 * cos(DV5) + DV1 * sin(DV5))
100 DV8 = 0.5 * (DV0 * -1* sin(DV5) + DV1 * cos(DV5))
110 DV10 = -1 * DV5
120 DV9 = DV2 - DV7 * tan(DV10) - DV8 * tan(DV6)
130 DA0(0) = DV9 * cos(DV6) * sin(DV10) + DV7 / cos(DV10)
140 DA1(0) = DV9 * cos(DV6) * cos(DV10)
150 DA2(0) = DV9 * sin(DV6) + DV8 / cos(DV6)
160 DV11 = ATAN(DV7/DV2/cos(DV6)) / 360 + DV5 / 360
170 DV12 = 0.5 * ATAN(DV8/DV2) / 360
180 LV2 = LV2 +1
190 JOG ABS X(DV11)
200 JOG INC Y(-1 * DV12)
210 LV1 = P6916
220 IF ((LV1-LV0) >= 1000) THEN LV3 = LV2
230 IF ((LV1-LV0) >= 1000) THEN LV2 = 0
240 IF (LV2 = 0) THEN LV0 = LV1
250 GOTO 20
DIM DV(13)
DIM LV(4)
DIM DA(3)
DIM DA0(3)
DIM DA1(3)
DIM DA2(3)
```

**Acroloop Program for Target Following Control Algorithm, Velocity Mode

```
SYS
HALT ALL
DETACH ALL
CONFIG ENC4 STEPPER4 STEPPER4 NONE

PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ATTACH AXIS0 ENC0 STEPPER0
ATTACH AXIS1 STEPPER1 STEPPER1
AXIS2 OFF
AXIS3 OFF
AXIS4 OFF
AXIS5 OFF
AXIS6 OFF
AXIS7 OFF

AXIS0 PGAIN 0.00017
AXIS0 IGAIN 0.005
AXIS0 ILIMIT 0.01
AXIS0 DGAIN 0.000001

AXIS1 PGAIN 0.0025
AXIS1 IGAIN 0.0
AXIS1 ILIMIT 0.0
PPU X614400 Y80000
ENC0 MULT -4
ENC1 MULT -4
RES X
RES Y
P06160=000000000

PERIOD 0.001
VEL 0.1 ACC 5 DEC 5 STP 5
JOG VEL X0.02 Y0.03
JOG ACC X3 Y5
JOG DEC X3 Y5
SET 32
SET 33
10 LV0 = 16816
20 DV0 = DV0
30 DV1 = DV1
40 DV2 = DV2
50 DV3 = P6144
60 DV4 = -1 * P6160
70 DV5 = DV3 / 614400 *360
80 DV6 = DV4 / 40000 *720
90 DV7 = -0.5 * (DV0 * cos(DV5) + DV1 * sin(DV5))
100 DV8 = 0.5 * (DV0 * -1* sin(DV5) + DV1 * cos(DV5))
110 DV10 = -1 * DV5
120 DV9 = DV2 - DV7 * tan(DV10) - DV8 * tan(DV6)
130 DA0(0) = DV9 * cos(DV6) * sin(DV10)  - DV7 / cos(DV10)
140 DA1(0) = DV9 * cos(DV6) * cos(DV10)
150 DA2(0) = DV9 * sin(DV6)  + DV8 / cos(DV6)
160 DV11 = -1*ATAN(DA0(0) / DA1(0)) / 360 - DV3/614400
170 DV12 = 0.5 * ATAN(DA2(0)/SQRT(DA0(0) * DA0(0) + DA1(0) * DA1(0))) / 360 - DV4
/40000
180 LV2 = LV2 +1
190 DV13 = SQRT(DV11*DV11)/0.040
200 DV14 = SQRT(DV12*DV12)/0.040
210 DV15 = DV13 * 1000
220 DV16 = DV14 * 1000
230 IF (DV13 >1.5) THEN DV13 = 1.5
240 IF (DV13 <= 0.0007) THEN DV13 = 0.0
250 IF (DV14 > 6.0) THEN DV14 = 6.0
```

```
260 IF (DV14 <=0.0004) THEN DV14 = 0.0
270 IF (DV15 >3.0) THEN DV15 = 3.0
280 IF (DV16 > 5.0) THEN DV16 = 5.0
290 JOG VEL X(DV13) Y(DV14)
300 JOG ACC X(DV15) Y(DV16)
310 JOG DEC X(DV15) Y(DV16)
320 IF (DV11 >= 0.0) THEN JOG FWD X
330 IF (DV11 < 0.0) THEN JOG REV X
340 IF (DV12 >= 0.0) THEN JOG REV Y
350 IF (DV12 < 0.0) THEN JOG FWD Y
360 LV1 = P6916
370 IF ((LV1-LV0) >= 1000) THEN LV3 = LV2
380 IF ((LV1-LV0) >= 1000) THEN LV2 = 0
390 IF (LV2 = 0) THEN LV0 = LV1
400 GOTO 20
DIM DV(17)
DIM LV(4)
DIM DA(3)
DIM DA0(3)
DIM DA1(3)
DIM DA2(3)
```

```
**Acroloop Program for Target Following Control Algorithm, Predictive Position
Mode

SYS
HALT ALL
DETACH ALL
CONFIG ENC4 STEPPER4 STEPPER4 NONE

PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ATTACH AXIS0 ENC0 STEPPER0
ATTACH AXIS1 STEPPER1 STEPPER1
AXIS2 OFF
AXIS3 OFF
AXIS4 OFF
AXIS5 OFF
AXIS6 OFF
AXIS7 OFF

AXIS0 PGAIN 0.00017
AXIS0 IGAIN 0.005
AXIS0 ILIMIT 0.01
AXIS0 DGAIN 0.000001

AXIS1 PGAIN 0.0025
AXIS1 IGAIN 0.0
AXIS1 ILIMIT 0.0
PPU X614400 Y80000
ENC0 MULT -4
ENC1 MULT -4
RES X
RES Y
P06160=000000000

PERIOD 0.001
VEL 0.1 ACC 5 DEC 5 STP 5
JOG VEL X2 Y3
JOG ACC X3 Y5
JOG DEC X3 Y5
SET 32
SET 33
10 LV0 = P6916
20 DV0 = DV0
30 DV1 = DV1
40 DV2 = DV2
50 DV3 = P6144
60 DV4 = -1 * P6160
70 DV5 = DV3 / 614400 *360
80 DV6 = DV4 / 40000 *720
90 DV7 = -0.5 * (DV0 * cos(DV5) + DV1 * sin(DV5))
100 DV8 = 0.5 * (DV0 * -1* sin(DV5) + DV1 * cos(DV5))
110 DV10 = -1 * DV5
120 DV9 = DV2 - DV7 * tan(DV10) - DV8 * tan(DV6)
130 DA0(2) = DA0(1)
140 DA1(2) = DA1(1)
150 DA2(2) = DA2(1)
160 DA0(1) = DA0(0)
170 DA1(1) = DA1(0)
180 DA2(1) = DA2(0)
190 DA0(0) = DV9 * cos(DV6) * sin(DV10)  - DV7 / cos(DV10)
200 DA1(0) = DV9 * cos(DV6) * cos(DV10)
210 DA2(0) = DV9 * sin(DV6)  + DV8 / cos(DV6)
220 DA3(0) = (DA0(0) - DA0(1)) / 0.03
230 DA3(1) = (DA1(0) - DA1(1)) / 0.03
240 DA3(2) = (DA2(0) - DA2(1)) / 0.03
250 DA4(0) = (DA0(0) - 2 * DA0(1) + DA0(2)) / (0.03 * 0.03)
```

```
260 DA4(1) = (DA1(0) - 2 * DA1(1) + DA1(2)) / (0.03 * 0.03)
270 DA4(2) = (DA2(0) - 2 * DA2(1) + DA2(2)) / (0.03 * 0.03)
260 DA5(0) = 0.25 * (DA3(0) * 0.03 + 0.5 * DA4(0) * 0.03 * 0.03) + DA0(0)
270 DA5(1) = 0.25 * (DA3(1) * 0.03 + 0.5 * DA4(1) * 0.03 * 0.03) + DA1(0)
280 DA5(2) = 0.25 * (DA3(2) * 0.03 + 0.5 * DA4(2) * 0.03 * 0.03) + DA2(0)

290 DV11 = ATAN(DA5(0) / DA5(1)) / 360
300 DV12 = 0.5 * ATAN(DA5(2)/SQRT(DA5(0) * DA5(0) + DA5(1) * DA5(1)   / 360 - DV4
/40000
310 LV2 = LV2 +1
320 DV13 = SQRT((DV11 - DV3/614400) * (DV11 - DV3/614400)) / 0.066
330 DV14 = SQRT(DV12 * DV12) / 0.066
REM 340 JOG VEL X(1*DV13) Y(1*DV14)
REM 350 JOG ACC X(100*DV13) Y(100*DV14)
REM 360 JOG DEC X(100*DV13) Y(100*DV14)
370 JOG ABS X(-1*DV11)
380 JOG INC Y(-1 * DV12)
390 LV1 = P6916
400 IF ((LV1-LV0) >= 1000) THEN LV3 = LV2
410 IF ((LV1-LV0) >= 1000) THEN LV2 = 0
420 IF (LV2 = 0) THEN LV0 = LV1
430 GOTO 20
DIM DV(15)
DIM LV(4)
DIM DA(6)
DIM DA0(3)
DIM DA1(3)
DIM DA2(3)
DIM DA3(3)
DIM DA4(3)
DIM DA5(3)
```

```
**Acroloop Program for Target Following Control Algorithm, Predictive Velocity
Mode

SYS
HALT ALL
DETACH ALL
CONFIG ENC4 STEPPER4 STEPPER4 NONE

PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ATTACH AXIS0 ENC0 STEPPER0
ATTACH AXIS1 STEPPER1 STEPPER1
AXIS2 OFF
AXIS3 OFF
AXIS4 OFF
AXIS5 OFF
AXIS6 OFF
AXIS7 OFF

AXIS0 PGAIN 0.00017
AXIS0 IGAIN 0.005
AXIS0 ILIMIT 0.01
AXIS0 DGAIN 0.000001

AXIS1 PGAIN 0.0025
AXIS1 IGAIN 0.0
AXIS1 ILIMIT 0.0
PPU X614400 Y80000
ENC0 MULT -4
ENC1 MULT -4
RES X
RES Y
P06160=000000000

PERIOD 0.001
VEL 0.1 ACC 5 DEC 5 STP 5
JOG VEL X2 Y3
JOG ACC X3 Y5
JOG DEC X3 Y5
SET 32
SET 33
10 LV0 = P6916
20 DV0 = DV0
30 DV1 = DV1
40 DV2 = DV2
50 DV3 = P6144
60 DV4 = -1 * P6160
70 DV5 = DV3 / 614400 *360
80 DV6 = DV4 / 40000 *720
90 DV7 = -0.5 * (DV0 * cos(DV5) + DV1 * sin(DV5))
100 DV8 = 0.5 * (DV0 * -1* sin(DV5) + DV1 * cos(DV5))
110 DV10 = -1 * DV5
120 DV9 = DV2 - DV7 * tan(DV10) - DV8 * tan(DV6)
130 DA0(2) = DA0(1)
140 DA1(2) = DA1(1)
150 DA2(2) = DA2(1)
160 DA0(1) = DA0(0)
170 DA1(1) = DA1(0)
180 DA2(1) = DA2(0)
190 DA0(0) = DV9 * cos(DV6) * sin(DV10)  - DV7 / cos(DV10)
200 DA1(0) = DV9 * cos(DV6) * cos(DV10)
210 DA2(0) = DV9 * sin(DV6)  + DV8 / cos(DV6)
220 DA3(0) = (DA0(0) - DA0(1)) / 0.03
230 DA3(1) = (DA1(0) - DA1(1)) / 0.03
240 DA3(2) = (DA2(0) - DA2(1)) / 0.03
250 DA4(0) = (DA0(0) - 2 * DA0(1) + DA0(2)) / (0.03 * 0.03)
```

```
260 DA4(1) = (DA1(0) - 2 * DA1(1) + DA1(2)) / (0.03 * 0.03)
270 DA4(2) = (DA2(0) - 2 * DA2(1) + DA2(2)) / (0.03 * 0.03)
260 DA5(0) = 0.25 * (DA3(0) * 0.03 + 0.5 * DA4(0) * 0.03 * 0.03) + DA0(0)
270 DA5(1) = 0.25 * (DA3(1) * 0.03 + 0.5 * DA4(1) * 0.03 * 0.03) + DA1(0)
280 DA5(2) = 0.25 * (DA3(2) * 0.03 + 0.5 * DA4(2) * 0.03 * 0.03) + DA2(0)

290 DV11 = -1*ATAN(DA5(0) / DA5(1)) / 360 - DV3/614400
300 DV12 = 0.5 * ATAN(DA5(2)/SQRT(DA5(0) * DA5(0) + DA5(1) * DA5(1))) / 360 - DV4
/40000
310 DV13 = SQRT(DV11 * DV11) / 0.1
320 DV14 = SQRT(DV12 * DV12) / 0.1
330 LV2 = LV2 +1
340 JOG VEL X(DV13) Y(DV14)
350 JOG ACC X(1000*DV13) Y(1000*DV14)
360 JOG DEC X(1000*DV13) Y(1000*DV14)
370 IF (DV11 >= 0.0) THEN JOG FWD X
380 IF (DV11 < 0.0) THEN JOG REV X
390 IF (DV12 >= 0.0) THEN JOG REV Y
400 IF (DV12 < 0.0) THEN JOG FWD Y
410 LV1 = P6916
420 IF ((LV1-LV0) >= 1000) THEN LV3 = LV2
430 IF ((LV1-LV0) >= 1000) THEN LV2 = 0
440 IF (LV2 = 0) THEN LV0 = LV1
450 GOTO 20
DIM DV(15)
DIM LV(4)
DIM DA(6)
DIM DA0(3)
DIM DA1(3)
DIM DA2(3)
DIM DA3(3)
DIM DA4(3)
DIM DA5(3)
```

Appendix F

Control programs created for the orientation measurement sub-system

```
orientationDl, cpp : implementation file


nclude "stdafx.h"
nclude "Orientation.h"
nclude "OrientationDlg.h"
nclude <olmem.h>
nclude <olerrors.h>
nclude <oldaapi.h>
nclude "acrolib.h"
nclude "math.h"

fdef _DEBUG
efine new DEBUG_NEW
ndef THIS_FILE
atic char THIS_FILE[] = __FILE__;
ndif

Error handling macros */

efine STRLEN 80         /* string size for general text manipulation   */
r str[STRLEN];          /* global string for general text manipulation */

efine SHOW_ERROR(ecode) AfxMessageBox(olDaGetErrorString(ecode,\
                 str,STRLEN), MB_OK, NULL);

efine CHECKERROR(ecode) if ((board.status = (ecode)) != OLNOERROR)\
                 {\
                 SHOW_ERROR(board.status);\
                 olDaReleaseDASS(board.hdass);\
                 olDaTerminate(board.hdrvr);\
                 exit(1);}

simple structure used with board */

pedef struct tag_board {
 HDEV  hdrvr;          /* driver handle            */
 HDASS hdass;          /* sub system handle        */
 ECODE status;         /* board error status       */
 HBUF  hbuf;           /* sub system buffer handle */
 PWORD lpbuf;          /* buffer pointer           */
 char name[STRLEN];    /* string for board name    */
 char entry[STRLEN];   /* string for board name    */
BOARD;

pedef BOARD* LPBOARD;

atic BOARD board;
LE* FileOCA;
signed long pACRAddress[1];
//////////////////////////////////////////////////////////////////////////////
 CAboutDlg dialog used for App About

ass CAboutDlg : public CDialog

blic:
  CAboutDlg();

 Dialog Data
  //{{AFX_DATA(CAboutDlg)
  enum { IDD = IDD_ABOUTBOX };
  //}}AFX_DATA

  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CAboutDlg)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

 Implementation
otected:
  //{{AFX_MSG(CAboutDlg)
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()


boutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)

  //{{AFX_DATA_INIT(CAboutDlg)
```

```
    //}}AFX_DATA_INIT

id CAboutDlg::DoDataExchange(CDataExchange* pDX)

    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP


GIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
D_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
 COrientationDlg dialog

rientationDlg::COrientationDlg(CWnd* pParent /*=NULL*/)
    : CDialog(COrientationDlg::IDD, pParent)

    //{{AFX_DATA_INIT(COrientationDlg)
    m_uiFreq = 0;
    m_sMax = _T("");
    m_sMin = _T("");
    m_sAveX2 = _T("");
    m_sAveX3 = _T("");
    m_sAveY2 = _T("");
    m_sAveY3 = _T("");
    m_sPx2 = _T("");
    m_sPx3 = _T("");
    m_sPy2 = _T("");
    m_sPy3 = _T("");
    m_sTime = _T("");
    m_sV0 = _T("");
    m_sV1 = _T("");
    m_sV2 = _T("");
    m_sV3 = _T("");
    m_sV4 = _T("");
    m_sV5 = _T("");
    m_sV6 = _T("");
    m_sV7 = _T("");
    m_sPitch = _T("");
    m_sYaw = _T("");
    m_sRoll = _T("");
    m_uiStepSize3 = 0;
    m_uiStepSize4 = 0;
    m_uiEncFreq = 0;
    m_lEnc3 = 0;
    m_lEnc4 = 0;
    m_sAveYaw = _T("");
    m_sAvePitch = _T("");
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);


id COrientationDlg::DoDataExchange(CDataExchange* pDX)

    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(COrientationDlg)
    DDX_Control(pDX, IDC_BUTTON_OCA, m_ctrlButtonOCA);
    DDX_Control(pDX, IDC_BUTTON_WRITE_FILE, m_ctrlWriteFile);
    DDX_Control(pDX, IDC_BUTTON_START_STOP, m_ctrlStartStop);
    DDX_Text(pDX, IDC_EDIT_Freq, m_uiFreq);
    DDX_Text(pDX, IDC_EDIT_MAX, m_sMax);
    DDX_Text(pDX, IDC_EDIT_MIN, m_sMin);
    DDX_Text(pDX, IDC_EDIT_AVE_X2, m_sAveX2);
    DDX_Text(pDX, IDC_EDIT_AVE_X3, m_sAveX3);
    DDX_Text(pDX, IDC_EDIT_AVE_Y2, m_sAveY2);
    DDX_Text(pDX, IDC_EDIT_AVE_Y3, m_sAveY3);
    DDX_Text(pDX, IDC_EDIT_PSDX2, m_sPx2);
    DDX_Text(pDX, IDC_EDIT_PSDX3, m_sPx3);
    DDX_Text(pDX, IDC_EDIT_PSDY2, m_sPy2);
    DDX_Text(pDX, IDC_EDIT_PSDY3, m_sPy3);
    DDX_Text(pDX, IDC_EDIT_TIME, m_sTime);
    DDX_Text(pDX, IDC_EDIT_V0, m_sV0);
    DDX_Text(pDX, IDC_EDIT_V1, m_sV1);
    DDX_Text(pDX, IDC_EDIT_V2, m_sV2);
    DDX_Text(pDX, IDC_EDIT_V3, m_sV3);
    DDX_Text(pDX, IDC_EDIT_V4, m_sV4);
    DDX_Text(pDX, IDC_EDIT_V5, m_sV5);
    DDX_Text(pDX, IDC_EDIT_V6, m_sV6);
    DDX_Text(pDX, IDC_EDIT_V7, m_sV7);
    DDX_Text(pDX, IDC_EDIT_PITCH, m_sPitch);
    DDX_Text(pDX, IDC_EDIT_YAW, m_sYaw);
    DDX_Text(pDX, IDC_EDIT_ROLL, m_sRoll);
    DDX_Text(pDX, IDC_EDIT_M3_STEPSIZE, m_uiStepSize3);
    DDX_Text(pDX, IDC_EDIT_M4_STEPSIZE, m_uiStepSize4);
    DDX_Text(pDX, IDC_EDIT_ENC_FREQ, m_uiEncFreq);
    DDX_Text(pDX, IDC_EDIT_ENC3, m_lEnc3);
    DDX_Text(pDX, IDC_EDIT_ENC4, m_lEnc4);
    DDX_Text(pDX, IDC_EDIT_AVE_YAW, m_sAveYaw);
    DDX_Text(pDX, IDC_EDIT_AVE_PITCH, m_sAvePitch);
    //}}AFX_DATA_MAP

GIN_MESSAGE_MAP(COrientationDlg, CDialog)
    //{{AFX_MSG_MAP(COrientationDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_TIMER()
    ON_BN_CLICKED(IDC_BUTTON_INIT, OnButtonInit)
    ON_BN_CLICKED(IDC_BUTTON_START_STOP, OnButtonStartStop)
    ON_BN_CLICKED(IDC_BUTTON_M3_POSITIVE, OnButtonM3Positive)
    ON_BN_CLICKED(IDC_BUTTON_M3_NEGATIVE, OnButtonM3Negative)
    ON_BN_CLICKED(IDC_BUTTON_M4_POSITIVE, OnButtonM4Positive)
    ON_BN_CLICKED(IDC_BUTTON_M4_NEGATIVE, OnButtonM4Negative)
    ON_BN_CLICKED(IDC_BUTTON_INIT_MOTOR, OnButtonInitMotor)
    ON_EN_SETFOCUS(IDC_EDIT_M3_STEPSIZE, OnSetfocusEditM3Stepsize)
    ON_EN_SETFOCUS(IDC_EDIT_M4_STEPSIZE, OnSetfocusEditM4Stepsize)
    ON_EN_KILLFOCUS(IDC_EDIT_M3_STEPSIZE, OnKillfocusEditM3Stepsize)
    ON_EN_KILLFOCUS(IDC_EDIT_M4_STEPSIZE, OnKillfocusEditM4Stepsize)
    ON_BN_CLICKED(IDC_BUTTON_RESET, OnButtonReset)
    ON_BN_CLICKED(IDC_BUTTON_ZERO, OnButtonZero)
    ON_BN_CLICKED(IDC_BUTTON_WRITE_FILE, OnButtonWriteFile)
    ON_BN_CLICKED(IDC_BUTTON_OCA, OnButtonOca)
    ON_BN_CLICKED(IDC_BUTTON_HOME, OnButtonHome)
    //}}AFX_MSG_MAP
D_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////////
 COrientationDlg message handlers

OL COrientationDlg::OnInitDialog()

    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon

    // TODO: Add extra initialization here
    m_bInitFlag = false;
    m_bSample = false;
    m_bEncoder = false;
    m_bWriteFile = false;
 m_bZero = false;
 m_bStartOCA = false;
 m_bOCASampleThread = false;
 Num = 0;
 SumX2 = 0.0;
 SumY2 = 0.0;
 SumX3 = 0.0;
 SumY3 = 0.0;
 //x_PSD = 0.0;
 //y_PSD = 0.0;

 Motor3.enc = 0;
 Motor3.CurrentAngle = 0.0;

 Motor4.enc = 0;
 Motor4.CurrentAngle = 0.0;

 if ((fp = fopen("Orientation.txt", "w")) == NULL)
   {
       puts("cannot open file");
       exit(1);
   }
 rewind(fp);          // set the cursor to the beginning of file
 fprintf(fp, "V5\t V6\t V7\t V8\t X2\t Y2\t V9\t V10\t V11\t V12\t X3\t Y3\n");

 SetTimer(0x11, 1000, NULL);
 return TRUE;  // return TRUE  unless you set the focus to a control


id COrientationDlg::OnSysCommand(UINT nID, LPARAM lParam)

 if ((nID & 0xFFF0) == IDM_ABOUTBOX)
 {
    CAboutDlg dlgAbout;
    dlgAbout.DoModal();
 }
 else
 {
    CDialog::OnSysCommand(nID, lParam);
 }


 If you add a minimize button to your dialog, you will need the code below
 to draw the icon.  For MFC applications using the document/view model,
 this is automatically done for you by the framework.

id COrientationDlg::OnPaint()

 if (IsIconic())
 {
    CPaintDC dc(this); // device context for painting

    SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

    // Center icon in client rectangle
    int cxIcon = GetSystemMetrics(SM_CXICON);
    int cyIcon = GetSystemMetrics(SM_CYICON);
    CRect rect;
    GetClientRect(&rect);
    int x = (rect.Width() - cxIcon + 1) / 2;
    int y = (rect.Height() - cyIcon + 1) / 2;

    // Draw the icon
    dc.DrawIcon(x, y, m_hIcon);
 }
 else
 {
    CDialog::OnPaint();
 }


 The system calls this to obtain the cursor to display while the user drags
 the minimized window.
URSOR COrientationDlg::OnQueryDragIcon()

 return (HCURSOR) m_hIcon;


OL CALLBACK GetDriver( LPSTR lpszName, LPSTR lpszEntry, LPARAM lParam )
```

```
s is a callback function of olDaEnumBoards, it gets the
ings of the Open Layers board and attempts to initialize
 board.  If successful, enumeration is halted.



LPBOARD lpboard = (LPBOARD)(LPVOID)lParam;

/* fill in board strings */

lstrcpyn(lpboard->name,lpszName,STRLEN);
lstrcpyn(lpboard->entry,lpszEntry,STRLEN);

/* try to open board */

lpboard->status = olDaInitialize(lpszName,&lpboard->hdrvr);
if   (lpboard->hdrvr != NULL)
  return FALSE;           /* false to stop enumerating */
else
  return TRUE;            /* true to continue          */


id COrientationDlg::OnTimer(UINT nIDEvent)

 // TODO: Add your message handler code here and/or call default
 if (nIDEvent == 0x11)
 {
    UpdateData(false);
    m_uiFreq = 0;
    m_uiEncFreq = 0;
    /*if (m_bSample)
    {
        m_ctrlStartStop.SetWindowText("START");
        m_bSample = false;
        //CloseHandle(g_pPSDThread->m_hThread);
    }*/
 }
 CDialog::OnTimer(nIDEvent);


d COrientationDlg::OnButtonInit()

 // TODO: Add your control notification handler code here
 m_bInitFlag = true;
 board.hdrvr = NULL;
 CHECKERROR (olDaEnumBoards(GetDriver,(LPARAM)(LPBOARD)&board));

 /* check for error within callback function */

 CHECKERROR (board.status);

 /* check for NULL driver handle - means no boards */

 if (board.hdrvr == NULL){
    AfxMessageBox(" No Open Layer boards!!!",MB_OK, NULL);
    exit(1);
 }

 /* get handle to A/D sub system */
 CHECKERROR (olDaGetDASS(board.hdrvr,OLSS_AD,0,&board.hdass));

 /* set subsystem for single value operation */
 CHECKERROR (olDaSetDataFlow(board.hdass,OL_DF_SINGLEVALUE));
 CHECKERROR (olDaSetChannelType(board.hdass, OL_CHNT_SINGLEENDED));
 olDaSetRange(board.hdass,10.0,-10.0);
 CHECKERROR (olDaConfig(board.hdass));

 CHECKERROR (olDaGetRange(board.hdass,&max,&min));
 m_sMax.Format("%.3f",max);
 m_sMin.Format("%.3f",min);
 UpdateData(false);
 CHECKERROR (olDaGetEncoding(board.hdass,&encoding));
 CHECKERROR (olDaGetResolution(board.hdass,&resolution));


oid COrientationDlg::OnOK()

 // TODO: Add extra validation here
 if (m_bInitFlag)
 {
    KillTimer(0x11);
    /* release the subsystem and the board */
    CHECKERROR (olDaReleaseDASS(board.hdass));
    CHECKERROR (olDaTerminate(board.hdrvr));
 }
 else
 {
    KillTimer(0x11);
 }
 CDialog::OnOK();


d COrientationDlg::OnButtonStartStop()

 // TODO: Add your control notification handler code here
 if (!m_bSample)
 {
    m_ctrlStartStop.SetWindowText("STOP");
    m_bSample = true;
    THREADPARAMS* pPSDThread = new THREADPARAMS;
    pPSDThread->lParam = (LPARAM) this;
    g_pPSDThread = AfxBeginThread(SamplePSDThread, pPSDThread, THREAD_PRIORITY_NORMAL, NULL

    m_bEncoder = true;
    THREADPARAMS* pEncoderThread = new THREADPARAMS;
    pEncoderThread->lParam = (LPARAM) this;
    g_pEncoderThread = AfxBeginThread(SampleEncoderThread, pEncoderThread, THREAD_PRIORITY_
RMAL, NULL);
 }
 else
 {
    m_ctrlStartStop.SetWindowText("START");
    m_bSample = false;
    CloseHandle(g_pPSDThread->m_hThread);
    m_bEncoder = false;
    CloseHandle(g_pEncoderThread->m_hThread);
 }


NT COrientationDlg::SamplePSDThread(LPVOID pParam)

 THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
 COrientationDlg* pPSDWnd = (COrientationDlg*) pThreadParams->lParam;
 delete pThreadParams;

 float volts0, volts1, volts2, volts3;
 float volts4, volts5, volts6, volts7;
 float RatioX2, RatioY2;
 float RatioX3, RatioY3;
 float AveX2, AveY2;
 float AveX3, AveY3;
 long value0, value1, value2, value3;
 long value4, value5, value6, value7;
 char buffer[256];
 double numerator2, numerator3;
 double denominator2, denominator3;
 SYSTEMTIME st;
 float OffsetX2, OffsetY2;
 float OffsetX3, OffsetY3;
 int DataNo;
 float SumYaw, SumPitch, SumRoll;
 int AveNo;
 float b1y, b1z, b2y, b2z;
 float b1x, b2x;

 OffsetX2 = 0.0;
 OffsetY2 = 0.0;
 OffsetX3 = 0.0;
 OffsetY3 = 0.0;
 DataNo = 0;
 AveNo = 0;
 SumYaw = 0;
 SumPitch = 0;
 SumRoll = 0;
 while (pPSDWnd->m_bSample)
 {
    /* get single value */
    //Temperorary using 12-15
    olDaGetSingleValue(board.hdass,&value3, 15, gain);
    olDaGetSingleValue(board.hdass,&value2, 14, gain);
    olDaGetSingleValue(board.hdass,&value1, 13, gain);
    olDaGetSingleValue(board.hdass,&value0, 12, gain);

    //With interference filter
    volts3 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value3 +
loat)pPSDWnd->min + 0.31325-0.073493+0.00142;
    volts2 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value2 +
loat)pPSDWnd->min - 0.09384+0.02012-0.00504-0.228445-0.001586;
    volts1 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value1 +
loat)pPSDWnd->min + 0.05239-0.014153+0.00294-0.111235+0.00123;
    volts0 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value0 +
loat)pPSDWnd->min - 0.259835+0.02099-0.00413+0.33484-0.001442;

    /* get single value */
    olDaGetSingleValue(board.hdass,&value7, 7, gain);
    olDaGetSingleValue(board.hdass,&value6, 6, gain);
    olDaGetSingleValue(board.hdass,&value5, 5, gain);
    olDaGetSingleValue(board.hdass,&value4, 4, gain);

    //With interference filter
    volts7 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value7 +
loat)pPSDWnd->min +0.1795-0.00406-0.001931;
    volts6 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value6 +
loat)pPSDWnd->min -0.116395+0.02185-0.008029;
    volts5 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value5 +
loat)pPSDWnd->min -0.1225-0.005702+0.00108;
    volts4 = ((float)pPSDWnd->max-(float)pPSDWnd->min)/(1L<<pPSDWnd->resolution) * value4 +
loat)pPSDWnd->min +0.04787+0.00376-0.004403;

    if ((volts1 == 0.0) && (volts3 == 0.0))
    {
       RatioY2 = 0.0;
    }
    else
    {
       RatioY2 = (volts1 - volts3) / (volts1 + volts3);
    }
    if ((volts2 == 0.0) && (volts0 == 0.0))
    {
       RatioX2 = 0.0;
    }
    else
    {
       RatioX2 = (volts2 - volts0) / (volts2 + volts0);
    }
    pPSDWnd->x_PSD2 = 3.6791 * RatioX2 * RatioX2 * RatioX2 + 0.1048 * RatioX2 * RatioX2 + 5
582 * RatioX2 - OffsetX2;
    pPSDWnd->y_PSD2 = 3.2779 * RatioY2 * RatioY2 * RatioY2 + 0.2419 * RatioY2 * RatioY2 + 6
109 * RatioY2 - OffsetY2;
    pPSDWnd->m_sPx2.Format("%.3f", pPSDWnd->x_PSD2);
    pPSDWnd->m_sPy2.Format("%.3f", pPSDWnd->y_PSD2);
    pPSDWnd->m_sV0.Format("%.3f", volts0);
    pPSDWnd->m_sV1.Format("%.3f", volts1);
    pPSDWnd->m_sV2.Format("%.3f", volts2);
    pPSDWnd->m_sV3.Format("%.3f", volts3);

    if ((volts5 == 0.0) && (volts7 == 0.0))
    {
       RatioY3 = 0.0;
    }
    else
    {
       RatioY3 = -1.0 * (volts5 - volts7) / (volts5 + volts7);
    }
    if ((volts6 == 0.0) && (volts4 == 0.0))
    {
       RatioX3 = 0.0;
    }
    else
    {
       RatioX3 = -1.0 * (volts6 - volts4) / (volts6 + volts4);
    }
    pPSDWnd->x_PSD3 = 2.6272 * RatioX3 * RatioX3 * RatioX3 + 0.1193 * RatioX3 * RatioX3 + 6
4 * RatioX3  - OffsetX3;
    pPSDWnd->y_PSD3 = 2.5995 * RatioY3 * RatioY3 * RatioY3 + 0.0682 * RatioY3 * RatioY3 + 6
592 * RatioY3 - OffsetY3;
```

```
    pPSDWnd->m_sPx3.Format("%.3f", pPSDWnd->x_PSD3);
    pPSDWnd->m_sPy3.Format("%.3f", pPSDWnd->y_PSD3);
    pPSDWnd->m_sV4.Format("%.3f", volts4);
    pPSDWnd->m_sV5.Format("%.3f", volts5);
    pPSDWnd->m_sV6.Format("%.3f", volts6);
    pPSDWnd->m_sV7.Format("%.3f", volts7);

    ::GetSystemTime(&st);
    pPSDWnd->m_sTime.Format("%d", (st.wMinute * 60000 + st.wSecond * 1000 + st.wMillisecond


    if (pPSDWnd->Num < 2000)
    {
        pPSDWnd->SumX2 += pPSDWnd->x_PSD2;
        pPSDWnd->SumY2 += pPSDWnd->y_PSD2;
        pPSDWnd->SumX3 += pPSDWnd->x_PSD3;
        pPSDWnd->SumY3 += pPSDWnd->y_PSD3;
        pPSDWnd->Num++;
    }
    else
    {
        AveX2 = pPSDWnd->SumX2 / pPSDWnd->Num;
        AveY2 = pPSDWnd->SumY2 / pPSDWnd->Num;
        pPSDWnd->m_sAveX2.Format("%.3f", AveX2);
        pPSDWnd->m_sAveY2.Format("%.3f", AveY2);
        AveX3 = pPSDWnd->SumX3 / pPSDWnd->Num;
        AveY3 = pPSDWnd->SumY3 / pPSDWnd->Num;
        pPSDWnd->m_sAveX3.Format("%.3f", AveX3);
        pPSDWnd->m_sAveY3.Format("%.3f", AveY3);

        pPSDWnd->Num = 0;
        pPSDWnd->SumX2 = 0.0;
        pPSDWnd->SumY2 = 0.0;
        pPSDWnd->SumX3 = 0.0;
        pPSDWnd->SumY3 = 0.0;
    }

    if (pPSDWnd->m_bZero)
    {
        OffsetX2 = 0.0;//AveX2;
        OffsetX3 = 0.0;//AveX3;
        OffsetY2 = AveY2;
        OffsetY3 = AveY3;
        pPSDWnd->m_bZero = false;
    }


    numerator2 = 1.2114 * (pPSDWnd->x_PSD2 - pPSDWnd->x_PSD3);
    denominator2 = 13 -12 - 2.0 * lc;
    pPSDWnd->yaw = atan(numerator2 / denominator2);

    numerator3 = 1.217 * (pPSDWnd->y_PSD2 + pPSDWnd->y_PSD3) * cos(pPSDWnd->yaw);
    denominator3 = 13 -12 - 2.0 * lc;
    pPSDWnd->pitch = atan(numerator3 / denominator3);

    b1x = -1.0 * AveX2 + (12-lc) * tan(pPSDWnd->yaw);
    b1y = 0.0;
    b1z = AveY2 - (12-lc) * tan(pPSDWnd->pitch) / cos(pPSDWnd->yaw);
    b2x = b1x * cos(pPSDWnd->yaw) + b1z * sin(pPSDWnd->yaw) * sin(pPSDWnd->pitch);
    b2y = 0.0; //sin(pPSDWnd->yaw) * cos(pPSDWnd->pitch) * b1x + cos(pPSDWnd->yaw) * b1y +
n(pPSDWnd->yaw) * sin(pPSDWnd->pitch) * b1z;
    b2z = cos(pPSDWnd->pitch) * b1z;
    pPSDWnd->roll = atan(b2z/b2x);

    if (AveNo < 100)
    {
        SumYaw += pPSDWnd->yaw;
        SumPitch += pPSDWnd->pitch;
        SumRoll += pPSDWnd->roll;
        AveNo ++;
    }
    else
    {
        pPSDWnd->AveYaw = SumYaw / AveNo;
        pPSDWnd->AvePitch = SumPitch / AveNo;
        pPSDWnd->AveRoll = SumRoll / AveNo;
        AveNo = 0;
        SumYaw = 0.0;
        SumPitch = 0.0;
        SumRoll = 0.0;
    }

    pPSDWnd->m_sYaw.Format("%.3f", pPSDWnd->yaw / pi * 180.0);
    pPSDWnd->m_sPitch.Format("%.3f", pPSDWnd->pitch / pi * 180.0);
    pPSDWnd->m_sRoll.Format("%.3f", pPSDWnd->roll / pi * 180.0);
    pPSDWnd->m_sAveYaw.Format("%.3f", pPSDWnd->AveYaw / pi * 180.0);
    pPSDWnd->m_sAvePitch.Format("%.3f", pPSDWnd->AvePitch / pi * 180.0);

    if (pPSDWnd->m_bWriteFile)
    {
        ::sprintf(buffer, "%i\t %s\t %.3f\t %.3f\t %.3f\t %.3f\t %.3f\t %.3f\t %.3f\t %.3f\
.3f\t %.3f\t %.3f\t%.3f\t %.3f\t %.3f\t %.3f\t %.3lf\t %.3lf\n", DataNo, pPSDWnd->m_sTime, v
s0, volts1, volts2, volts3, pPSDWnd->x_PSD2, pPSDWnd->y_PSD2, volts4, volts5, volts6, volts7
pSDWnd->x_PSD3, pPSDWnd->y_PSD3, pPSDWnd->yaw/ pi * 180.0, pPSDWnd->pitch/ pi * 180.0, pPSDW
>roll/ pi * 180.0, pPSDWnd->Motor3.CurrentAngle, pPSDWnd->Motor4.CurrentAngle);
        ::fprintf(pPSDWnd->fp, buffer);
        //::sprintf(buffer, "%s\t %.3f\t %.3f\t %.3lf\t %.3lf\n", pPSDWnd->m_sTime, AveYaw/
 * 180.0, AvePitch/ pi * 180.0, pPSDWnd->Motor3.CurrentAngle, pPSDWnd->Motor4.CurrentAngle);
        //::fprintf(pPSDWnd->fp, buffer);
        DataNo++;
        if (DataNo>100)
        {
            pPSDWnd->m_bWriteFile = false;
            DataNo = 0;
            pPSDWnd->m_ctrlWriteFile.SetWindowText("WriteToFile");
        }
    }

    pPSDWnd->m_uiFreq++;
 }
 return 0;

T COrientationDlg::SampleEncoderThread(LPVOID pParam)

 THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
 COrientationDlg* pEncoderWnd = (COrientationDlg*) pThreadParams->lParam;
 delete pThreadParams;
 long pPosParameter[2];

 while (pEncoderWnd->m_bEncoder)
 {
    A8_BIN_GROUP_GETLONG(0x18, 0x00, 0x0C, pPosParameter, 0);
    //A8_BIN_GROUP_GETLONG(0x30, 0x09, 0x03, pPosParameter, 0);
    pEncoderWnd->Motor3.enc = pPosParameter[0];
    pEncoderWnd->Motor4.enc = pPosParameter[1];
    pEncoderWnd->Motor3.CurrentAngle = (float(pEncoderWnd->Motor3.enc) /4000.0 / 45.0 * 360
);     //Angle in Degree
    pEncoderWnd->Motor4.CurrentAngle = (float(pEncoderWnd->Motor4.enc) /2000.0 * 360.0);
  //Angle in Degree
    pEncoderWnd->m_lEnc3 = pEncoderWnd->Motor3.enc;
    pEncoderWnd->m_lEnc4 = pEncoderWnd->Motor4.enc;
    pEncoderWnd->m_uiEncFreq++;
 }
 return 0;

id COrientationDlg::OnButtonM3Positive()

 // TODO: Add your control notification handler code here
 Motor3.DriveAngle = -1.0 * double(m_uiStepSize3);
 Motor4.DriveAngle = 0.0;
 DriveMotor(Motor3, Motor4);


id COrientationDlg::OnButtonM3Negative()

 // TODO: Add your control notification handler code here
 Motor3.DriveAngle = double(m_uiStepSize3);
 Motor4.DriveAngle = 0.0;
 DriveMotor(Motor3, Motor4);


id COrientationDlg::OnButtonM4Positive()

 // TODO: Add your control notification handler code here
 Motor3.DriveAngle = 0.0;
 Motor4.DriveAngle = double(m_uiStepSize4);
 DriveMotor(Motor3, Motor4);
```

```
d COrientationDlg::OnButtonM4Negative()

 // TODO: Add your control notification handler code here
 Motor3.DriveAngle = 0.0;
 Motor4.DriveAngle = -1.0 * double(m_uiStepSize4);
 DriveMotor(Motor3, Motor4);


d COrientationDlg::DriveMotor(CMotor MotorData3, CMotor MotorData4)

 char cmdline[256];

 //if (fabs(MotorData3.DriveAngle) < 25600)
 //{
    //MotorData2.DriveAngle = -1 * MotorData2.DriveAngle;
    //Increment mode used in jogging
    sprintf(cmdline, "JOG INC Z%lf A%lf", MotorData3.DriveAngle, MotorData4.DriveAngle);
    AcroSendString(cmdline, 0);
    return;
 //}
 //else
 //{
 //  MessageBeep(0);
 //  AfxMessageBox("DANGER!! MOTOR 1 trajectory is out of range", MB_ICONEXCLAMATION);
 //}


id COrientationDlg::OnButtonInitMotor()

 // TODO: Add your control notification handler code here

 AcroInitialize(0);
 if (AcroGetError()!=ACRO_SUCCESS)
 {
    MessageBeep(0);
    AfxMessageBox("Card Driver Not Started", MB_ICONEXCLAMATION);
    exit(0);
 }
 AcroSendString("P06176 = 0", 0);
 AcroSendString("P06192 = 0", 0);
 A8_BIN_ADDRESS(0, 0x00, pACRAddress, 0);


id COrientationDlg::OnSetfocusEditM3Stepsize()

 // TODO: Add your control notification handler code here
 KillTimer(0x11);


id COrientationDlg::OnSetfocusEditM4Stepsize()

 // TODO: Add your control notification handler code here
 KillTimer(0x11);



id COrientationDlg::OnKillfocusEditM3Stepsize()

 // TODO: Add your control notification handler code here
 UpdateData(true);
 SetTimer(0x11, 1000, NULL);


id COrientationDlg::OnKillfocusEditM4Stepsize()

 // TODO: Add your control notification handler code here
 UpdateData(true);
 SetTimer(0x11, 1000, NULL);


id COrientationDlg::OnButtonReset()

 // TODO: Add your control notification handler code here
 AcroSendString("P06176 = 0", 0);
 AcroSendString("P06192 = 0", 0);
 AcroSendString("RES Z", 0);
 AcroSendString("RES A", 0);


d COrientationDlg::OnButtonZero()

 // TODO: Add your control notification handler code here
 m_bZero = true;


d COrientationDlg::OnButtonWriteFile()

 // TODO: Add your control notification handler code here
 if (!m_bWriteFile)
 {
    m_ctrlWriteFile.SetWindowText("Stop Write");
    m_bWriteFile = true;
 }
 else
 {
    m_ctrlWriteFile.SetWindowText("WriteToFile");
    m_bWriteFile = false;
 }


d COrientationDlg::OnButtonOca()

 // TODO: Add your control notification handler code here
 char buffer[256];
 float pSendIEEEOCA[2];
 if (!m_bStartOCA)
 {
    if ((FileOCA = fopen("OCA.txt", "w")) == NULL)
    {
       puts("cannot open file");
       exit(1);
    }
    rewind(FileOCA);              // set the cursor to the beginning of file
    fprintf(FileOCA, "Time\tRoll\tPitch\tYaw\t\n");
    sprintf(buffer, "%s\t%.3lf\t%.3lf\t%.3lf\t\n", m_sTime, RadianToDegree(AveRoll), Radian
Degree(AvePitch), RadianToDegree(AveYaw));
    fprintf(FileOCA, buffer);
    m_bStartOCA = true;
    m_ctrlButtonOCA.SetWindowText("STOP OCA");

    AcroSendString("RUN", 0);
    m_bOCASampleThread = true;
    THREADPARAMS* pOCASampleThread = new THREADPARAMS;
    pOCASampleThread->lParam = (LPARAM) this;
    g_pOCASampleThread = AfxBeginThread(OCASampleThread, pOCASampleThread, THREAD_PRIORITY_
RMAL, NULL);
    Motor3.DriveAngle = -1.0 * (AveYaw / pi * 180.0) / 360.0 * 50800.0 * 45.0;
    Motor4.DriveAngle = -1.0 * (AvePitch / pi * 180.0) / 360.0 * 51200.0;
    //DriveMotor(Motor3, Motor4);
    pSendIEEEOCA[0] = Motor3.DriveAngle;
    pSendIEEEOCA[1] = Motor4.DriveAngle;
    A8_BIN_POKE_IEEE(0x01, 2, pACRAddress[0] + 1, pSendIEEEOCA, 0);
    //AcroSendString("RUN", 0);
 }
 else
 {
    AcroSendString("HALT", 0);
    m_bOCASampleThread = false;
    CloseHandle(g_pOCASampleThread->m_hThread);
    m_bStartOCA = false;
    m_ctrlButtonOCA.SetWindowText("START OCA");
    fclose(FileOCA);
 }



NT COrientationDlg::OCASampleThread(LPVOID pParam)

 THREADPARAMS* pThreadParams = (THREADPARAMS*) pParam;
 COrientationDlg* pOCAWnd = (COrientationDlg*) pThreadParams->lParam;
 delete pThreadParams;
 char buffer[256];
 float pSendIEEEOCA[2];

 while (pOCAWnd->m_bOCASampleThread)
 {
    sprintf(buffer, "%s\t%.3lf\t%.3lf\t%.3lf\t\n", pOCAWnd->m_sTime, RadianToDegree(pOCAWnd
veRoll), RadianToDegree(pOCAWnd->AvePitch), RadianToDegree(pOCAWnd->AveYaw));
    fprintf(FileOCA, buffer);
    pOCAWnd->Motor3.DriveAngle = -1.0 * (pOCAWnd->AveYaw / pi * 180.0) / 360.0 * 50800.0 *
0;
    pOCAWnd->Motor4.DriveAngle = -1.0 * (pOCAWnd->AvePitch / pi * 180.0) / 360.0 * 51200.0;
    pSendIEEEOCA[0] = pOCAWnd->Motor3.DriveAngle;
    pSendIEEEOCA[1] = pOCAWnd->Motor4.DriveAngle;
    A8_BIN_POKE_IEEE(0x01, 2, pACRAddress[0] + 1, pSendIEEEOCA, 0);
    ::Sleep(300);
 }
 return 0;

d COrientationDlg::OnButtonHome()

 // TODO: Add your control notification handler code here
 AcroSendString("JOG ABS Z0 A0", 0);
```

```
OrientationDlg.h : header file


!defined(AFX_ORIENTATIONDLG_H__249B3157_A707_11D7_9C89_000021CC063C__INCLUDED_)
efine AFX_ORIENTATIONDLG_H__249B3157_A707_11D7_9C89_000021CC063C__INCLUDED_

f _MSC_VER > 1000
ragma once
ndif // _MSC_VER > 1000

ass CMotor

blic:
 double DriveAngle;
 double CurrentAngle;
 long enc;
 long bb_enc;


nst double l2 = 56.1392;
nst double l3 = 56.1392;
nst double lc = 60.0;
nst double pi = 3.1415926535;
//////////////////////////////////////////////////////////////////////////////
COrientationDlg dialog

ass COrientationDlg : public CDialog

Construction
blic:
 COrientationDlg(CWnd* pParent = NULL);  // standard constructor
 static UINT SamplePSDThread(LPVOID);
 static UINT SampleEncoderThread(LPVOID);
 static UINT OCASampleThread(LPVOID);
 FILE* fp;
 double min,max;
 UINT encoding,resolution;
 bool m_bInitFlag;
 bool m_bSample;
 double x_PSD2, y_PSD2;
 double x_PSD3, y_PSD3;
 bool m_bWriteFile;

 int Num;
 float SumX2, SumY2;
 float SumX3, SumY3;

 void DriveMotor(CMotor, CMotor);
 CMotor Motor3, Motor4;
 bool m_bEncoder;
 bool m_bZero;
 bool m_bStartOCA;
 bool m_bOCASampleThread;
 double roll, yaw, pitch;
 float AveRoll, AveYaw, AvePitch;
Dialog Data
 //{{AFX_DATA(COrientationDlg)
 enum { IDD = IDD_ORIENTATION_DIALOG };
 CButton m_ctrlButtonOCA;
 CButton m_ctrlWriteFile;
 CButton m_ctrlStartStop;
 UINT    m_uiFreq;
 CString m_sMax;
 CString m_sMin;
 CString m_sAveX2;
 CString m_sAveX3;
 CString m_sAveY2;
 CString m_sAveY3;
 CString m_sPx2;
 CString m_sPx3;
 CString m_sPy2;
 CString m_sPy3;
 CString m_sTime;
 CString m_sV0;
 CString m_sV1;
 CString m_sV2;
 CString m_sV3;
 CString m_sV4;
 CString m_sV5;
CString m_sV6;
CString m_sV7;
CString m_sPitch;
CString m_sYaw;
CString m_sRoll;
UINT    m_uiStepSize3;
UINT    m_uiStepSize4;
UINT    m_uiEncFreq;
long    m_lEnc3;
long    m_lEnc4;
CString m_sAveYaw;
CString m_sAvePitch;
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(COrientationDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

Implementation
otected:
HICON m_hIcon;

// Generated message map functions
//{{AFX_MSG(COrientationDlg)
virtual BOOL OnInitDialog();
afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
afx_msg void OnTimer(UINT nIDEvent);
afx_msg void OnButtonInit();
virtual void OnOK();
afx_msg void OnButtonStartStop();
afx_msg void OnButtonM3Positive();
afx_msg void OnButtonM3Negative();
afx_msg void OnButtonM4Positive();
afx_msg void OnButtonM4Negative();
afx_msg void OnButtonInitMotor();
afx_msg void OnSetfocusEditM3Stepsize();
afx_msg void OnSetfocusEditM4Stepsize();
afx_msg void OnKillfocusEditM3Stepsize();
afx_msg void OnKillfocusEditM4Stepsize();
afx_msg void OnButtonReset();
afx_msg void OnButtonZero();
afx_msg void OnButtonWriteFile();
afx_msg void OnButtonOca();
afx_msg void OnButtonHome();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()


/{{AFX_INSERT_LOCATION}}
/ Microsoft Visual C++ will insert additional declarations immediately before the previous lin


ndif // !defined(AFX_ORIENTATIONDLG_H__249B3157_A707_11D7_9C89_000021CC063C__INCLUDED_)
```

```
Orientation.cpp : Defines the class behaviors for the application.


nclude "stdafx.h"
nclude "Orientation.h"
nclude "OrientationDlg.h"

fdef _DEBUG
efine new DEBUG_NEW
ndef THIS_FILE
atic char THIS_FILE[] = __FILE__;
ndif

//////////////////////////////////////////////////////////////////////////////
COrientationApp

GIN MESSAGE_MAP(COrientationApp, CWinApp)
  //{{AFX_MSG_MAP(COrientationApp)
      // NOTE - the ClassWizard will add and remove mapping macros here.
      //    DO NOT EDIT what you see in these blocks of generated code!
  //}}AFX_MSG
  ON_COMMAND(ID_HELP, CWinApp::OnHelp)
D_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////////
COrientationApp construction

rientationApp::COrientationApp()

  // TODO: add construction code here,
  // Place all significant initialization in InitInstance


//////////////////////////////////////////////////////////////////////////////
The one and only COrientationApp object

rientationApp theApp;

//////////////////////////////////////////////////////////////////////////////
COrientationApp initialization

OL COrientationApp::InitInstance()

  AfxEnableControlContainer();

  // Standard initialization
  // If you are not using these features and wish to reduce the size
  //  of your final executable, you should remove from the following
  //  the specific initialization routines you do not need.

fdef _AFXDLL
  Enable3dControls();          // Call this when using MFC in a shared DLL
lse
  Enable3dControlsStatic();    // Call this when linking to MFC statically
ndif

  COrientationDlg dlg;
  m_pMainWnd = &dlg;
  int nResponse = dlg.DoModal();
  if (nResponse == IDOK)
  {
      // TODO: Place code here to handle when the dialog is
      //  dismissed with OK
  }
  else if (nResponse == IDCANCEL)
  {
      // TODO: Place code here to handle when the dialog is
      //  dismissed with Cancel
  }

  // Since the dialog has been closed, return FALSE so that we exit the
  //  application, rather than start the application's message pump.
  return FALSE;
```

```
/ Orientation.h : main header file for the ORIENTATION application
/

f !defined(AFX_ORIENTATION_H__249B3155_A707_11D7_9C89_000021CC063C__INCLUDED_)
efine AFX_ORIENTATION_H__249B3155_A707_11D7_9C89_000021CC063C__INCLUDED_

f _MSC_VER > 1000
ragma once
ndif // _MSC_VER > 1000

fndef __AFXWIN_H__
 #error include 'stdafx.h' before including this file for PCH
ndif

nclude "resource.h"       // main symbols

////////////////////////////////////////////////////////////////////////////
/ COrientationApp:
/ See Orientation.cpp for the implementation of this class
/

lass COrientationApp : public CWinApp

blic:
 COrientationApp();

/ Overrides
 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(COrientationApp)
 public:
 virtual BOOL InitInstance();
 //}}AFX_VIRTUAL

/ Implementation

 //{{AFX_MSG(COrientationApp)
    // NOTE - the ClassWizard will add and remove member functions here.
    //    DO NOT EDIT what you see in these blocks of generated code !
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()


////////////////////////////////////////////////////////////////////////////

/{{AFX_INSERT_LOCATION}}
/ Microsoft Visual C++ will insert additional declarations immediately before the previous lin

ndif // !defined(AFX_ORIENTATION_H__249B3155_A707_11D7_9C89_000021CC063C__INCLUDED_)
```

```
stdafx.cpp : source file that includes just the standard includes
 Orientation.pch will be the pre-compiled header
 stdafx.obj will contain the pre-compiled type information

nclude "stdafx.h"
```

```
/ stdafx.h : include file for standard system include files,
/ or project specific include files that are used frequently, but
/      are changed infrequently
/

f !defined(AFX_STDAFX_H__249B3159_A707_11D7_9C89_000021CC063C__INCLUDED_)
efine AFX_STDAFX_H__249B3159_A707_11D7_9C89_000021CC063C__INCLUDED_

f _MSC_VER > 1000
ragma once
ndif // _MSC_VER > 1000

efine VC_EXTRALEAN          // Exclude rarely-used stuff from Windows headers

nclude <afxwin.h>           // MFC core and standard components
nclude <afxext.h>           // MFC extensions
nclude <afxdisp.h>          // MFC Automation classes
nclude <afxdtctl.h>         // MFC support for Internet Explorer 4 Common Controls
fndef _AFX_NO_AFXCMN_SUPPORT
nclude <afxcmn.h>           // MFC support for Windows Common Controls
ndif // _AFX_NO_AFXCMN_SUPPORT

nst gain = 1;
atic CWinThread*  g_pPSDThread;
atic CWinThread*  g_pEncoderThread;
atic CWinThread*  g_pOCASampleThread;
efine RadianToDegree(radian) (57.29577951308 * radian)
/Structure to store parameter for thread function
pedef struct tagTHREADPARAMS

 LPARAM lParam;
THREADPARAMS;
/{{AFX_INSERT_LOCATION}}
/Microsoft Visual C++ will insert additional declarations immediately before the previous lin


ndif // !defined(AFX_STDAFX_H__249B3159_A707_11D7_9C89_000021CC063C__INCLUDED_)
```

```
/{{NO_DEPENDENCIES}}
/Microsoft Developer Studio generated include file.
/Used by Orientation.rc
/
define IDM_ABOUTBOX                    0x0010
define IDD_ABOUTBOX                    100
define IDS_ABOUTBOX                    101
define IDD_ORIENTATION_DIALOG          102
define IDR_MAINFRAME                   128
define IDC_EDIT_V0                     1000
define IDC_EDIT_Freq                   1001
define IDC_EDIT_V1                     1002
define IDC_EDIT_V2                     1003
define IDC_EDIT_V3                     1004
define IDC_EDIT_MIN                    1005
define IDC_EDIT_MAX                    1006
define IDC_EDIT_PSDX2                  1007
define IDC_EDIT_PSDY2                  1008
define IDC_BUTTON_INIT                 1009
define IDC_BUTTON_START_STOP           1010
define IDC_EDIT_TIME                   1011
define IDC_EDIT_V4                     1012
define IDC_EDIT_AVE_X2                 1013
define IDC_EDIT_AVE_Y2                 1014
define IDC_EDIT_V5                     1015
define IDC_EDIT_V6                     1016
define IDC_EDIT_V7                     1017
define IDC_STATIC_Ch0                  1018
define IDC_STATIC_Ch1                  1019
define IDC_STATIC_Ch2                  1020
define IDC_STATIC_Ch3                  1021
define IDC_STATIC_Ch4                  1022
define IDC_STATIC_Ch5                  1023
define IDC_STATIC_Ch6                  1024
define IDC_STATIC_Ch7                  1025
define IDC_EDIT_PSDX3                  1026
define IDC_EDIT_PSDY3                  1027
define IDC_EDIT_AVE_X3                 1028
define IDC_EDIT_AVE_Y3                 1029
define IDC_EDIT_YAW                    1030
define IDC_EDIT_PITCH                  1031
define IDC_EDIT_ROLL                   1032
define IDC_BUTTON_M3_POSITIVE          1033
define IDC_BUTTON_M3_NEGATIVE          1034
define IDC_BUTTON_M4_POSITIVE          1035
define IDC_BUTTON_M4_NEGATIVE          1036
define IDC_EDIT_M3_STEPSIZE            1037
define IDC_EDIT_M4_STEPSIZE            1038
define IDC_EDIT_ENC3                   1039
define IDC_EDIT_ENC4                   1040
define IDC_BUTTON_INIT_MOTOR           1041
define IDC_EDIT_ENC_FREQ               1042
define IDC_BUTTON_RESET                1043
define IDC_BUTTON_ZERO                 1044
define IDC_BUTTON_WRITE_FILE           1045
define IDC_EDIT_AVE_YAW                1046
define IDC_EDIT_AVE_PITCH              1047
define IDC_EDIT_AVE_ROLL               1048
define IDC_BUTTON_OCA                  1049
define IDC_BUTTON_HOME                 1050

/ Next default values for new objects
/
fdef APSTUDIO_INVOKED
fndef APSTUDIO_READONLY_SYMBOLS
define _APS_NEXT_RESOURCE_VALUE        129
define _APS_NEXT_COMMAND_VALUE         32771
define _APS_NEXT_CONTROL_VALUE         1051
define _APS_NEXT_SYMED_VALUE           101
ndif
ndif
```

```
**Acroloop Program for Orientation Compensation Algorithm

HALT ALL
DETACH ALL
CONFIG ENC4 STEPPER4 STEPPER4 NONE

PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "Z"
ATTACH SLAVE1 AXIS1 "A"
ATTACH AXIS0 STEPPER2 STEPPER2
ATTACH AXIS1 STEPPER3 STEPPER3
AXIS4 OFF
AXIS5 OFF
AXIS6 OFF
AXIS7 OFF
AXIS0 PGAIN 0.0025
AXIS1 PGAIN 0.0025
AXIS0 IGAIN 0
AXIS1 IGAIN 0
AXIS0 ILIMIT 0
AXIS1 ILIMIT 0
PPU Z1 A1
ENC2 MULT -4
ENC3 MULT -4
VEL 100000 ACC100000 DEC100000  STP100000
JOG VEL Z 100000 A 3000
JOG ACC Z 100000 A 3000
JOG DEC Z 100000 A 3000

10 DV0 = DV0
20 DV1 = DV1
30 JOG INC Z(DV0) A(DV1)
REM 40 DV0 = 0
REM 50 DV1 = 0
60 GOTO 10
DIM DV(2)
```

Appendix G

Publications

B. Shirinzadeh, P. L. Teoh, C. W. Foong, "Orientation measurement using vision and non-vision based techniques in laser tracking system", 30th International Symposium on Robotics, Tokyo, Japan, pp. 317-324, 1999.

Orientation Measurement Using Vision and Non-Vision Based Techniques in Laser Tracking System

Bijan SHIRINZADEH, Pek Loo TEOH and Chee Wei FOONG

Department of Mechanical Engineering
Monash University
Clayton, VIC 3168
Australia
Email: bijan.shirinzadeh@eng.monash.edu.au

Abstract:

Most applications of robots involve the interaction between the robot's end-effector and the objects in the physical environment. These require accurate placement of robot's end-effector along a specific path. To accurately define a pose (position and orientation) of a manipulator's end-effector, measurement of six parameters is required - three for position and three for orientation. Different approaches have been studied over the last few years to measure the orientation of the end-effector dynamically and precisely. However, there are still some difficulties in determining the orientation of the end-effector in real time. In this paper, two methods of orientation measurement in Laser Interferometry-based Tracking System (LITS) will be described. These include vision and non-vision based techniques to measure the orientation. The algorithms of both approaches will be presented. Both techniques will be compared to determine their efficiencies and limitations.

Keywords: Laser Interferometry-based Tracking, Orientation measurement

1. INTRODUCTION

The application areas for robots in manufacturing and service industries are increasing. These application areas are generally more complex, requiring higher robot accuracy and advanced flexible programming techniques. Laser interferometry-based tracking system (LITS) can be used in dynamic performance measurements of position and orientation as well as providing measurements for robot calibration. As the traditional methods of pose measurement are very time-consuming and expensive, a measurement tool with high accuracy, a large working space, a high sampling rate and automatic target tracking is desirable. These specifications are currently only met by a LITS [1].

The performance of a LITS depends on various factors. These include the sampling rate of the sub-system responsible for offset error measurement, the sampling rate of the motion controller card, the accuracy or resolution of different sub-systems and also the laser beam power. The inefficiency of one of these factors will consequently become the limiting factor to the system and thus reduces the performance.

In this paper, the sub-system being studied is the orientation measurement sub-system. Two methods of orientation measurement will be described [2, 3, 4]. These include vision and non-vision based techniques to measure the orientation in real time. The experimental setup and the algorithm for both techniques will also be presented. Both techniques will be compared in terms of the performance and efficiency by looking at the accuracy of these techniques and their limitations.

2. PRINCIPLE OF LASER INTERFEROMETRY-BASED TRACKING SYSTEM FOR POSITION MEASUREMENT

The LITS developed for this study is shown in Figure 1 [5]. The laser beam emitted from the HeNe laser passes through an interferometer and is split into a

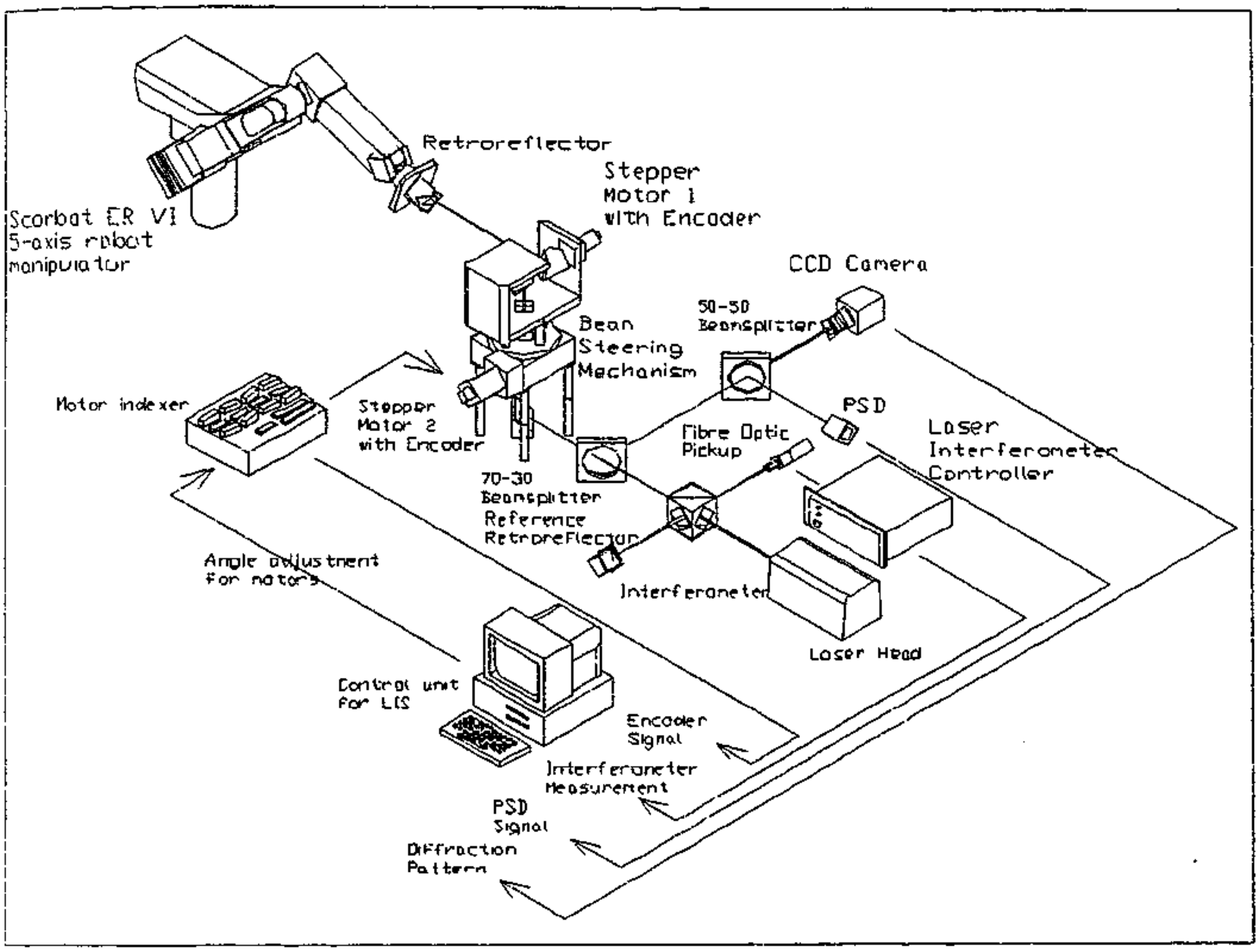


Figure 1: Setup of Laser Interferometry-based Tracking System

reference beam and a measurement beam. The reference beam will be picked up by a fibre optic cable and be transmitted to the measurement board on the laser controller. The measurement beam passes through a 70-30 beamsplitter before being directed by the beam steering mechanism onto the retroreflector mounted on the end-effector of the robot manipulator. This beam is referred to as the incident beam.

The retroreflector is designed in such a way that the reflected beam is always parallel to the incident beam. This reflected beam is again directed by the beam steering mechanism back to the 70-30 beamsplitter. 70% of the reflected beam will pass through the beamsplitter and travel to the interferometer before being picked up by the fibre optic cable. It is then transmitted and combined with the reference beam to determine the displacement of the laser beam (i.e. the distance of the centre of retroreflector from the interferometer) using Doppler Shift. The remaining 30% of the reflected beam is directed perpendicularly to the measurement beam and travels through a 50-50 beamsplitter. 50% of this beam is being incident onto a position sensitive diode (PSD), which determines the tracking error of the beam from the centre position of the retroreflector. The other 50% will be transmitted to a CCD (Charged-Coupled Device) camera for orientation measurement.

The beam steering mechanism consists of three 45° beam-steers, with one being stationary, and the other two driven by two stepper motors - one for vertical axis and the other for horizontal axis. Continuos tracking of the retroreflector are possible by controlling the motors to rotate by the corrective angles determined from the kinematics of the system using tracking error obtained from the PSD. This provides continuos tracking and update of the end-effector's pose in real time.

3. VISION BASED ORIENTATION MEASUREMENT

Originally, the vision based orientation measurement was proposed by Vincze, *et. al.* [3]. As mentioned in the previous section, part of the reflected measurement beam from the retroreflector is directed to the CCD camera. The camera will capture the image of the diffraction pattern of the retroreflector and a high-acquisition-rate frame grabber will acquire this signal to perform image digitisation. Figure 2 shows the image of the diffraction as displayed on a computer screen. It consists of a white circle on a dark background, with 3 dark lines intersecting each other. The intersection point is the centre point of the retroreflector and the dark lines are caused by the edges of the three mirrors on the retroreflector. The image can be simplified as shown in Figure 3. The vectors V_i (i = 1, 2, 3) are the projected edge vectors of the diffraction pattern. Blob analysis is performed on the digitised image before Vectors Detection process can be carried out.

3.1 Blob Analysis and Vectors Detection

The vectors V_i in Figure 2 are noticeable to the human eyes but the software program does not readily recognise these. Additional steps are required to inform the control software about the presence of the vectors. Figure 4 shows the schematic diagram of the strategy of the analysis.

Figure 2: Diffraction pattern of the reflected beam from the retroreflector

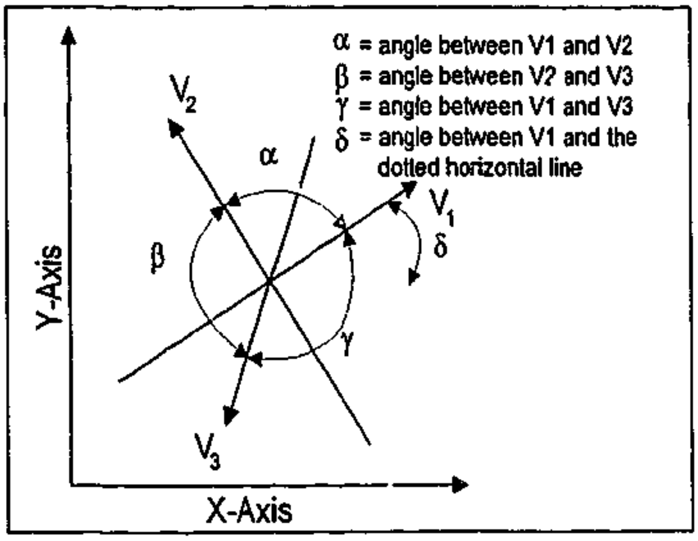

α = angle between V1 and V2
β = angle between V? and V3
γ = angle between V1 and V3
δ = angle between V1 and the dotted horizontal line
V2
V1
V3
α
β
γ
δ
Y-Axis
X-Axis

Figure 3: Projected edge vectors of the diffraction pattern

The diffraction pattern captured consists of a significant amount of noise indicated by the distribution of white pixels along the dark regions within the circle, as shown in Figure 2. Therefore, the digitised image has to be first filtered to remove noise. Three regions of interest (ROI) are then created such that the vectors are embedded in these regions. It must be noted that the vectors V_i in the image could not be distinguished from one another. Therefore, each vector must be recognised once and followed. The recognition can be achieved by specifying a home position with a known home orientation (to be set to zero). The first set of ROI will always be applied to this diffraction pattern at home position. Application software has been developed to detect the vectors representing the diffraction pattern. This software utilises the library routines provided by the frame grabber and its image processing capabilities. The angle between the head of the vector and the x-axis can thus be determined.

When the new image is being captured, the vectors V_i will be rotated by a certain amount due to the roll angle rotation of the retroreflector. It is assumed that the rotation angle is small enough so that the vector lines would still be covered by the previous ROI. The new vectors can then be detected as well as the new set of angles. The centre of the retroreflector (i.e. the intersection points of the three vector lines) might have moved away from the previous position due to changes of the pitch and yaw angles. Therefore, it has to be calculated again using the new vectors. With the new centre found, the ROI is rotated by the difference between the old angle sets and the new angle sets, about the new centre point. The process is repeated until the angle change is negligibly small.

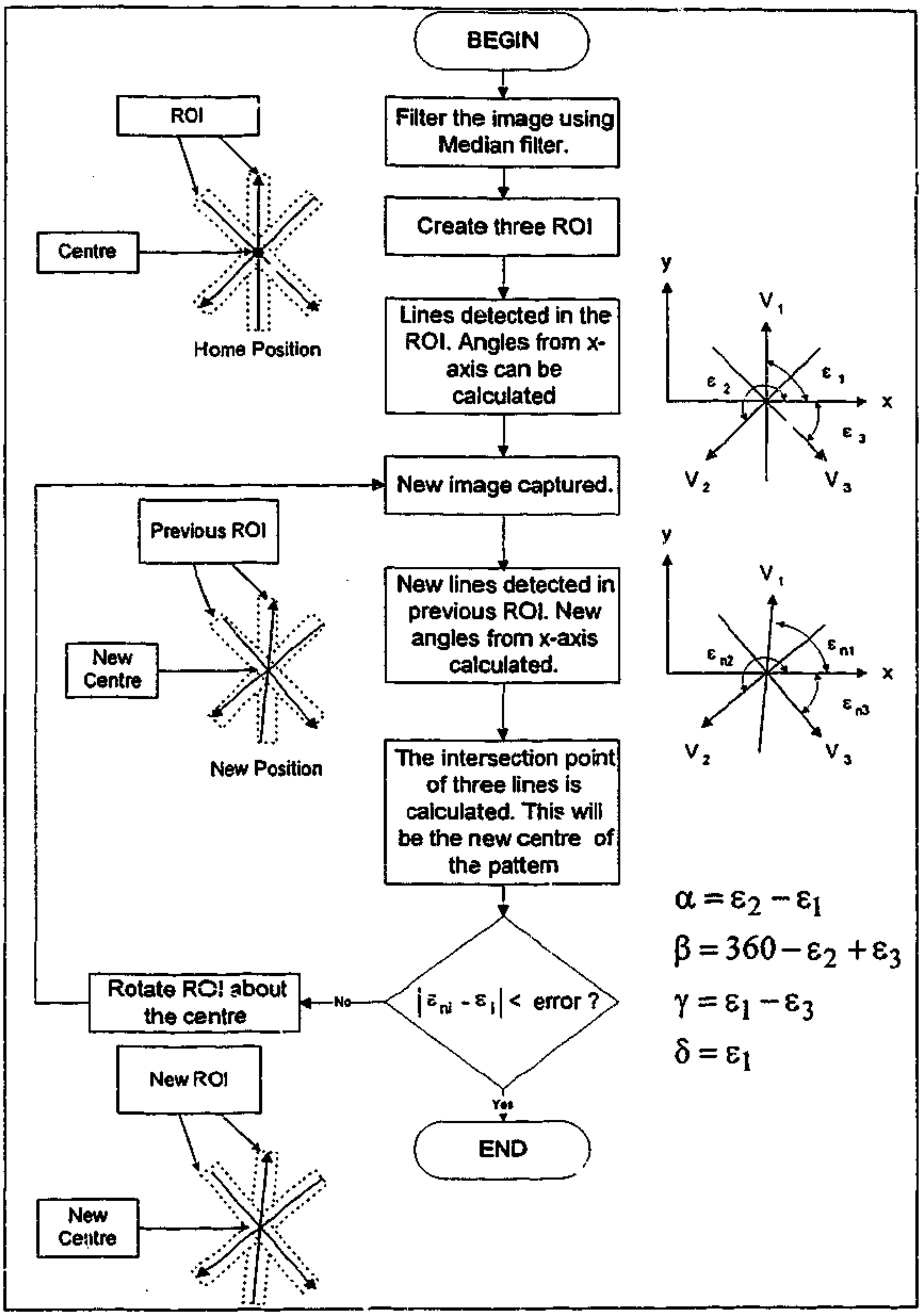

BEGIN
Filter the image using Median filter.
Create three ROI
Lines detected in the ROI. Angles from x-axis can be calculated
New image captured.
New lines detected in previous ROI. New angles from x-axis calculated.
The intersection point of three lines is calculated. This will be the new centre of the pattern
$|\varepsilon_{ni} - \varepsilon_i| <$ error ?
No
Yes
Rotate ROI about the centre
END
ROI
Centre
Home Position
Previous ROI
New Centre
New Position
New ROI
New Centre
y
x
V_1
V_2
V_3
ε_1
ε_2
ε_3
y
x
V_1
V_2
V_3
ε_{n1}
ε_{n2}
ε_{n3}
$\alpha = \varepsilon_2 - \varepsilon_1$
$\beta = 360 - \varepsilon_2 + \varepsilon_3$
$\gamma = \varepsilon_1 - \varepsilon_3$
$\delta = \varepsilon_1$

Figure 4: Flowchart for vision-based orientation measurement

3.2 Orientation Calculation

When the angles α, β, γ, and δ have been established, the orientation (roll ϕ, pitch Θ and yaw φ) of the retroreflector relative to the laser beam can be determined by using the following equations [3]:

$$\theta = -\arcsin\left(\sqrt{\frac{2}{3\tan\alpha\tan\gamma}} - \sqrt{\frac{-1}{6\tan(\alpha+\gamma)\tan\alpha}} - \sqrt{\frac{-1}{6\tan\gamma\tan(\alpha+\gamma)}} \right) \tag{1}$$

$$\phi = \arcsin\Big(\frac{1}{\cos\theta}\Big(\sin\delta\sqrt{\frac{-2\cos(\gamma+\alpha)}{3\sin\alpha\sin\gamma}}$$

$$-\sin(\delta+\alpha)\sqrt{\frac{\cos\gamma}{6\sin\alpha\sin(\alpha+\gamma)}} \qquad (2)$$

$$-\sin(\delta-\gamma)\sqrt{\frac{\cos\alpha}{6\sin\gamma\sin(\alpha+\gamma)}}\Big)\Big)$$

$$\varphi = \arcsin\Big(\frac{1}{\cos\theta}\sqrt{\frac{-1}{2\tan(\alpha+\gamma)\tan\alpha}} \qquad (3)$$

$$-\frac{1}{\cos\theta}\sqrt{\frac{-1}{2\tan\gamma\tan(\alpha+\gamma)}}\Big)$$

It must be emphasised that these angles are relative orientation from home (zero) orientation.

4. NON-VISION BASED ORIENTATION MEASUREMENT

Another approach to orientation measurement in LITS is the non-vision based orientation measurement [2]. The approach utilises two position sensitive diodes (PSDs), a beamsplitter, and a retroreflector mounted on a specially designed Gimbal unit as shown in Figure 5.

When the laser beam from the beam steering mechanism passes through the 70-30 beamsplitter, it splits into two beams. 30% of the beam is incident on the first PSD and the remaining will be directed towards the retroreflector. The reflected beam will travel back to the beamsplitter, parallel to the incident beam. At the beamsplitter, 30% of the laser beam will again be directed to the second PSD and the remaining laser beam will return to the beam steering mechanism.

A flowchart for the non-vision based orientation measurement algorithm is provided in Figure 6. From the flowchart, it can be observed that the orientation of the retroreflector is determined from the readings along the x and z axes on the PSDs.

Figure 7 shows a top view of the optical arrangement on the Gimbal unit. In the current experimental set-up, the gimbal unit is designed as shown in Figure 5. However, due to different approach in calculation, modification had been performed on the equations that were presented in [2].

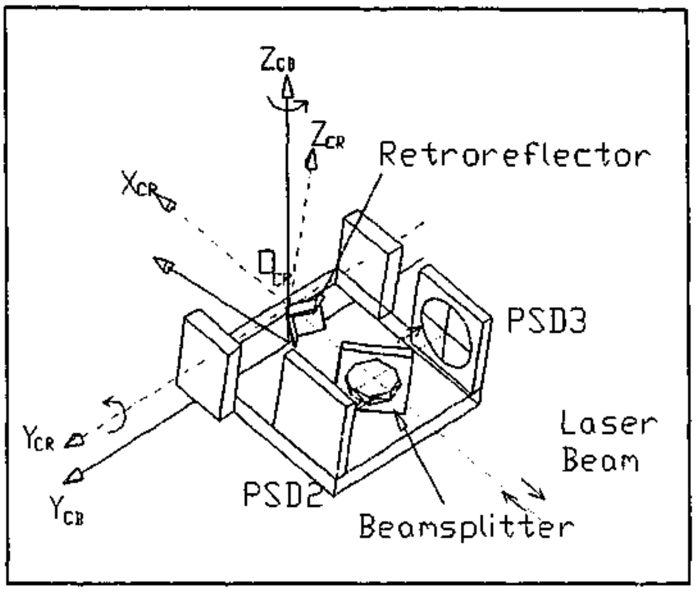
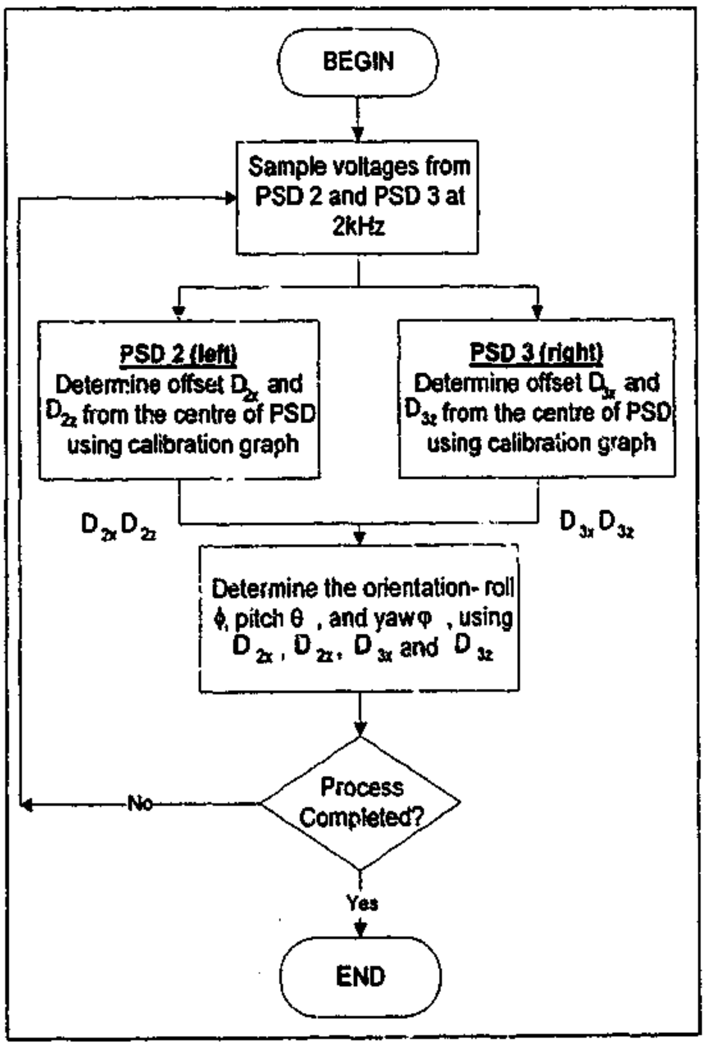
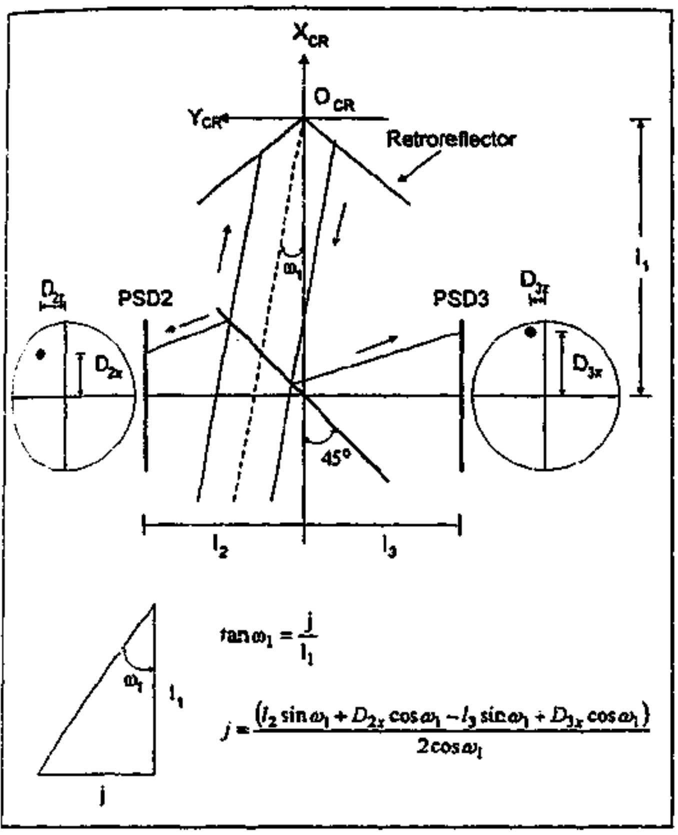


Figure 5: Gimbal unit assembly



BEGIN
Sample voltages from PSD 2 and PSD 3 at 2kHz
PSD 2 (left)
Determine offset D_{2x} and D_{2z} from the centre of PSD using calibration graph
PSD 3 (right)
Determine offset D_{3x} and D_{3z} from the centre of PSD using calibration graph
$D_{2x} D_{2z}$
$D_{3x} D_{3z}$
Determine the orientation- roll ϕ, pitch θ , and yaw φ , using D_{2x} , D_{2z}, D_{3x} and D_{3z}
Process Completed?
No
Yes
END

Figure 6: Flowchart for non-vision based orientation measurement



X_{CR}
O_{CR}
Y_{CR}
Retroreflector
ω_1
l_1
D_{2z}
PSD2
PSD3
D_{3z}
D_{2x}
D_{3x}
45°
l_2
l_3
$\tan\omega_1 = \frac{j}{l_1}$
ω_1
l_1
$j = \frac{(l_2 \sin\omega_1 + D_{2x}\cos\omega_1 - l_3 \sin\omega_1 + D_{3x}\cos\omega_1)}{2\cos\omega_1}$
j

Figure 7: Top view of Gimbal unit

The governing equations to determine the orientation of the retroreflector (co-ordinate system C_R) relative to the laser beam (co-ordinate system C_L) are thus as follow:

$$^{C_L}T_{C_R} = Rot(z,\omega_1)Rot(y,-\omega_2)Rot(x,\omega_3) \qquad (5)$$

From Figure 7, it can be seen from the properties of triangle that:

$$\omega_1 = \arctan\frac{D_{2x} + D_{3x}}{2l_1 - l_2 + l_3} \qquad (6)$$

With similar method in evaluating ω_1,

$$\omega_2 = \arctan\frac{(D_{2z} + D_{3z})\cos\omega_1}{2l_1 - l_2 + l_3} \qquad (7)$$

where l_1 is the distance between the centres of the beamsplitter and the retroreflector. l_2 and l_3 are distances between the centres of the beamsplitter and PSD2 and PSD3 respectively. D_{2x} and D_{3x} are the detection positions of laser beam on PSD2 and PSD3 along the x-axis, respectively. D_{2z} and D_{3z} are the detection positions of laser beam on PSD2 and PSD3 along the z-axis, respectively.

ω_3 can be calculated from the point of intersection between the laser beam and the yz plane of co-ordinate system C_B which is a stationary base of the gimbal unit in Figure 5.

5. COMPARISON BETWEEN VISION BASED AND NON-VISION BASED MEASUREMENT TECHNIQUES

5.1 Accuracy of orientation measurement

Generally, the accuracy of determining the orientation angles of the retroreflector, regardless of the orientation measurement techniques used, depends on a number of factors [6]:

- Geometric errors of the laser interferometry-based tracking system;
- Sensitivity of the laser beam due to the changes in environmental conditions. These include temperature, pressure, humidity, etc;
- Accuracy of other sub-components of the laser interferometry-based tracking system

In this paper, we are only interested in the comparison between the two techniques proposed. Therefore, only the accuracy of the equipment used in the measurement is being considered, without having to take into account the above errors.

Vision-based orientation measurement technique uses a CCD camera. The accuracy of this technique relies heavily on the capability of the software program to determine the vector lines projected onto the CCD camera. Experiments have shown that the accuracy for the current set-up is of the order of ±0.2 degrees. The current set-up consists of a DT3152 frame grabber board, a Pulnix TM6CN CCD camera and EasyLib image processing software.

Non-vision based orientation measurement technique uses a PSD sub-system on the Gimbal unit to determine the orientation angles, as described previously. The PSD sub-system consists of two PSDs with special circuit and a data acquisition card. These PSDs are of lateral effect detector type – i.e. two electrodes opposite to each other are used to determine the position of laser beam along a particular axis on the detector. The offset of the laser beam from the centre of the detector on a particular axis will be represented by a difference in voltage values between the two electrodes for that axis. Calibration has to be carried out to obtain a relationship between the voltage and displacement. This has to be carefully performed as the performance of non-vision based orientation measurement is heavily dependent on the sampling rate and accuracy of the PSDs. The software program for orientation measurement will refer to the calibration graph when the voltage values are detected to convert these into corresponding displacements.

The accuracy of the PSD used is 7 μm, which is half of its resolution plus the accuracy of the calibrator. Furthermore, the design tolerance of the Gimbal unit is in the order of 5 μm. Therefore, the accuracy was found to be in the order of ±0.012 degrees.

5.2 Limitations on Orientation Measurement

One major limitation to both vision and non-vision based orientation measurement is the rotational range of tracking. This tracking range is generally small (about ±30 degrees for pitch and yaw angles but no limitation for roll angle) due to the limited rotation angle of the retroreflector with respect to the laser beam. That is, the retroreflector will gradually turn away from the incoming laser beam until the system loses track of the signal. As a result, measurement cannot be performed.

However, for non-vision based measurement, this limitation can be overcome by rotating motors mounted on the Gimbal unit. The rotation angle will be equal to the calculated orientation angles. In this approach, the motors will turn the Gimbal unit (and thus the retroreflector) in such a way that the retroreflector will always face the direction of the incoming laser beam, contributing to zero roll, pitch and yaw angles relative to the laser beam. This method gives more flexibility to the orientation measurement.

For vision-based measurement, the sampling rate of the CCD camera is only 30 Hz. If an angle change of 1 degree is being measured in each cycle, the system is capable of providing a measurement for retroreflector rotating at up to 30 degrees/s. However, in the case of non-vision based measurement, the sampling rate of the PSDs is 2000 Hz. With the same angle change being measured at each cycle, the system is capable of measuring a rotational speed of 2000 degrees/s. This is about 66 times faster.

6. CONCLUSION AND FUTURE WORK

Laser tracking system has the ability to accurately provide measurements for position and orientation of robot's end-effector. Two methods of orientation measurements and their algorithms have been described. The techniques were compared and the limitations of each method were outlined. Due to low laser beam power, full experimental data could not be collected for this paper. However, from the preliminary calculations and feasibility experiments, it can be observed that the non-vision based measurement technique is more precise and faster than the vision based measurement technique.

Future development will involve the full incorporation of orientation measurement into LITS. The laser interferometer is to be replaced by another higher power laser interferometer. Predictive control algorithm will also be incorporated to improve the speed of measurement for both techniques.

ACKNOWLEDGEMENT

This research project was partly funded by grants from the Australian Research Council (ARC) and the Harold Armstrong Research Fund. The authors wish to thank H. C. CHONG and K. C. LEE for their work on the research project.

REFERENCES

[1] S. Spiess, M. Vincze, P. Krautgartner, K. Filz, "On modelling the kinematics and optics of a laser tracking system for contactless robot measurement", *Institute of Flexible Automation.* Technical University of Vienna, Austria, 1996.

[2] Y. Bao, N. Fujiwara, "Dynamic Measurement Orientation by LTS", *Japan/USA Symposium on Flexible Automation*, Vol. 1, pp 545-548, 1996.

[3] M. Vincze, J. P. Prenninger, H. Gander, "A Laser Tracking System to Measure Position and Orientation of Robot End Effectors Under Motion", *The International Journal of Robotics Research,* Vol. 13, No. 4, pp 305-314, 1994.

[4] B. Shirinzadeh, H. C. Chong, K. C. Lee, P. L. Teoh, "Issues and Techniques for Interferometry-Based Laser Guidance of a Manipulator" *The Fifth International Conference on Control, Automation, Robotics and Vision, Singapore,* Vol. 1, pp. 271-275, 1998.

[5] B. Shirinzadeh, "Laser interferometry-based tracking for dynamic measurements", *Industrial Robot,* Vol. 25. No. 1, pp. 35-41, 1998.

[6] K. M. Filz, M. Vincze, J. P. Prenninger, "Camera system to detect the orientation of a corner cube in real time", *IEEE International Conference on Robotics and Automation,* pp 1713-1719, 1995.

B. Shirinzadeh, P. L. Teoh, C. W. Foong, Y. D. Liu, "Orientation measurement technique in laser interferometry based tracking system for robot manipulator calibration", Proceedings of Pacific Conference on Manufacturing (PCM2000), Detroit, USA, Vol. 2, pp. 788-793, September 2000.

Orientation Measurement Technique in Laser Interferometry Based Tracking System for Robot Manipulator Calibration

*P. L. Teoh, *B. Shirinzadeh, *C. W. Foong, +Y. D. LIU

* Department of Mechanical Engineering
Monash University
Clayton, VIC 3168
AUSTRALIA
TEL: 61 3 9905 1565
FAX: 61 3 9905 1825
Email: bijan.shirinzadeh@eng.monash.edu.au

+ State Key Laboratory of Precision Measurement Technology and Instruments
TsingHua University
Beijing 100084
China

Abstract:

Laser Interferometry-based Tracking System (LITS) has been proposed as an advanced calibration technique that measures the robot position and orientation (pose). This technique can provide dynamic robot pose measurement in real time and has high accuracy, a large working space, a high sampling rate and automatic target tracking. The performance of LITS is dependent on the sub-systems. In this paper, the sub-system being investigated is the orientation measurement sub-system. Two measurement techniques; namely the vision and non-vision based techniques will be described. The principles and algorithms in obtaining the orientation of the robot's end-effector for both techniques will be presented. In particular, the emphasis is placed on the establishment of the non-vision based technique and investigating the feasibility of this technique for the measurement of orientation about a single axis. Comparison will be made between the vision and non-vision based measurement techniques in terms of efficiencies and limitations for orientation measurement.

Keywords: Laser interferometry based tracking system (LITS), Orientation measurement, Vision and non-vision based measurement techniques

1. INTRODUCTION

In the last two decades, there has been an increase in robot application in manufacturing and service industries. The robots are required to perform more complex assembly operation and high precision path following operations. These operations generally require higher robot accuracy and advanced flexible programming techniques. Calibration techniques that involves position and orientation (pose) measurement are required to improve the performance of robots. As the traditional methods of pose measurement are very time-consuming and expensive, a measurement tool with high accuracy, a large working space, a high sampling rate and automatic target tracking is desirable [1]. Laser interferometry-based tracking system (LITS) can be used in dynamic measurements of position and orientation thus providing measurements for robot calibration.

LITS comprises of various measuring sub-systems. The performance of a LITS depends on the performance of these sub-systems. These include the sampling rate of the sub-system responsible for offset error measurement, the sampling rate of the motion controller card, the accuracy or resolution of different sub-systems and the laser beam used. An ineffective sub-system will limit the performance of the LITS.

In this paper, the sub-system being studied is the orientation measurement sub-system. Two methods of orientation measurement will be described. These include vision (VBT) and non-vision based techniques (NVBT) to measure the orientation in real time. The principle and the algorithm for both techniques will be presented. Experiments on NVBT have been carried out to determine the feasibility of this technique for the measurement of orientation about a single axis, which includes accuracy and range of measurement. Both techniques will be compared in terms of the performance and efficiency by examining the accuracy of these techniques and their limitations.

2. PRINCIPLE OF LASER INTERFEROMETRY- BASED TRACKING SYSTEM FOR POSITION MEASUREMENT

The LITS developed for this study is shown in Figure 1 [2]. As shown, the laser beam emitted from the HeNe laser passes through an interferometer and is split into a reference beam and a measurement beam. The reference beam will be picked up by a fibre optic cable and be transmitted to the measurement board on the laser controller. The measurement beam passes through a 70-30 beamsplitter before being directed by the beam steering mechanism onto the retroreflector mounted on the end-effector of the robot manipulator. This beam is referred to as the incident beam.

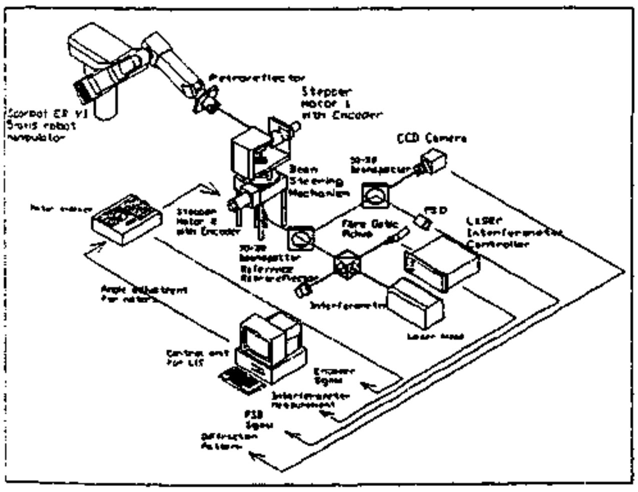


Figure 1: Setup of Laser Interferometry-based Tracking System

The retroreflector is designed in such a way that the reflected beam is always parallel to the incident beam, regardless of the incident angle of the beam. This reflected beam is again directed by the beam steering mechanism back to the 70-30 beamsplitter. 70% of the reflected beam will pass through the beamsplitter and travel to the interferometer before being picked up by the fibre optic cable. It is then transmitted and combined with the reference beam to determine the displacement of the laser beam (i.e. the distance of the centre of retroreflector from the interferometer) using Doppler Shift. The remaining 30% of the reflected beam is directed perpendicularly to the measurement beam and travels through a 50-50 beamsplitter. The beam is being split further with 50% of this beam is being incident onto a position sensitive diode (PSD), which determines the offset of the beam from the centre position of the retroreflector. The other 50% will be transmitted to a CCD (Charged-Coupled Device) camera for orientation measurement, which will be discussed in the next section.

The beam steering mechanism consists of three 45° beam-steers, with one being stationary, and the other two driven by two stepper motors - one for vertical axis and the other for horizontal axis. Continuos tracking of the retroreflector are possible by controlling the motors to rotate by the corrective angles determined from the kinematics of the LITS using displacement of the laser beam and the tracking error obtained from the PSD. This provides continuos tracking and update of the end-effector's pose in real time.

3. VISION BASED ORIENTATION MEASUREMENT

The vision based orientation measurement technique (VBT) was initially proposed by Vincze, *et. al.* [3, 4]. As mentioned in the previous section, part of the reflected measurement beam from the retroreflector is directed to the CCD camera. The current set-up consists of a Pulnix TM6CN CCD camera, DT3152 frame grabber board, and EasyLib image processing software [5]. The camera will capture the image of the diffraction pattern of the retroreflector and a high-acquisition-rate frame grabber will acquire this signal to perform image digitisation using image processing software. Figure 2 shows the image of the diffraction pattern as displayed on a computer screen. It consists of a white circle on a dark background, with 3 dark lines intersecting each other. The intersection point is the centre point of the retroreflector and the dark lines are caused by the edges of the three mirrors on the retroreflector. The image can be simplified as shown in Figure 3. The vectors V_i (i = 1, 2, 3) are the projected edge vectors of the diffraction pattern. Application software has been developed to enhance the captu:ed image and to detect the vectors representing the diffraction pattern. This software utilises the library routines provided by the frame grabber and its image processing capabilities. The angle between the head of the vector and the x-axis can thus be determined.

When the angles α, β, γ, and δ have been established, the orientation (roll ϕ, pitch θ and yaw φ) of the retroreflector relative to the laser beam can be determined by using the following equations [3]:

$$\theta = -\arcsin\left(\sqrt{\frac{2}{3\tan\alpha\tan\gamma}} - \sqrt{\frac{-1}{6\tan(\alpha+\gamma)\tan\alpha}} - \sqrt{\frac{-1}{6\tan\gamma\tan(\alpha+\gamma)}} \right) \tag{1}$$

$$\phi = \arcsin\left(\frac{1}{\cos\theta}\left(\sin\delta\sqrt{\frac{-2\cos(\gamma+\alpha)}{3\sin\alpha\sin\gamma}} - \sin(\delta+\alpha)\sqrt{\frac{\cos\gamma}{6\sin\alpha\sin(\alpha+\gamma)}} - \sin(\delta-\gamma)\sqrt{\frac{\cos\alpha}{6\sin\gamma\sin(\alpha+\gamma)}} \right)\right) \tag{2}$$

$$\varphi = \arcsin\left(\frac{1}{\cos\theta}\sqrt{\frac{-1}{2\tan(\alpha+\gamma)\tan\alpha}} - \frac{1}{\cos\theta}\sqrt{\frac{-1}{2\tan\gamma\tan(\alpha+\gamma)}} \right) \tag{3}$$

It must be emphasised that these angles are relative orientation from home (zero) orientation. A new set of vectors as well as the new set of angles can be detected when there is a change in the orientation of the retroreflector. The process is repeated until the angle change is negligibly small.

Preliminary experiments have shown that the accuracy for the current set-up is of the order of ±3 degrees. Further, real time, automated measurement is not possible at the current stage. This is due to low laser power and unstable laser beam. Diffraction patterns with clear distinction between the dark and light regions cannot be obtained automatically.

Figure 2: Diffraction pattern of the :eflected beam from the retroreflector

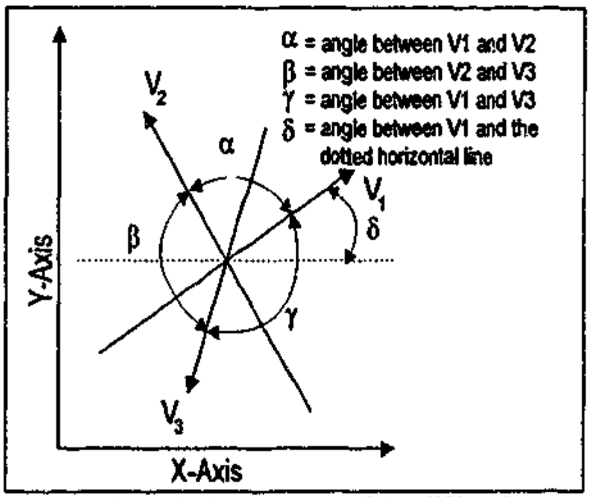

α = angle between V1 and V2
β = angle between V2 and V3
γ = angle between V1 and V3
δ = angle between V1 and the dotted horizontal line
V_2
α
V_1
β
δ
γ
V_3
Y-Axis
X-Axis

Figure 3: Projected edge vectors of the diffraction pattern

4. NON-VISION BASED ORIENTATION MEASUREMENT

The next approach to orientation measurement in LITS is the dual PSD non-vision based orientation measurement technique (NVBT) [6]. This approach utilises two position sensitive diodes (PSDs), a beamsplitter, and a retroreflector mounted on a specially designed Gimbal unit as shown in Figure 4. This Gimbal unit will replace the target retroreflector originally mounted on the robot end-effector as described in section 2.

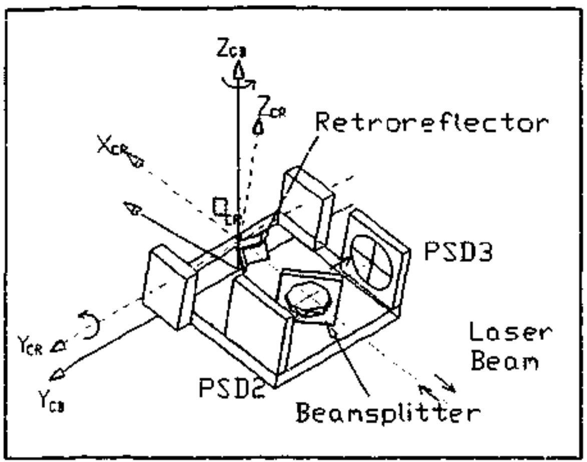


Figure 4: Gimbal unit assembly

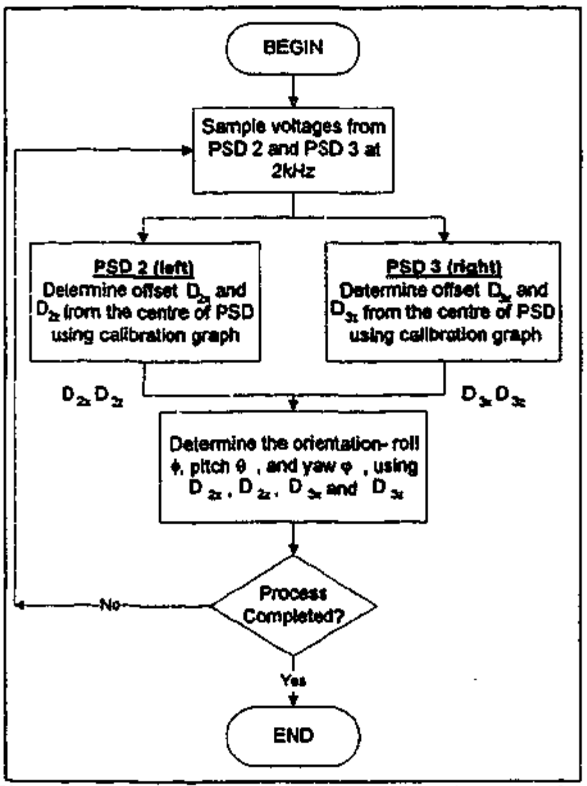

BEGIN
Sample voltages from PSD 2 and PSD 3 at 2kHz
PSD 2 (left)
Determine offset D_{2x} and D_{2z} from the centre of PSD using calibration graph
PSD 3 (right)
Determine offset D_{3x} and D_{3z} from the centre of PSD using calibration graph
D_{2x} D_{2z}
D_{3x} D_{3z}
Determine the orientation- roll ϕ, pitch θ , and yaw φ , using D_{2x} , D_{2z} , D_{3x} and D_{3z}
Process Completed?
No
Yes
END

Figure 5: Flowchart for non-vision based orientation measurement

A flowchart for the non-vision based orientation measurement algorithm is provided in Figure 5. As shown, the laser beam emerges from the beam steering mechanism passes through the 70-30 beamsplitter on the Gimbal and splits into two beams. 30% of the beam is incident on the first PSD and the remaining will be directed towards the retroreflector. The reflected beam will travel back to the beamsplitter, parallel to the incident beam. At the beamsplitter, 30% of the laser beam will again be directed to the second PSD and the remaining laser beam will return to the beam steering mechanism. Both PSDs can detect the beam offset from the centre along the x and z axes on the PSDs. These offset readings are used for the calculation of the orientation of the retroreflector.

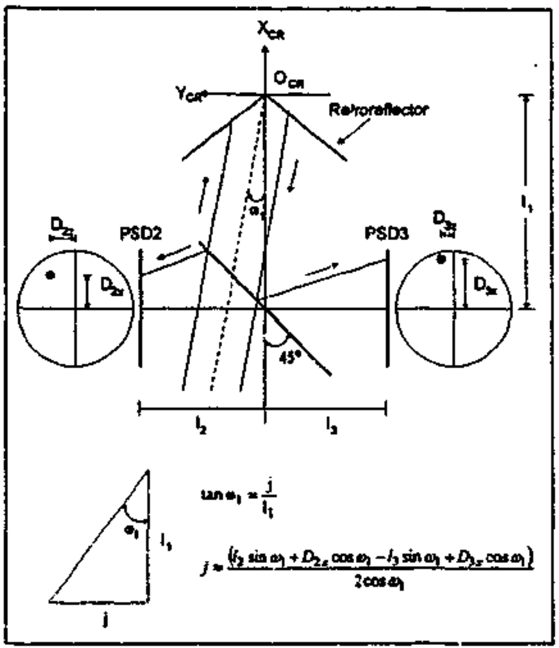

X_{CR}
O_{CR}
Y_{CR}
Retroreflector
l_1
ω_1
D_{2z}
PSD2
PSD3
D_{3z}
D_{2x}
D_{3x}
45°
l_2
l_3
$\tan\omega_1 = \frac{j}{l_1}$
ω_1
l_1
$j = \frac{(l_2 \sin\omega_1 + D_{2x}\cos\omega_1 - l_3 \sin\omega_1 + D_{3x}\cos\omega_1)}{2\cos\omega_1}$
j

Figure 6: Optical arrangement of Gimbal unit with beam path

The governing equations to determine the orientation of the retroreflector (co-ordinate system C_R) relative to the laser beam (co-ordinate system C_L) are thus as follow. Due to a different approach in calculation, modification has been performed on the equations that were presented in [6].:

$$^{C_L}T_{C_R} = Rot(z,\omega_1)Rot(y,-\omega_2)Rot(x,\omega_3) \qquad (5)$$

From Figure 6, it can be seen from the properties of triangle that:

$$\omega_1 = \arctan\frac{D_{2x} + D_{3x}}{2l_1 - l_2 + l_3} \qquad (6)$$

With similar method in evaluating ω_1,

$$\omega_2 = \arctan\frac{(D_{2z} + D_{3z})\cos\omega_1}{2l_1 - l_2 + l_3} \qquad (7)$$

where l_1 is the distance between the centres of the beamsplitter and the retroreflector. l_2 and l_3 are distances between the centres of the beamsplitter and PSD2 and PSD3 respectively. D_{2x} and D_{3x} are the detection positions of the laser beam on PSD2 and PSD3 along the x-axis, respectively. D_{2z} and D_{3z} are the detection positions of the laser beam on PSD2 and PSD3 along the z-axis, respectively.

ω_3 can be calculated from the point of intersection between the laser beam and the yz-plane of co-ordinate system C_B which is a stationary base of the gimbal unit in Figure 4.

4.1 Experimental setup

As shown in Figure 5, the PSDs used samples voltages at 2000Hz. These PSDs are of lateral effect detector type – i.e. two electrodes opposite to each other are used to determine the position of laser beam along a particular axis on the detector. The offset of the laser beam from the centre of the detector on a particular axis will be represented by a difference in voltage values between the two electrodes for that axis. Proper calibration has to be carried out to determine the relationship between the voltages sampled and the actual position offset of the beam on the PSDs surface. The PSDs were calibrated by directing the laser beam onto the centre of each PSD mounted on a linear positioning table. The voltage difference between the electrodes and the corresponding beam displacement was recorded while micro-stepping the PSD in one direction. Due to non-linearity of the PSD, the results were linearised with linearisation algorithm and a calibration graph of relationship between the voltage difference and the displacement can then be obtained. The software program for orientation measurement will refer to the calibration graph when the voltage values are detected to convert these into corresponding displacements.

Experiments were performed in order to observe the performance of the dual PSD offsets NVBT in determining the Gimbal rotation about a single axis (i.e. the z-axis). The equipment used for this experiment included two PSDs mounted on the Gimbal unit, a micro-stepping rotational table with a maximum resolution of 2286000 steps per revolution, and a HeNe laser interferometer.

4.2 Results and Discussion

The Gimbal unit was first mounted onto a rotational table stage. The resolution of the rotational table was set at 18000 steps per revolution. The stage was then stepped through from 0 to 200 steps with 5 steps increment. The PSD offsets were recorded. The results are shown in Figure 7.

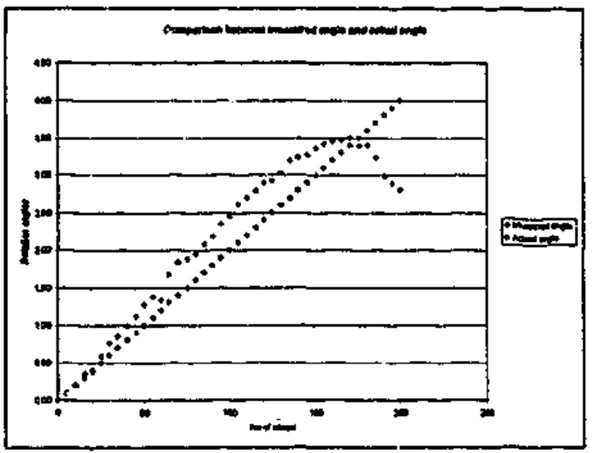


Figure 7: Comparison between measured and actual angle

The graph in Figure 7 shows that at small rotations (from 0 to 25 steps), the measured angular rotation about the z axis using the Gimbal unit agrees closely with the actual angular rotation with a deviation of 2.5%. The deviation increases to 24% with larger rotation angle. The main reason for the deviation is due to the low laser power. As voltage is directly proportional to power, reduction in power will reduce the maximum voltage sampled. A change in beam displacement will cause a small voltage change. This increases the sensitivity of the PSDs to ambient environment. In this setup, only 30% and 21% of the beam power directed to the Gimbal unit will be projected to PSD1 and PSD2 respectively. The other reason is the simple linearisation algorithm used. More complex algorithm is required.

It can be observed in Figure 7 that the measured angle reached a maximum value of 170 steps and then started to decrease. This is due to the beam is out of the PSDs range. The measurement range is thus ±3.4°.

5. COMPARISON BETWEEN VISION BASED AND NON-VISION BASED MEASUREMENT TECHNIQUES

5.1 Accuracy of orientation measurement

The accuracy of VBT orientation measurement relies heavily on the capability of the software program to determine the vector lines projected onto the CCD camera. Experiments have shown that the accuracy for the current set-up is of the order of ±0.2 degrees.

NVBT orientation measurement uses PSDs to determine the orientation angles, as described previously. The accuracy of NVBT is heavily dependent on the accuracy of the calibration graph of the PSDs. Experiments have shown that the current set-up has a deviation of ±2.5% for small rotation angles and ±24% for larger rotation angles.

5.1 Limitations on Orientation Measurement

The major limitation experienced by both VBT and NVBT orientation measurement is the rotational range of measurement. There is no range limitation for roll angle. The measurement range of about ±30 degrees for pitch and yaw angles is due to the rotation of the retroreflector, gradually turning away from the incoming laser beam until the system loses track of the signal. As a result, measurement cannot be performed.

The measurement range of current NVBT set-up as mentioned above is about ±3.5 degrees. However, this limitation can be overcome during actual operation with the Gimbal mounted on the robot end-effector by rotating motors connected to the Gimbal unit. The rotation angle will be equal to the calculated orientation angles. In this approach, the motors will turn the Gimbal unit (and thus the retroreflector) in such a way that the retroreflector will always face the direction of the incoming laser beam. This method gives more flexibility to the orientation measurement.

6. CONCLUSION AND FUTURE WORK

Two methods of orientation measurements and their algorithms have been described. Experiment verified that NVBT is feasible in making orientation measurement. Better results could be obtained with better equipment such as higher laser power. The techniques were compared and it can be observed that the NVBT has more flexibility compare to the VBT.

Further experiments will be performed to improve the technique.

7. ACKNOWLEDGEMENT

This research project was partly funded by grants from the Australian Research Council (ARC) and the Harold Armstrong Research Fund.

8. REFERENCES

[1] S. Spiess, M. Vincze, P. Krautgartner, K. Filz, "On modelling the kinematics and optics of a laser tracking system for contactless robot measurement", *Institute of Flexible Automation.* Technical University of Vienna, Austria, 1996.

[2] B. Shirinzadeh, "Laser interferometry-based tracking for dynamic measurements", *Industrial Robot*, Vol. 25. No. 1, pp. 35-41, 1998.

[3] M. Vincze, J. P. Prenninger, H. Gander, "A Laser Tracking System to Measure Position and Orientation of Robot End Effectors Under Motion", *The Int. Journal of Robotics Research*, Vol. 13, No. 4, pp 305-314, 1994.

[4] K. M. Filz, M. Vincze, J. P. Prenninger, "Camera system to detect the orientation of a corner cube in real time", *IEEE Int. Conf. on Robotics and Automation*, pp 1713-1719, 1995.

[5] B.Shirinzadeh, P. L. Teoh, C. W. Foong, "Orientation measurement using vision and non-vision based techniques in laser tracking system", *30th Inte. Symp. on Robotics, Tokyo, Japan*, pp. 317-324, 1999.

[6] Y. Bao, N. Fujiwara, "Dynamic Measurement Orientation by LTS", *Japan/USA Symp. on Flexible Automation*, Vol. 1, pp 545-548, 1996.

P. L. Teoh, B. Shirinzadeh, "Dual position sensitive diode-based orientation measurement in laser interferometry-based sensing and measurement technique", Proceedings of SPIE-The International Society for Optical Engineering, Vol. 4564, pp. 98-106, October 2001.

Dual Position Sensitive Diode-based Orientation Measurement in Laser Interferometry-based Sensing and Measurement Technique

B. Shirinzadeh, P. L. Teoh
Department of Mechanical Engineering
Monash University
Clayton, VIC 3168
Australia
Email: bijan.shirinzadeh@eng.monash.edu.au
pek.teoh@eng.monash.edu.au

ABSTRACT

The accurate measurement of the position and orientation (pose) of a robot manipulator's end-effector is the most critical issue for calibration of the robotic devices. To accurately define a position and orientation of a robot manipulator's end-effector, measurement of six parameters is required - three for position and three for orientation. Different approaches have been studied over the last few years to measure the orientation of the end-effector dynamically and precisely. However, there are still some difficulties in determining the orientation of the end-effector in real time. In this paper, an orientation measurement methodology based on Position Sensitive Diode (PSD) measurement in Laser Interferometry-based Sensing and Measurement (LISM) technique will be described. The principle and algorithms of this approach will be presented. The experimental set-up will also be described. The efficiencies and limitations of such approach will be examined.

Keywords: Laser interferometry based sensing and measurement technique (LISM), orientation measurement, dual PSD based orientation measurement, gimbal unit

1. INTRODUCTION

The application areas for robots in manufacturing and service industries are increasing in recent years. These application areas are generally more complex, requiring higher robot accuracy and advanced flexible programming techniques. Due to these requirements, a more accurate placement of robot's end-effector along a desired path is required. However, current industrial robots have much lower accuracy compared to their repeatability [1]. Therefore, an advanced calibration technique that involves an accurate measurement of robot position and orientation (pose) will be required to greatly improve the absolute positioning accuracy of robot manipulators.

The pose of the robot manipulator's end-effector is defined by six parameters- three for position and three for orientation. Traditional methods of pose measurement (such as ball bar, CMM) are very time-consuming and many do not provide for real time dynamic measurement. Laser Interferometry-based Sensing and Measurement (LISM) technique has been proposed to perform dynamic performance measurements of position and orientation as well as providing measurements for robot calibration. This technique can provide high accuracy measurements within a large working space with a high sampling rate and automatic target tracking capability in real time [2]. It has been shown that a number of different sub-systems are responsible for position and orientation measurement in LISM technique. The performance of LISM technique is dependent on the measurement methodology employed by these sub-systems. In this paper, the sub-system being investigated is the orientation measurement sub-system.

Different methodologies have been proposed to perform orientation measurements of robot's end-effector dynamically and precisely in LISM. These include the use of CCD cameras and arrays [2, 3, 4]. These measurement methodologies suffer similar limitation of low speed and/or accuracy. This paper describes an alternative method that can be used in orientation measurement; namely the dual Position Sensitive Diode (PSD) based orientation measurement technique.

This method consists of a Gimbal unit mounted on the robot's end-effector. It utilises geometric parameters of Gimbal unit and laser beam offset error measurements from the PSDs on the Gimbal unit to perform orientation measurements in real time. The principles and algorithms in obtaining the orientation of the robot's end-effector using this method will be presented. The preliminary experimental set-up will also be described. The efficiency and limitation of this methodology in providing orientation measurement in LISM measurement technique will be examined.

2. PRINCIPLE OF LASER INTERFEROMETRY-BASED SENSING AND TRACKING TECHNIQUE

Laser Interferometry-based Sensing and Measurement (LISM) technique generally involves the dynamic acquisition of the three dimensional position of an end-effector in its workspace [2, 5]. It can also be modified to measure the orientation of robot end-effector [1, 3, 6, 7]. The LISM technique uses the angular and distance data, obtained from the beam steering mechanism and the interferometer, respectively, to provide the position of the target retroreflector attached to the end-effector of the manipulator. It maintains tracking of the target by sensing the offset of the incident and reflected beam. The tracking is carried out by adjusting the angles of a beam steering mechanism. A LISM apparatus was developed for this study. A functional layout of the overall design of the LISM apparatus is provided in Figure 1.

In this layout, the laser beam generated by the HeNe laser head travels to the interferometer where it is split into a reference beam and a measurement beam. The reference beam is directed to the measurement board in the laser interferometer controller via the fibre optic pickup. This beam will later be compared with the returning measurement beam, whose frequency will be Doppler-shifted, to determine the distance between the target (i.e. a retroreflector mounted on the robot end-effector) and the laser head. The measurement beam travels through a 70-30 percent beam splitter before entering the two-axis beam steering mechanism. The beam steering mechanism consists of three 45° beam-steer mirrors, with one being stationary and the other two driven by two stepper motors – one rotate about the vertical axis and the other about the horizontal axis. Two high-resolution optical encoders are attached to the stepper motors to provide the angular displacement of the stepper motors and thus the rotational angles of the beam-steer mirrors. The beam is directed to the target retroreflector by rotating the stepper motors attached to the appropriate beam-steer mirrors. Once the beam hits the target retroreflector, the beam is reflected through 180° and it travels back parallel with the incident beam. There will exist an offset between the incident and the reflected measurement beam if the beam does not hit the centre of the retroreflector.

The reflected beam then travels through the beam-steer mirrors in the beam steering mechanism and back to the 70-30 percent beam splitter. 30 percent of the beam power is diverted through a 50-50 percent beam splitter, where it is split equally and directed to a Charged Coupled Device (CCD) camera and a Position Sensitive Diode (PSD). The CCD camera, which is connected to a high-data-acquisition-rate frame grabber board, captures the diffraction pattern of the retroreflector, so that analysis can be performed to determine the orientation of the target retroreflector. The PSD is attached to a data acquisition card and it detects the offset of the beam from the centre of the PSD sensor. The reflected beam position from the centre of the PSD will be acquired by the card and the offset, referred to as the tracking error will be stored in the LISM control unit. The remaining 70 percent of the beam power will be combined with the reference beam via the interferometer and the fibre optic pickup. The Doppler shift of the reflected beam can be detected and used by the processing electronics within the laser interferometer controller to determine the displacement of the beam and the velocity.

The LISM control system minimises the tracking error obtained from the PSD acquisition system by signaling the motor controller which in turn rotates the axes of the beam steering mechanism by a corrective angles calculated using the tracking error, thus following the arbitrary movements of the target. Measurement of the position of the target in space is obtained from the interferometer measurement, tracking errors, angular displacements of the axes of the beam steering mechanism, and the kinematics of the LISM apparatus. The tracking algorithm utilises a predictive control algorithm that allows estimation of future position of the target from the previous position, velocity and acceleration values [8].

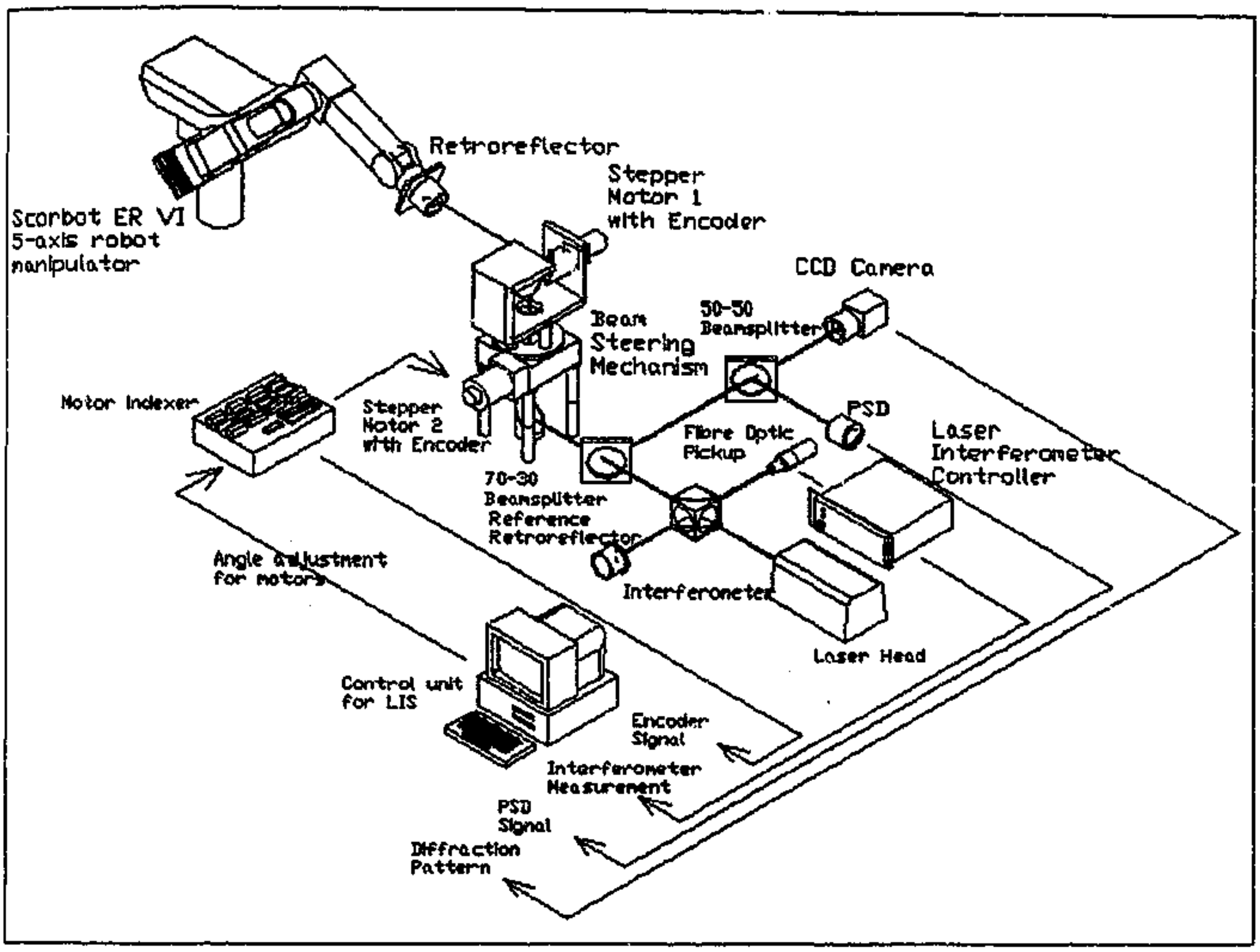


Figure 1: Functional layout of LISM apparatus

3. PRINCIPLE OF DUAL PSD-BASED ORIENTATION MEASUREMENT METHODOLOGY

There are a few methods of orientation measurement in laser interferometry-based measurement including the CCD camera and the CCD array based methods [2, 3, 4]. Another approach includes multi-laser interferometry-based technique that utilizes triangulation. All these methods can provide real time orientation measurement. However, for the CCD camera and CCD array based methods, the range of measurement is limited due to the small incident angle range of the retroreflector. Moreover, the center of the retroreflector has to be covered by the laser beam for a valid measurement to be made. For the case of the multi-laser interferometry-based technique, the set-up cost will be expensive compared to the single laser-interferometry based technique. Another approach to orientation measurement in LISM is the dual PSD-based orientation measurement [9]. This method is proposed to dynamically measure the orientation of a mobile vehicle with a specially designed Gimbal unit mounted on top of the vehicle.

3.1 Gimbal unit

The layout of the Gimbal unit is provided in Figure 2. The Gimbal unit consists of two position sensitive diodes (PSDs), a beamsplitter, and a retroreflector mounted on the intersection of the axes of rotations. When the laser beam from the beam steering mechanism passes through the 70-30 beamsplitter on the Gimbal unit, it splits into two beams. 30% of the beam is incident on the first PSD and the remaining will be directed towards the retroreflector. The reflected beam will travel back to the beamsplitter, parallel to the incident beam. At the beamsplitter, 30% of the laser beam will again be directed to the second PSD and the remaining laser beam will return to the beam steering mechanism. Orientation of the retroreflector with respect to the laser beam is calculated from the beam positions measurement along the x and y axes of the two PSDs and the geometry of the Gimbal unit. A flowchart for the non-vision based orientation measurement algorithm is provided in Figure 3.

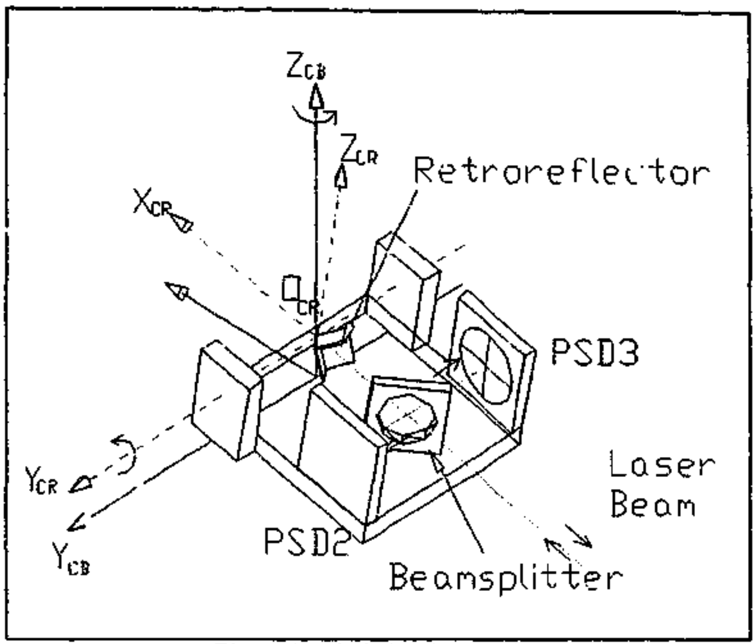


Figure 2: Gimbal unit assembly

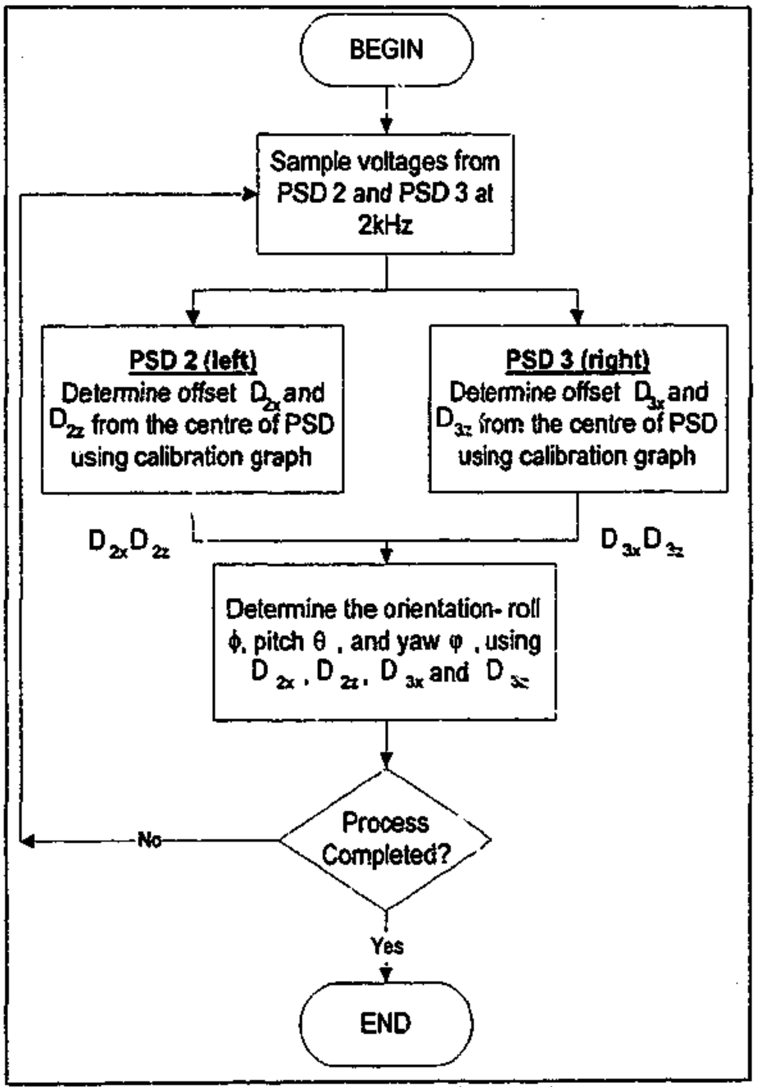

BEGIN
Sample voltages from PSD 2 and PSD 3 at 2kHz
PSD 2 (left)
Determine offset D2x and D2z from the centre of PSD using calibration graph
PSD 3 (right)
Determine offset D3x and D3z from the centre of PSD using calibration graph
D2x D2z
D3x D3z
Determine the orientation- roll ϕ, pitch θ , and yaw φ , using D2x , D2z , D3x and D3z
Process Completed?
No
Yes
END

Figure 3: Flowchart for non-vision based orientation measurement

3.2 Position Sensitive Diode

The PSDs used are of lateral effect detector type as shown in Figure 4. It has four electrodes connected equidistant around its perimeter. The electrodes yield photo-current corresponding to the displacement of the beam from the centre of the PSD [10]. Photo-currents are converted to voltages by special electronics before being sampled by the controller. Calibration has to be carried out to obtain a relationship between the voltage difference of the opposite electrode pair and the displacement of the laser beam in the X and Y direction from the centre of the PSDs. This has to be carefully performed as the performance of non-vision based orientation measurement is heavily dependent on the accuracy of the PSDs. Due to only 30% and 21% of the original laser beam power is directed to PSD2 and PSD3 respectively, calibration have to be performed with the correct laser beam power. Moreover, it was observed that there is significant amount of reflection from the surface of the PSDs. The reflections from PSD2 will cause corruption to the data sampled by PSD3 and vice versa. A quarter wave plate is placed in front of both PSDs to minimize reflection. This again has to be taken into account during the calibration process. The software program for orientation measurement will refer to the calibration graph when the voltage values are detected to convert these into corresponding displacements. Figure 5 shows a graphical relationship between the position of the laser beam and the voltage difference detected by the PSD. The range of the PSD is ±6mm from the center as beyond this range the laser beam will be out of the PSD detection area.

3.3 Formulation

Figure 6 shows a top view of the optical arrangement on the Gimbal unit. In the current set-up, the gimbal unit is designed as shown in Figure 2. However, due to different approach in calculation, modification had been performed on the equations that were presented in [9].

The governing equations to determine the orientation of the retroreflector (co-ordinate system C_R) relative to the laser beam (co-ordinate system C_L) are thus as follow:

$$ {}^{C_L}T_{C_R} = Rot(z,\omega_1)Rot(y,-\omega_2)Rot(x,\omega_3) \tag{1} $$

From Figure 6, it can be seen from the properties of triangle that:

$$ \omega_1 = \arctan\frac{D_{2x}+D_{3x}}{2l_1 - l_2 + l_3} \tag{2} $$

With similar method in evaluating ω_1,

$$ \omega_2 = \arctan\frac{(D_{2z}+D_{3z})\cos\omega_1}{2l_1 - l_2 + l_3} \tag{3} $$

where l_1 is the distance between the centres of the beamsplitter and the retroreflector. l_2 and l_3 are distances between the centres of the beamsplitter and PSD2 and PSD3 respectively. D_{2x} and D_{3x} are the detection positions of laser beam on PSD2 and PSD3 along the x-axis, respectively. D_{2z} and D_{3z} are the detection positions of laser beam on PSD2 and PSD3 along the z-axis, respectively.

ω_3 can be calculated from the point of intersection between the laser beam and the yz-plane of the co-ordinate system of the base of the Gimbal unit, C_B.

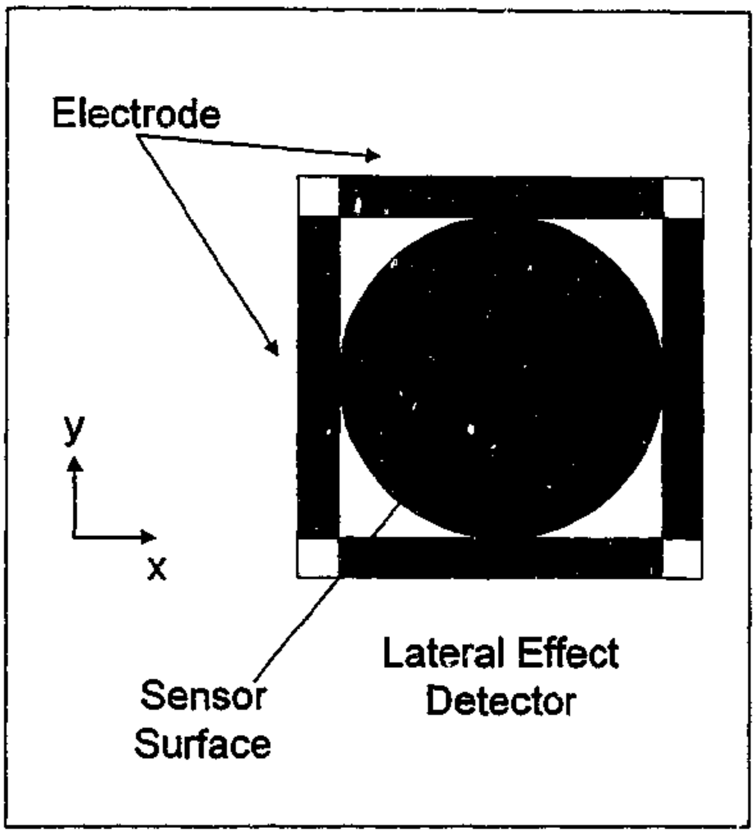


Figure 4: Dual axis lateral effect detector

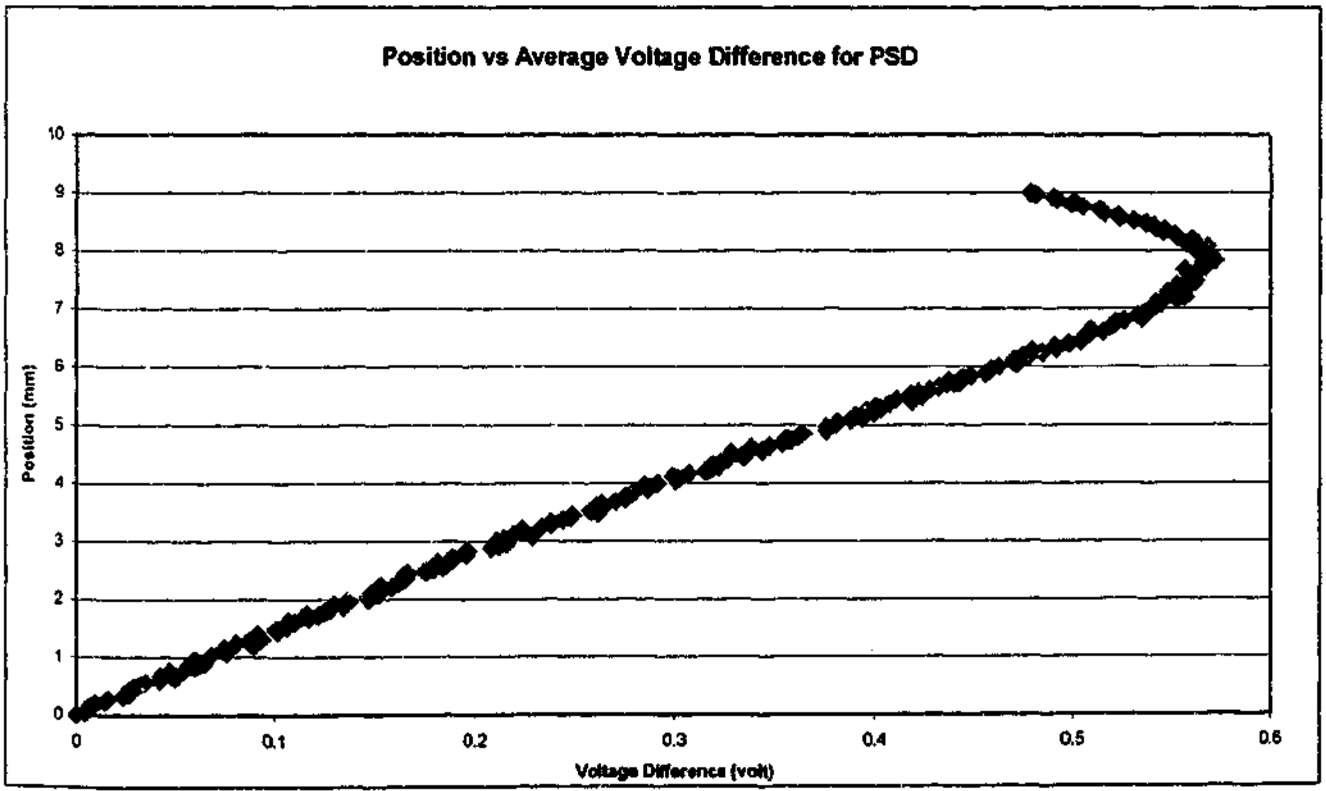
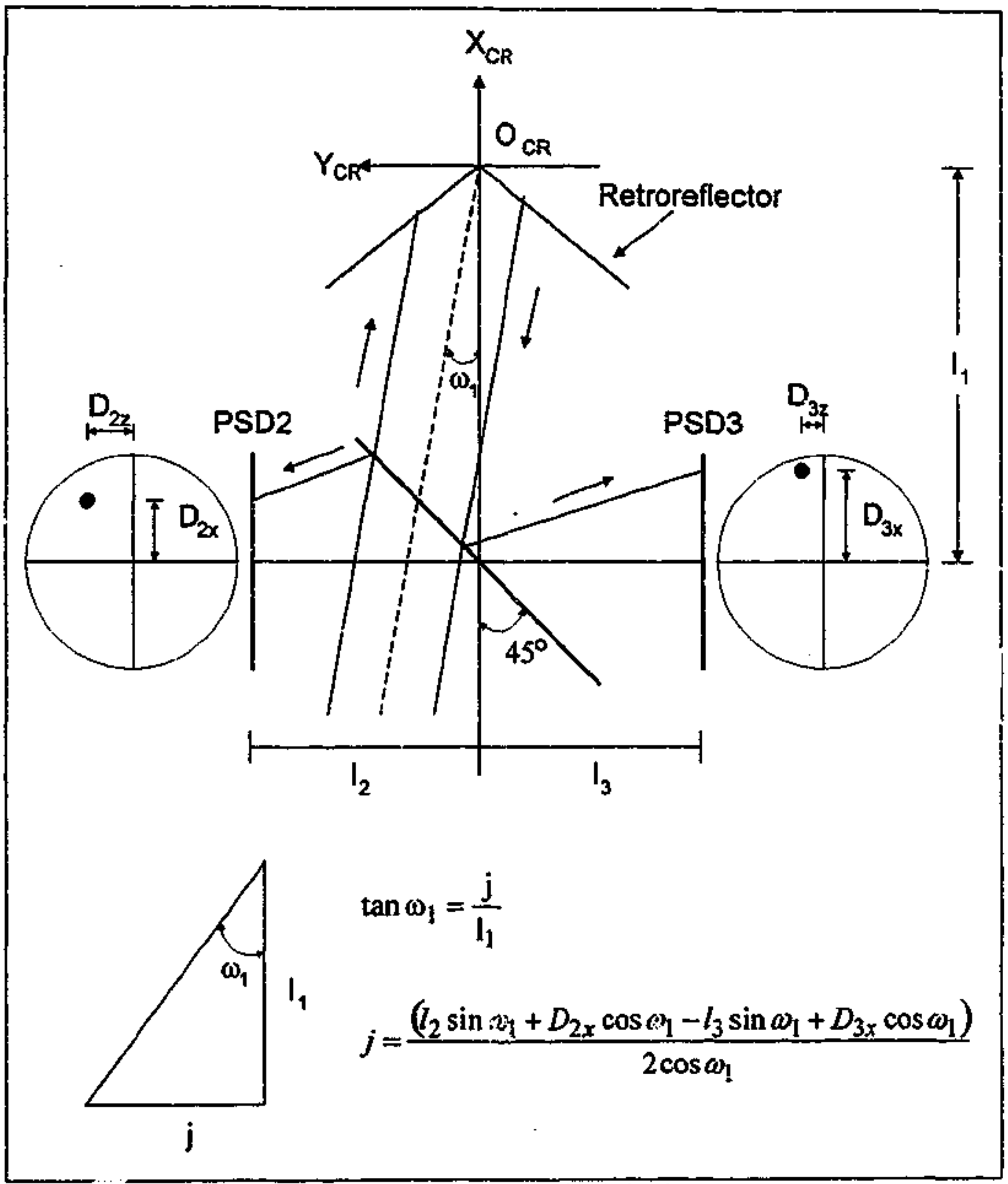


Figure 5: Calibration graph for PSD



$$\tan \omega_1 = \frac{j}{l_1}$$

$$j = \frac{\left(l_2 \sin \omega_1 + D_{2x} \cos \omega_1 - l_3 \sin \omega_1 + D_{3x} \cos \omega_1\right)}{2 \cos \omega_1}$$

Figure 6: Top view of Gimbal unit

4. PRELIMINARY EXPERIMENTAL SET-UP AND RESULTS

The experimental set-up consists of the Gimbal unit described above, a HeNe laser interferometer and a micro stepping rotary table with a resolution of 0.02° and a maximum resolution of 2286000 steps per revolution. The Gimbal unit is mounted on the rotary table with the x-axis of the Gimbal unit co-ordinate system coincide with the rotating axis of the rotary table. The experiments conducted dealt with the orientation measuring performance of the LISM. The experiments focused on determining the accuracy, range of orientation measurement about a single axis (i.e. the x-axis of the Gimbal unit co-ordinate system).

During the experiment, the rotary table is commanded to step through from 0 to 200 steps with 5 steps increment. The positions of the laser beam from the center of both PSDs are recorded at the same time. Equations 1 is then used to calculated the orientation of the retroreflector. The results are compared with the theoretical value calculated using the commanded step size and velocity. A graphical representation of the comparison is shown in Figure 7.

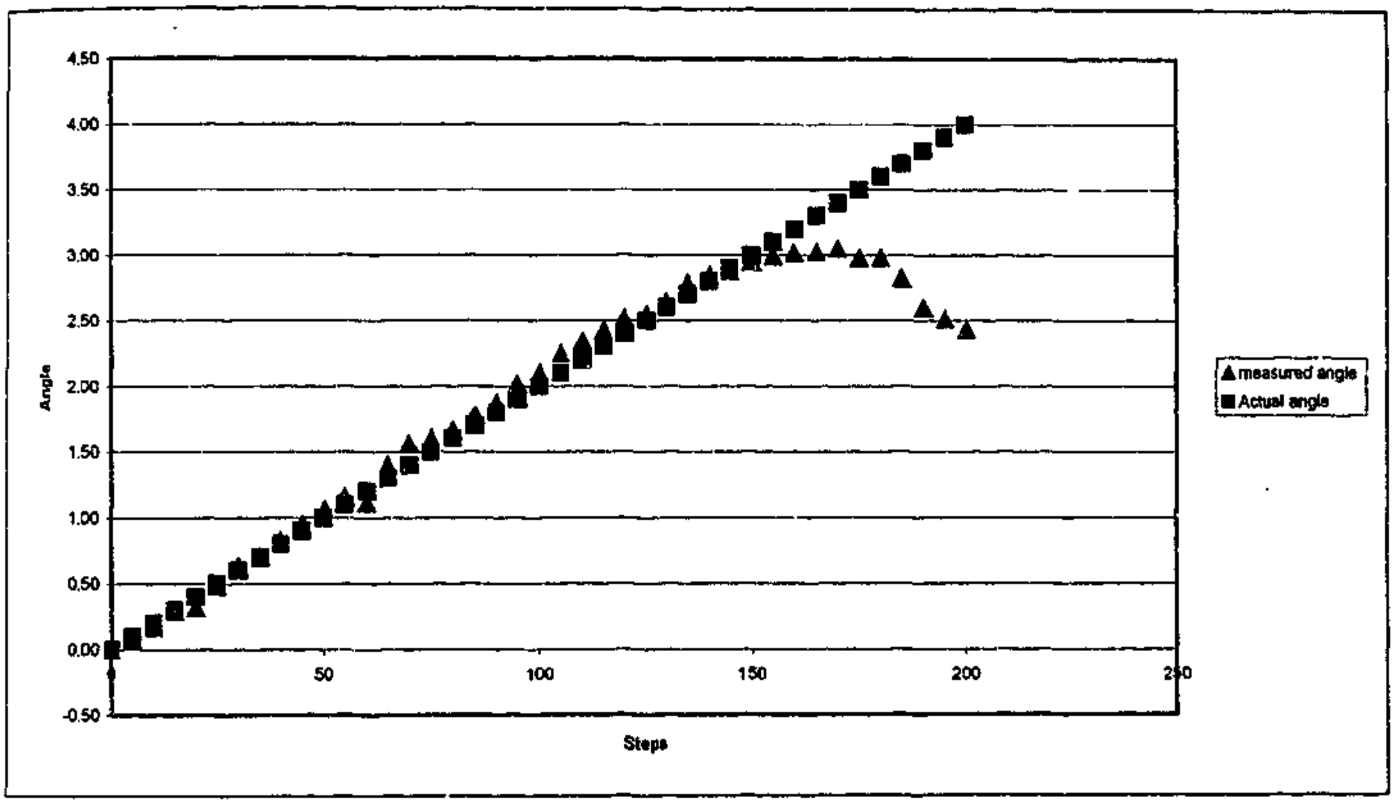


Figure 7: Comparison between actual and measured angle

5. DISCUSSION

The graph in Figure 7 shows the measured angular rotation about the z-axis using the Gimbal unit agrees closely with the actual angular rotation up to 50 steps. The deviation increases with larger rotation angle. The main reason for the deviation is due to the low laser power. As voltage is directly proportional to power, reduction in power will reduce the maximum voltage sampled. A change in beam displacement will cause a small voltage change. This increases the sensitivity of the PSDs to ambient environment. In this setup, only 30% and 21% of the beam power directed to the Gimbal unit will be projected to PSD1 and PSD2 respectively. The other reason is the simple linearisation algorithm used to determine the relationship between the position of the beam and the voltage difference. A better algorithm maybe used to obtain a more accurate relationship. Other sources of errors include noise and geometric uncertainties of the Gimbal unit.

One major limitation to this methodology is the rotational range of tracking. It can be observed in Figure 7 that the measured angle reached a maximum value of 150 steps and then started to decrease. This is due to the beam is out of the PSDs range. The measurement range is thus ±3°. However, this limitation can be overcome by rotating motors mounted on the Gimbal unit. The rotation angle will be equal to the calculated orientation angles. In this approach, the motors will turn the Gimbal unit (and thus the retroreflector) in such a way that the retroreflector will always face the direction of the incoming laser beam, contributing to zero roll, pitch and yaw angles relative to the laser beam. This method gives more flexibility to the orientation measurement.

6. CONCLUSIONS AND FUTURE WORK

Dual PSD-based orientation measurements methodology and its principle have been described. Experiment verified that this methodology is feasible in making orientation measurement. Better results could be obtained with better equipment such as higher laser power. Further experiments will be performed to measure orientation of target about the other two axes. Moreover, experiments to determine the efficiency of the method in determining the orientation of target running at different will also be established.

The sampling rate of the PSDs is 2000 Hz. This is faster than all other methods and therefore this method is capable of real time dynamic orientation measurement.

As this method has only been apply on the tracking of a mobile vehicle, the weight of the Gimbal does not affect the measurement obtained. However, in normal robot manipulator operation, the payload of the manipulator will affect the performance of the manipulator. To minimise payload due to the weight of the Gimbal unit on the robot manipulator, the Gimbal unit shown in Figure 2 has to be redesigned.

ACKNOWLEDGEMENTS

This project is partly funded by Australian Research Council (ARC) and Harold Armstrong Fund.

REFERENCES

1. H. Gander, M. Vincze, J.P. Prenninger, "An external 6-D-sensor for industrial robots," *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, Yokohama, Japan*, pp. 975-978, 1993.
2. M. Vincze, J.P. Prenninger, H. Gander, "A laser tracking system to measure position and orientation of robot end effectors under motion," *The International Journal of Robotics Research* 13(4), pp. 305-314, 1994.
3. K.M. Filz, M. Vincze, J.P. Prenninger, "Camera system to detect the orientation of a corner cube in real time," *Proceedings IEEE International Conference on Robotics and Automation* 6, pp.1713-1718, 1995.
4. J.P. Prenninger, H. Gander, M. Vincze, "Contactless position and orientation measurement of robot end-effectors," *IEEE Conference on Robotics and Automation* 1, pp. 180-185, 1993.
5. B. Shirinzadeh, "Laser interferometry-based tracking for dynamic measurements," *Industrial Robot* 25(1), pp. 35-41, 1998.
6. B. Shirinzadeh, P. L. Teoh, C. W. Foong, "Orientation measurement using vision and non-vision based techniques in laser tracking system," *30th International Symposium on Robotics, Tokyo, Japan*, pp. 317-324, 1999.
7. B. Shirinzadeh, P. L. Teoh, C. W. Foong, Y. D. Liu, "Orientation measurement technique in laser interferometry based tracking system for robot manipulator calibration," *Proceedings of Pacific Conference on Manufacturing (PCM2000), Detroit, USA* 2, pp. 788-793, 2000.
8. B. Shirinzadeh, P. L. Teoh, "A study of predictive control for laser tracking of robots," *Proceedings of Pacific Conference on Manufacturing (PCM98), Brisbane, Australia*, pp. 328-333, 1998.
9. Y. Bao, N. Fujiwara, "Dynamic measurement orientation by LTS." *Proceedings of the Japan/USA Symposium on Flexible Automation* 1, pp. 545-548, 1996.
10. Melles Griot, *Melles Griot 1995/96 Catalogue*, Melles Griot USA, 1996.

P. L. Teoh, B. Shirinzadeh, C. W. Foong, G, Alici, "The measurement uncertainties in laser interferometry-based sensing and tracking technique", Journal of Measurement, Vol. 32 No. 2, pp 135-150, September 2002.

ELSEVIER

Measurement 32 (2002) 135–150

Measurement

www.elsevier.com/locate/measurement

The measurement uncertainties in the laser interferometry-based sensing and tracking technique

Pek Loo Teoh, Bijan Shirinzadeh*, Chee Wei Foong, Gürsel Alici

Robotics and Mechatronics Research Laboratory, Department of Mechanical Engineering, Monash University, Clayton, Vic. 3800, Australia

Received 5 June 2001; received in revised form 30 November 2001; accepted 7 February 2002

Abstract

The laser interferometry-based sensing and tracking (LIST) technique can be used to perform real time position measurements of dynamic systems such as robot manipulators. These measurements are necessary to provide accurate calibration and performance measures of robot manipulators. Therefore, it is important that the LIST technique is highly accurate. However, due to the dependence of LIST technique on different sensors and transducer sub-systems, there exists some degree of uncertainty in the measurements obtained. This paper presents the measurement uncertainties associated with the LIST technique. The uncertainties contributed by different sub-systems within the LIST apparatus are analysed. A general expression for uncertainty estimation is also developed. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Laser interferometry; Laser-based sensing and tracking; Uncertainty; Robot manipulators

1. Introduction

The increase in the number of complex applications of robots in manufacturing and service industries has emphasised the need for improved positioning accuracy of robot manipulators. It is well known that the repeatability of today's industrial robots is at least an order of magnitude better than their absolute accuracy [1]. Therefore, accurate robot position and orientation measurement (pose), as well as calibration techniques are required to improve robot accuracy. The laser interferometry-based sensing and tracking (LIST) technique has been proposed [1] to provide dynamic measurements of position and orientation of the robot's end-effector for robot calibration. This technique can be further developed to precisely guide the robot tool centre point to a desired location or along a specific path. This technique is expected to provide high accuracy pose measurements in real time within a large working space at high sampling rate and with automatic target tracking capability [2].

It must be emphasised that any technique based on one or more measurements from other sub-systems involves some degree of uncertainties. Measurements made by the LIST technique rely heavily on the use of other sensors and transducers. These sensors and transducers have certain errors associated with them. Such errors may be caused by individual inaccuracy of sensors, random variations in measurands (i.e. a particular quantity to be measured at a particular condition), calibration techniques, vibration and in-

*Corresponding author. Tel.: +61-3-9905-1565.
E-mail address: bijan.shirinzadeh@eng.monash.edu.au (B. Shirinzadeh).

0263-2241/02/$ – see front matter © 2002 Elsevier Science Ltd. All rights reserved.
PII: S0263-2241(02)00006-4

complete knowledge of the effect of environmental conditions. Therefore, the errors of these components contribute to the overall uncertainty of the measurement taken by the LIST technique. An analysis is needed to identify the contribution of each source of error to the overall measurement uncertainty of the whole system. Depending on the accuracy required, appropriate corrections in the calibration of the sensors and transducers are necessary to reduce the uncertainty, and thus to improve the accuracy and repeatability of the measurements.

This paper describes the uncertainties associated with the LIST technique. The principle of LIST and the arrangement of sub-systems are first described. The source and magnitude of errors associated with each sub-system are then analysed. An uncertainty analysis approach is introduced and the overall measurement uncertainty of the LIST technique is presented.

2. Principle of laser interferometry-based sensing and tracking

Laser interferometry-based sensing and tracking of robot motion generally involves the dynamic acquisition of the positions of a robot end-effector in its workspace. It can also be used to provide orientation measurements of robot end-effector. The LIST technique uses the linear and angular displacement data, obtained from the interferometer and beam steering mechanism, respectively, to provide the position of the target retroreflector attached to the end-effector of a robot manipulator. It maintains tracking of the target retroreflector by sensing the offset of the incident and reflected laser beam from the target. It subsequently performs offset corrections by adjusting the angles of the beam steering mechanism. The functional layout of the LIST technique developed for this study [2,3] is shown in Fig. 1.

The laser interferometer used in this technique is a special variant of the Michelson laser interferometer employing the Zeeman split. A heterodyne laser beam consisting of two orthogonally polarised frequency components F_1 and F_2, offset by 20 MHz, is emitted fr⌐⌐ı the laser head. The laser beam passes through a polarisation beamsplitter in the interferometer, where it is split into a reference beam with frequency F_1 and a measurement beam with frequ y F_2. The reference beam is directed to the reference retroreflector where it is reflected and directed to the fibre optic pick-up. It is then transmitted to the measurement board on the laser interferometer controller. The measurement beam passes through a 70–30 beamsplitter before being directed to a retroreflector by the beam steering mechanism. The retroreflector is generally mounted on a target (e.g. robot's end-effector). The retroreflector is designed in such a way that the incident beam is reflected through 180° and travels back parallel to the incident beam. If the incident beam does not hit the centre of the retroreflector, there will be an offset between the incident and the reflected measurement beams.

This reflected beam follows the same path back to the 70–30 beamsplitter; 70% of this beam power is directed to the interferometer before being deflected by the polarisation beamsplitter in the interferometer and picked up by the fibre optic cable. It is then transmitted and combined with the reference beam at the laser interferometer controller. As the end-effector carrying the retroreflector is moved, the frequency F_2 of the measurement beam shifts from the frequency of the previous measurement beam based on the Doppler principle [4], yielding a frequency change of ΔF_2. In the interferometer controller, F_1 and F_2 are recombined to give an interference signal with a frequency of $F_1-(F_2\pm\Delta F_2)$. Electronics in the laser head generates a reference signal with frequency equal to F_1-F_2. The interference and reference signal are combined to determine the frequency change ΔF_2, which gives the rate of change in target displacement. This rate is then integrated to provide the relative displacement of the target. The remaining 30% of the reflected beam power is directed by the 70–30 beamsplitter perpendicular to the original laser path and travels to a 50–50 beamsplitter. This beam is further split where 50% of this beam power is incident onto a position sensitive diode (PSD), which determines the offset error of the reflected laser beam from the centre of the retroreflector. This offset error is referred to as the tracking error. The other 50% is transmitted to a CCD (Charged-Coupled Device) camera. The CCD camera connected to a frame grabber board captures the diffraction pattern of the reflected beam, which

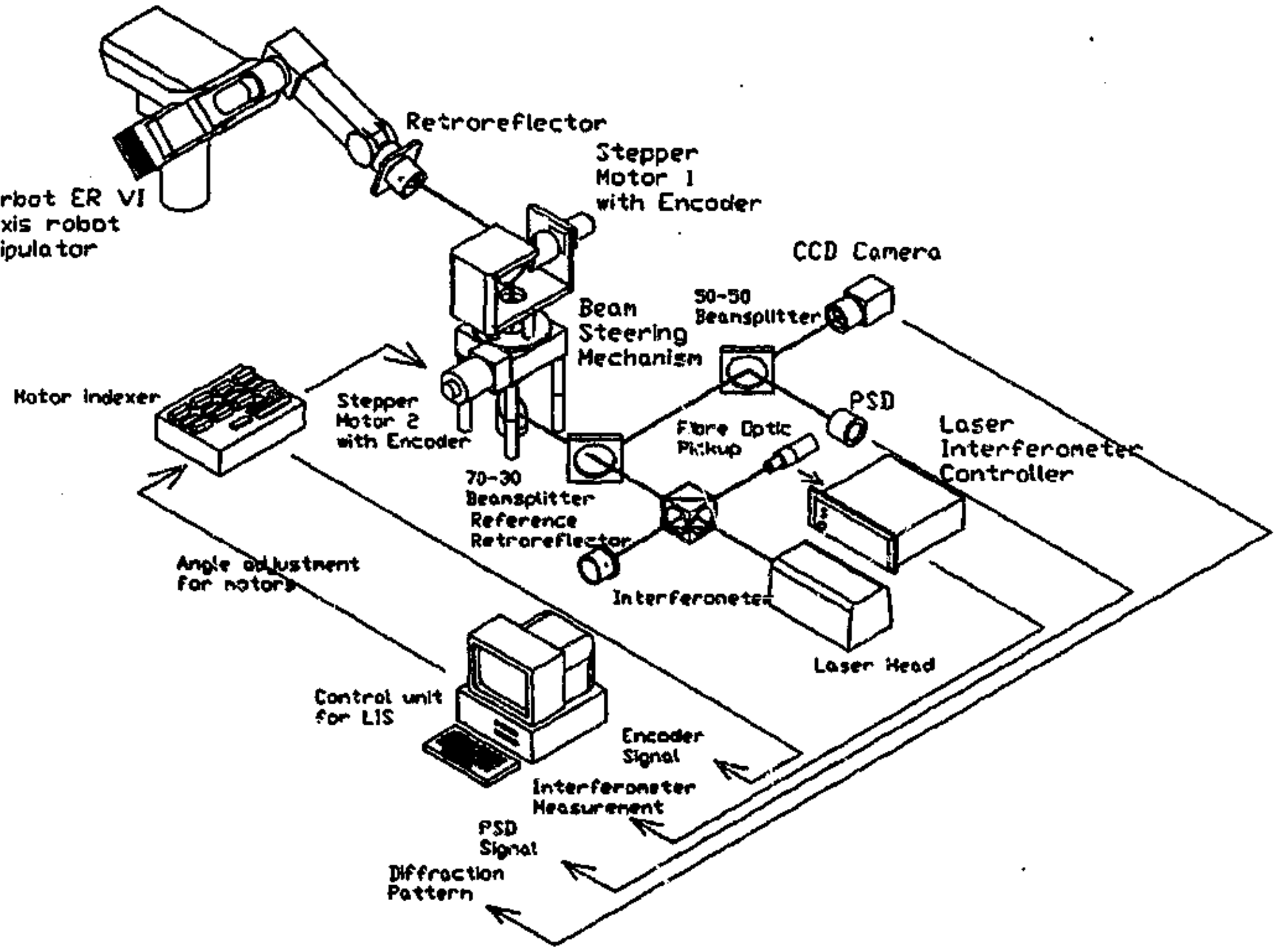


Fig. 1. Functional layout of the LIST technique.

allows the determination of the orientation of the retroreflector.

The LIST control system minimises the tracking error obtained from the PSD acquisition system in real time by signalling the motor controller which in turn rotates the axes of the beam steering mechanism, thus following the arbitrary movements of the target. Measurement of the position of the target in space is obtained from the displacement of the beam, tracking errors, and angular displacements of the axes of the beam steering mechanism. All these data are utilised within the equations describing the kinematics of the LIST apparatus. A predictive control algorithm that allows estimation of the future position of the target from the previous position, velocity and acceleration values [5] has been employed to improve the speed of tracking, and thus increase the speed of measurement.

3. Expression of uncertainty

The objective of any measurement technique is to determine the value of the measurand. In general, the result of a measurement technique is only an estimate of the true value of the measurand and is only complete when stated with the uncertainty of that estimate. The uncertainty, therefore, indicates the dispersion of the values of the measurand [6]. Some of the possible sources of uncertainty in a measurement by LIST technique include:

- unknown effects of the environment conditions on the measurement;
- error in measurement acquisition from various instruments;
- resolution of various instruments;
- approximations and averaging steps (where applicable) made in the measurement process;
- inexact values of reference materials (e.g. for calibration purposes).

The total uncertainty comprises uncertainties of many components, which may be obtained from either the results of a series of measurements or experience/other information. The former is termed Type A Uncertainty, and the latter Type B Uncertainty [6]. In order to develop a formulation for the uncertainties, it is important to establish the formulation for correlated and uncorrelated inputs. For a measurement m, whose results depend on uncorrelated input estimates $x_1, x_2, \ldots x_N$, the standard uncertainty of the measurement is obtained by appropriately combining the standard uncertainties of these input estimates. The combined standard uncertainty of the estimate m denoted by $u_c(m)$ is calculated from the following equations [6]:

$$m = f(x_1, x_2, \ldots, x_N) \tag{1}$$

$$u_c^2(m) = \sum_{i=1}^{N} \left[\frac{\partial f}{\partial x_i} \right]^2 u^2(x_i) \tag{2}$$

where f is the function of m in terms of input estimates $x_1, x_2, \ldots x_N$, and each $u(x_i)$ is a standard uncertainty which may be evaluated either from Type A Uncertainty or from Type B Uncertainty. When the input estimates are correlated, the combined variance $u_c^2(m)$ associated with the results of a measurement is determined by the following equation:

$$u_c^2(m) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j} u(x_i, x_j)$$
$$= \sum_{i=1}^{N} \left[\frac{\partial f}{\partial x_i} \right]^2 u^2(x_i) + 2 \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j} u(x_i, x_j) \tag{3}$$

where $u(x_i, x_j)$ denotes the estimated covariance associated with x_i and x_j.

4. Analysis of uncertainties

The LIST technique described in this paper consists of four different sub-systems. These sub-systems are the laser interferometer sub-system, motors and angular encoders sub-system, position sensitive diode (PSD) sub-system, and finally the orientation measurement sub-system. In this section, the uncertainties within each of these sub-systems as well as uncertainties contributed by other sources are described.

4.1. Laser interferometer

The overall accuracy of any laser interferometry apparatus is affected by a number of different factors including [7,8]:

- Environmental errors;
- Geometric errors;
- Instrument errors.

4.1.1. Environmental errors

The refractive index of the laser beam is affected by the deviations in the environmental conditions such as ambient temperature, air pressure, and humidity. A variation in the index of refraction introduces an error in the wavelength and changes the displacement measured. An error of approximately one part per million (ppm) occurs in the index of the laser used due to each of the following environmental changes [7]:

- 1 °C change in the air temperature
- 2.8 mmHg change in air pressure
- 90% change in the relative humidity

Controlling the climate will minimise the environmental sources of error.

A uniform change in temperature causes expansion or contraction of the interferometer components and introduces an error in the displacement measured. The linear interferometer used within this LIST apparatus has a temperature coefficient of less than 0.022 μm/°C [7].

4.1.2. Geometric errors

Misalignment of the optics induces errors such as cosine error, Abbe offset error and polarisation mixing error. Cosine error is a measurement error caused by an angular misalignment between the laser beam and the axis of motion of the displacement being measured. Cosine error degrades the signal received by the receiver and more importantly, reduces the accuracy of measurement because of not measuring the actual target displacement.

Abbe offset is a result of an offset between the measurement laser beam and the axis of motion of the target. Positioning the beam as close as possible to the axis of motion will reduce the Abbe offset.

Polarisation mixing error is due to the misalignment of the laser head relative to the interferometer and the imperfections in the polarisation beamsplitter. This produces a leakage of undesired polarisation state into the two polarised frequencies. Polarisation mixing will introduce distortion in the interference signal which will produce a non-linearity between the measured displacement and the actual displacement and thus affect accuracy of measurement. Fig. 2 shows a properly aligned polarising system. The transmitted and reflected beams each contain only one frequency, F_2 and F_1, respectively. For the polarisation mixing as shown in Fig. 3, the transmitted and reflected beams contain not only the frequency F_2 and F_1, respectively, but also a small portion of the each other's beam frequency. Polarisation mixing error is cyclic with a period of 360° occurring approximately every 158 nm (a quarter of the wavelength) of displacement for a double pass interferometer [7]. This cyclic error is non-cumulative.

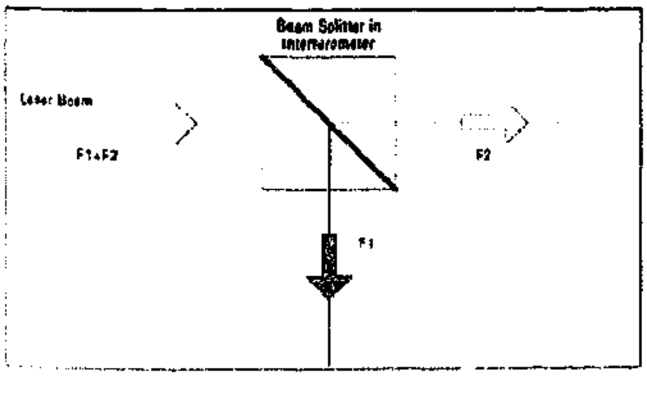


Fig. 2. Proper polarisation alignment.

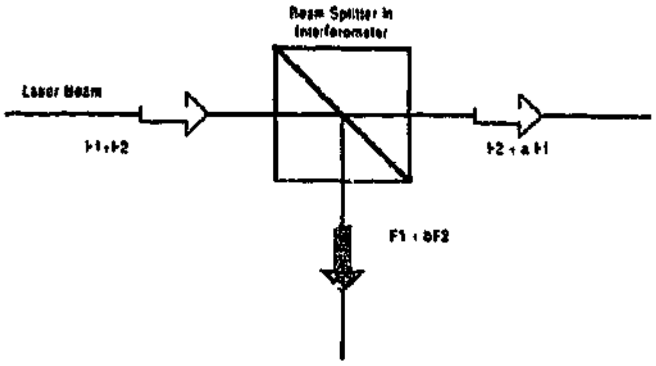


Fig. 3. Polarisation mixing.

4.1.3. Instrument errors

The laser interferometer used has a maximum electronic error of 1.3 counts. The contribution of this error to the uncertainty analysis is a product of the electronic accuracy and the optical resolution of the interferometer used. In this case, the uncertainty is 3.25 nm [7].

Due to the imperfections of the optical components and their coatings, there will again be a leakage of undesired polarisation state. This will cause polarisation mixing error of the beams within the interferometer. For a linear interferometer, this error is ±0.8 nm [7].

4.2. Motors and angular encoders

This sub-system consists of two stepper motors, two optical encoders and a motion controller card. The stepper motors provide the horizontal and vertical rotations of the beam steering mechanism, allowing the laser beam to be directed towards the centre of the retroreflector in order to minimise the tracking error determined by the PSD.

The vertical axis has an increment of 0.02°, using a stepper motor rate of 400 steps per revolution and a precision rotary table with a 45:1 gear ratio. The horizontal axis is direct drive and it has an increment of 0.007° using a step rate of 51 200 steps per revolution. For the vertical axis, a 500-line encoder with a maximum resolution of 0.72° is used. The encoder used for the horizontal axis is a 1000-line encoder with a maximum resolution of 0.36°.

Possible sources of errors are backlash due to gearing, heat generated, friction between the moving parts, shaft alignment between the stepper motors and the encoders, the weight of the mirrors causing bending and tumbling of motors [9].

4.3. Position sensitive diode (PSD)

The PSD used is of lateral effect detector type. It has four electrodes connected equidistant around its perimeter. The electrode˜ yield photo-currents corresponding to the displacement of the beam from the centre of the PSD [10]. Photo-currents from opposite pairs are converted to voltages and are processed to give the displacement of the beam in the X and Y directions from the centre of the PSD. Fig. 4 shows the schematic diagram of this detector type. Voltages sampled from each electrode are labelled V_1 through V_4 as shown in the diagram.

The photo-currents are converted to voltages and sampled by the LIST controller. Calibration had been carried out to obtain a relationship between the voltage difference of the opposite electrode pair and the displacement of the laser beam from the centre of the PSD. The experiment was conducted by mounting the PSD onto a rotational stage and then onto a three-axis translational stage. The rotational stage is to ensure the elimination of cosine error. Fig. 5 shows an example of the calibration graph.

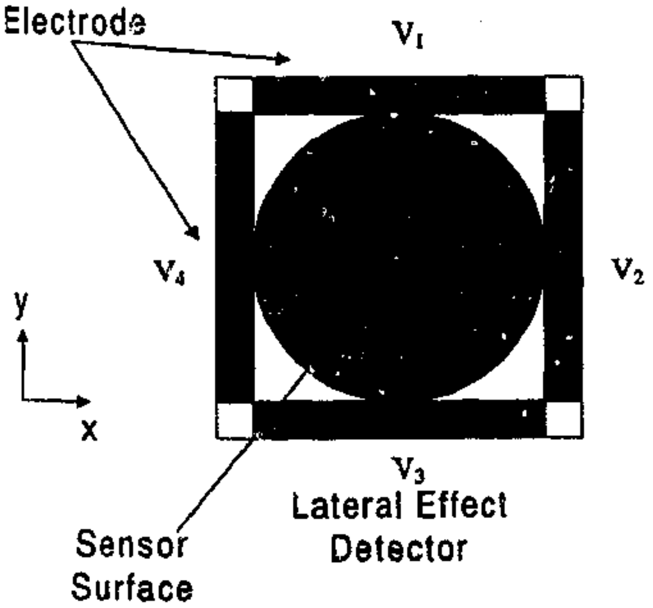


Fig. 4. Dual axis lateral effect detector.

As the beam is displaced from the centre of the PSD, the voltage difference between opposite electrode pairs $V_2 - V_4$ and/or $V_1 - V_3$ increases. When the beam is at the centre, the voltage difference between those electrode pairs is the same, thus giving zero displacement (i.e. zero tracking error). The PSD has a sampling rate of 2 kHz and a range of ±6 mm in both X and Y directions. This range is determined from the maximum allowable range between the laser beam and the centre of PSD before the change in voltage difference becomes non-linear or negligible.

The main sources of error include the resolution of the PSD and the calibration technique used. The maximum error is 16.6 μm that is calculated from the deviation of the calibrated results from the known reference.

4.4. Orientation measurement

There are two different approaches to orientation measurement in the LIST technique. These are the CCD camera-based approach and the dual PSD-based approach. Sources of errors for both approaches are described below.

4.4.1. CCD camera-based approach

As mentioned in the previous section, part of the measurement beam reflected from the retroreflector is directed to the CCD camera. The camera captures the image of the diffraction pattern of the retroreflector and a high-acquisition-rate frame grabber acquires this signal to perform image digitisation. Fig. 6 shows the image of the diffraction captured and displayed on a computer screen. It consists of a white circle on a dark background, with three dark lines intersecting each other. The intersection point is the centre point of the retroreflector and the dark lines are caused by the edges of the three mirrors on the retroreflector. The image can be simplified as shown in Fig. 7. The vectors V_i ($i = 1, 2, 3$) are the projected edge vectors of the diffraction pattern. The process of blob analysis is performed on the digitised image before the vector detection process described in Ref. [11] can be carried out using the appropriate

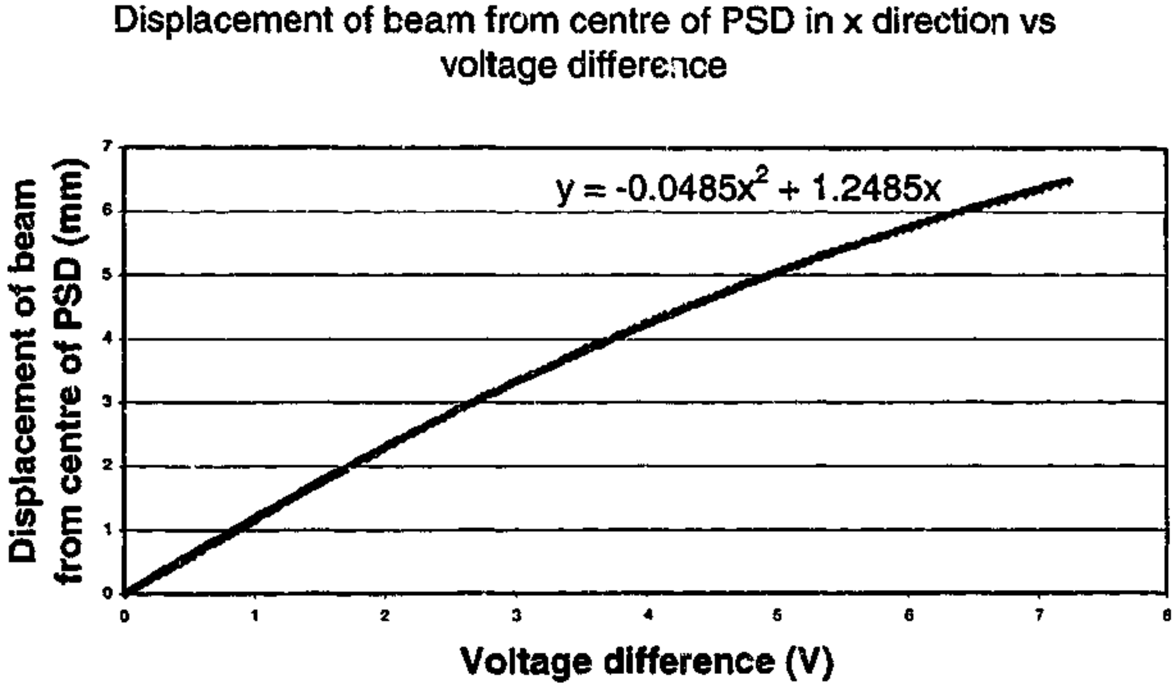


Fig. 5. Calibration graph for PSD.

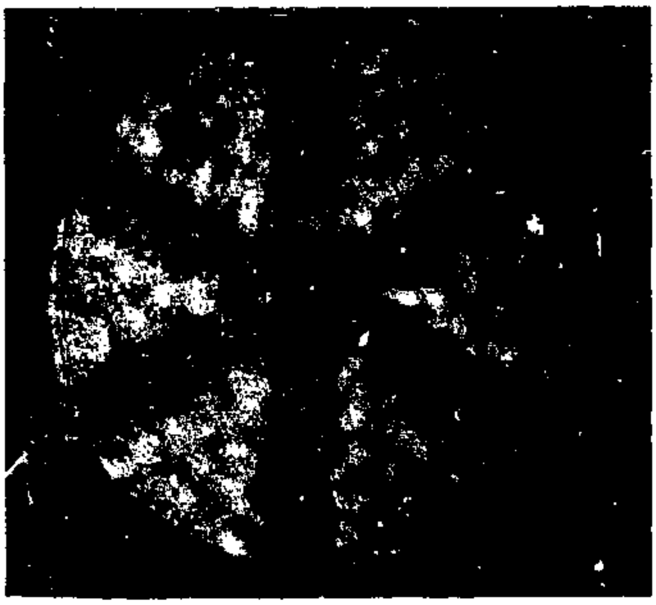


Fig. 6. Diffraction pattern of the reflected beam from the retro.eflector.

algorithm. Using the algorithm, the angle between the head of the vectors and the x-axis can thus be determined.

When the angles α, β, γ, and δ as shown in Fig. 7 have been established, the orientation (roll ϕ, pitch θ and yaw φ) of the retroreflector relative to the laser beam can be determined using the following equations [1,12]:

$$\theta = -\arcsin\left(\sqrt{\frac{2}{3\tan\alpha\tan\gamma}} - \sqrt{\frac{-1}{6\tan(\alpha+\gamma)\tan\alpha}} - \sqrt{\frac{-1}{6\tan\gamma\tan(\alpha+\gamma)}}\right) \tag{4}$$

$$\phi = \arcsin\left(\frac{1}{\cos\theta}\left(\sin\delta\sqrt{\frac{-2\cos(\gamma+\alpha)}{3\sin\alpha\sin\gamma}} - \sin(\delta+\alpha)\sqrt{\frac{\cos\gamma}{6\sin\alpha\sin(\alpha+\gamma)}} - \sin(\delta-\gamma)\sqrt{\frac{\cos\alpha}{6\sin\gamma\sin(\alpha+\gamma)}}\right)\right) \tag{5}$$

$$\varphi = \arcsin\left(\frac{1}{\cos\theta}\sqrt{\frac{-1}{2\tan(\alpha+\gamma)\tan\alpha}} - \frac{1}{\cos\theta}\sqrt{\frac{-1}{2\tan\gamma\tan(\alpha+\gamma)}}\right) \tag{6}$$

It must be emphasised that these angles are relative to home (i.e. zero) orientation. Possible sources of

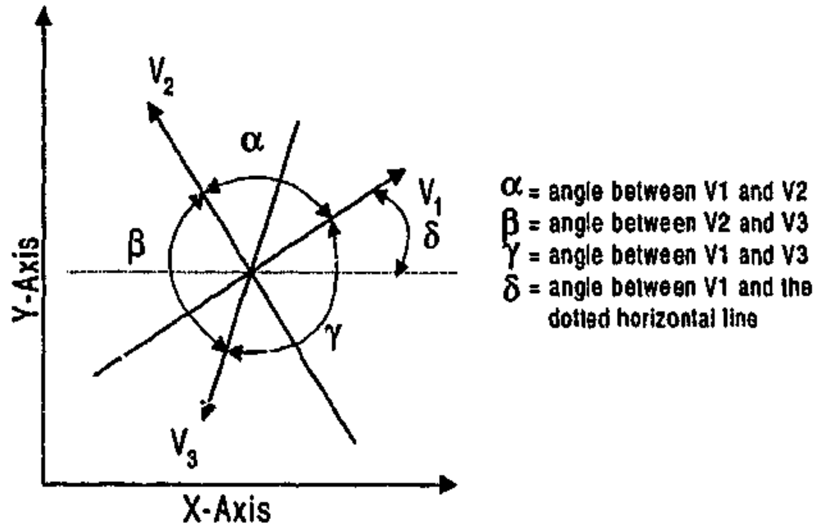

V2
V1
V3
α
β
γ
δ
Y-Axis
X-Axis
α = angle between V1 and V2
β = angle between V2 and V3
γ = angle between V1 and V3
δ = angle between V1 and the dotted horizontal line

Fig. 7. Projected edge vectors of the diffraction pattern.

errors are the orthogonality of the CCD camera to the laser beam, and the capability of the application software in detecting the vector lines and determining the angles.

4.4.2. *Dual PSD-based approach*

Another approach to orientation measurement in LIST is the dual PSD-based orientation measurement [11,13]. This approach utilises two position sensitive diodes (PSDs), a beamsplitter, and a retroreflector mounted on a specially designed Gimbal unit as shown in Fig. 8. In the CCD camera-based approach, the laser beam is pointed directly at the retroreflector mounted on the robot end-effector. However, in this case, the laser beam is directed by the beam steering mechanism at a 70–30 beamsplitter mounted on the Gimbal unit. This beam is then split into two beams; 30% of the beam power is incident on the first PSD and the remaining part is directed towards the retroreflector. The reflected beam travels back to the beamsplitter, parallel to the incident beam. At the beamsplitter, 30% of the laser beam power is again directed to the second PSD and the remaining laser beam returns to the beam steering mechanism.

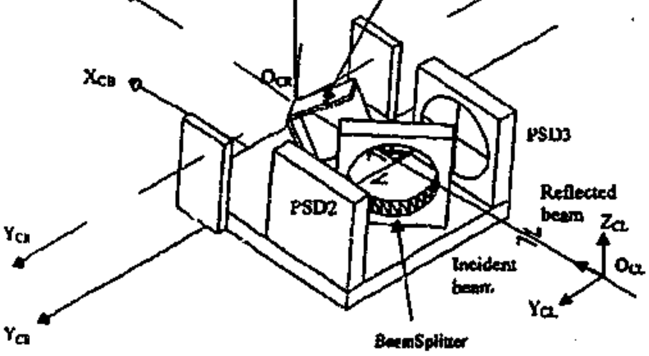


Fig. 8. Gimbal unit assembly.

A flowchart for the dual PSD-based orientation measurement algorithm is provided in Fig. 9. From the flowchart, it can be observed that the orientation of the retroreflector is determined from the measurements obtained along the x and y axes on the PSDs. The governing equations to determine the orientation of the retroreflector (co-ordinate system C_R) relative to the laser beam (co-ordinate system C_L) are thus as follows:

$$^{C_L}T_{C_R} = [Rotz(\theta_1)][Roty(\theta_2)][Rotx(\theta_3)] \tag{7}$$

where $Rotz(\theta_1)$, $Roty(\theta_2)$, and $Rotx(\theta_3)$ are rotational matrices [14], about axis z, y, and x with angles θ_1, θ_2, and θ_3, respectively.

Fig. 10 shows a top view of the optical components mounted on the gimbal unit. From Fig. 10, it can be seen from the properties of a triangle that:

$$\theta_1 = \arctan \frac{D_{2x} + D_{3x}}{2l_1 - l_2 + l_3} \tag{8}$$

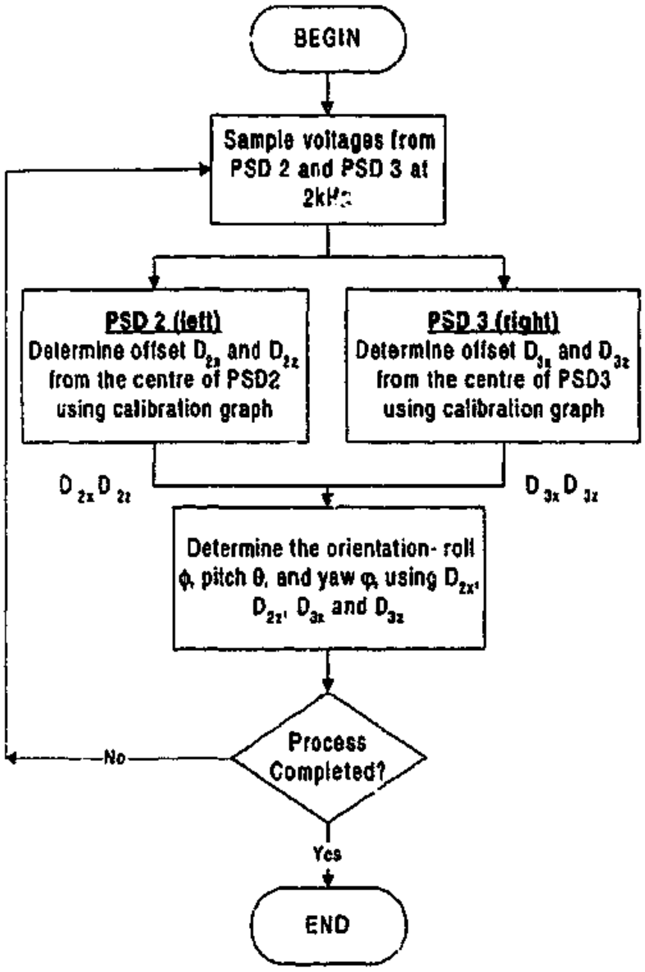

BEGIN
Sample voltages from PSD 2 and PSD 3 at 2kHz
PSD 2 (left)
Determine offset D_{2x} and D_{2z} from the centre of PSD2 using calibration graph
PSD 3 (right)
Determine offset D_{3x} and D_{3z} from the centre of PSD3 using calibration graph
D_{2x} D_{2z}
D_{3x} D_{3z}
Determine the orientation- roll φ, pitch θ, and yaw φ, using D_{2x}, D_{2z}, D_{3x} and D_{3z}
Process Completed?
No
Yes
END

Fig. 9. Flowchart for non-vision-based orientation measurement.

With similar method in evaluating θ_2,

$$\theta_2 = \arctan\frac{(D_{2z} + D_{3z})\cos\theta_1}{2l_1 - l_2 + l_3} \tag{9}$$

where l_1 is the distance between the centres of the beamsplitter and the retroreflector. l_2 and l_3 are the distances between the centre of the beamsplitter and PSD2 and PSD3, respectively. D_{2x} and D_{3x} are the detection positions of the laser beam on PSD2 and PSD3 along the x-axis, respectively. D_{2z} and D_{3z} are the detection positions of the laser beam on PSD2 and PSD3 along the z-axis, respectively.

θ_3 can be calculated from the point of intersection between the laser beam and the yz plane of coordinate system C_B, which is a stationary base of the gimbal unit in Fig. 8. The main source of error is the error inherent in the PSD. Another source of error might be the errors due to the manufacturing of the gimbal unit.

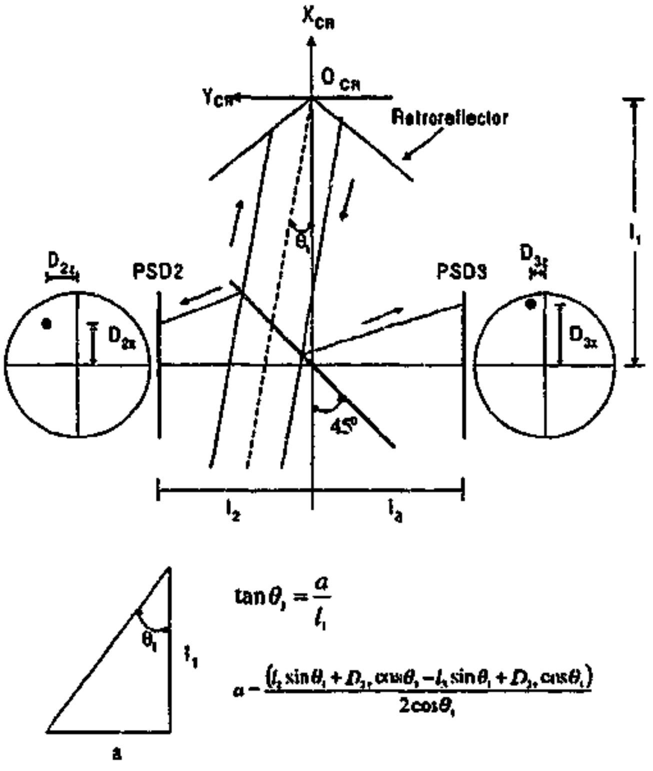

X_{CR}
O_{CR}
Y_{CR}
Retroreflector
θ_1
l_1
D_{2z}
PSD2
PSD3
D_{3z}
D_{2x}
D_{3x}
45°
l_2
l_3
θ_1
l_1
a
$\tan\theta_1 = \frac{a}{l_1}$
$a = \frac{(l_3\sin\theta_1 + D_{3x}\cos\theta_1 - l_2\sin\theta_1 + D_{2x}\cos\theta_1)}{2\cos\theta_1}$

Fig. 10. Top view of Gimbal unit.

4.5. Reflecting target

The reflecting target is the most commonly used air-path type retroreflector. Fig. 1? shows a diagram of this retroreflector. It is made up of three mirrors with high reflective indices assembled in such a way that they are orthogonal to each other. The intersection point of the three mirrors will be the centre of the retroreflector. An ideal retroreflector has the properties to ensure that a light beam incident onto any point on the retroreflector, regardless of its orientation, is reflected through 180° and the reflected beam travels back parallel to the incident beam. The centre of the retroreflector is always exactly centred between the incident and the reflected beams.

Possible errors in a retroreflector are associated with the construction of the retroreflector. The mirrors are joined together usually with adhesive. There could be a position error in joining the mirrors, and the three mirrors used might not inter-

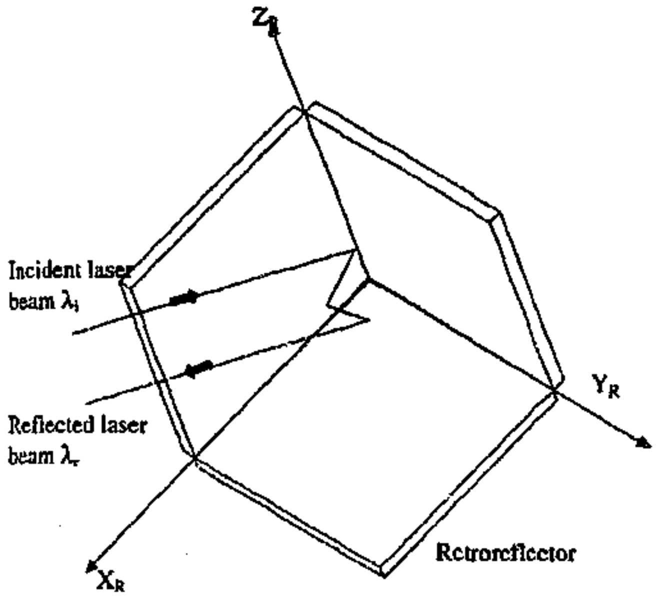

Z_R
Incident laser beam λ_i
Reflected laser beam λ_r
Y_R
X_R
Retroreflector

Fig. 11. Air-path type retroreflector.

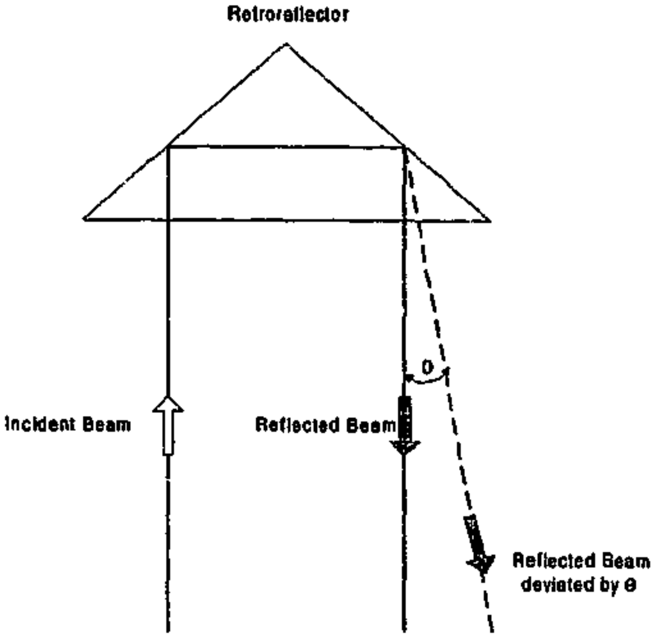


Fig. 12. Angular deviation of beam caused by retroreflector.

sect each other at a single point. This causes an offset error so that the centre of the retroreflector is not exactly centred between the incident and the reflected beams. This contributes to the uncertainty of the x, y measurement in LIST, as the offset recorded by the PSD is not the real offset due to this error. Further, there may exist an error associated with the orthogonality of the mirrors. Therefore, the beam may not be reflected through an angle of 180°, rather with a slight angular deviation. The angular deviation is specified as 5 arc seconds in this case as stated in Ref. [10]. Therefore, the reflected beam may not be parallel to the incident beam and there could be a slight change in displacement of the beam. Fig. 12 shows the error due to the imperfection in the construction of the retroreflector.

4.6. Measurement platform geometry and environmental effect

All the sub-systems including the laser interferometer and the other optics are located on an optical breadboard with M6 holes on 25-mm centres. The flatness of the breadboard will affect the accuracy of the LIST by contributing to the cosine error. Furthermore, the error between the mounting holes contributes further to the uncertainty of the results. The assembly of the beam steering mechanism also affects the accuracy due to the imperfection in the construction process. Environmental conditions may also affect the properties of materials used in the beam steering mechanism.

5. Estimation of uncertainties

The uncertainties in position measurement and orientation measurement can be estimated, provided that these measurements are independent.

5.1. Uncertainties in position measurement

The errors of the sub-systems contributing to the uncertainty of the measurement obtained using the LIST technique can be further divided into geometric and non-geometric errors. Geometric errors include mirror-positioning error, motor offset, retroreflector deviation, laser interferometer error, and PSD error. These errors can be compensated through calibration. Non-geometric errors comprise tumbling motion, motor backlash, sensitivity of the material to temperature, encoder coupling and noise. These errors are random, and thus cannot be determined.

Geometric errors can be modelled into a kinematic model for the LIST technique. The kinematic model provides a mathematical description of the path of the laser beam to the retroreflector on the robot end-effector and makes use of the coordinate systems (CS) shown in Fig. 13. To account for the geometric errors, every transformation parameter p in the kinematic model of the LIST technique consists of the ideal value given in the design specifications plus an error value, which represents the geometric error associated with that parameter. This gives:

$$p = p_{\text{ideal}} + \Delta p \tag{10}$$

By using techniques described in Ref. [1], a co-ordinate frame is placed on each of the three mirrors with the xy-plane describing the mirror surface and the z-axis directed to the blind side of the mirror. This gives the description of the mirrors relative to the reference co-ordinate frame as follows:

$$M_1 = [{}^{w}_{1}Transl(x_1,y_1,z_1)][{}^{1}_{1n'}R_{x'y'x'}][{}^{1n'}A_{\text{mirr}}] \tag{11}$$

$$M_2 = [{}^{w}_{2}Transl(x_2,y_2,z_2)][{}^{2}_{2n}Rotz(q_1)][{}^{2n}_{2n'}R_{x'y'x'}] \\ [{}^{2n'}A_{\text{mirr}}] \tag{12}$$

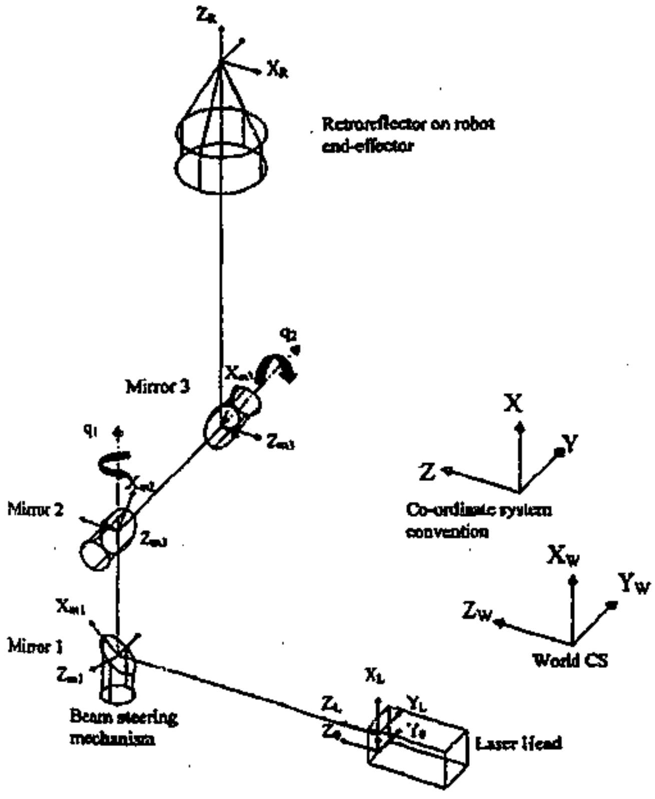


Fig. 13. Kinematic model for LIST.

$$M_3 = [{}^{w}_{2}Transl(x_2,y_2,z_2)][{}^{2}_{2n}Rotx(q_1)] \\ [{}^{w}_{3}Transl(x_3,y_3,z_3)][{}^{3}_{3n}Roty(q_2)][{}^{3n}_{3n'}R_{x'y'x'}] \\ [{}^{3n'}A_{\text{mirr}}] \tag{13}$$

where $R_{x'y'x'}$ is the rotation with respect to the translated frame tc place the xy-plane on the mirroring surface and the z-axis towards the blind side of the mirror, $Rotx(q_1)$ and $Roty(q_2)$ are rotations due to the vertical and horizontal motors and

$$A_{\text{mirr}} = [Roty(\Delta\alpha)][Rotx(\Delta\beta)][Transl(\Delta z)] \tag{14}$$

which represents the error associated with the positioning of the mirrors. It is assumed that on a perfectly plane mirror surface, the rotation about the z-axis and translation along the xy-plane will not alter the reflected beam.

Another co-ordinate frame is placed on the source of the laser beam with the z-axis aligned with the beam. This frame is translated to each mirror in turn and reflected according to the physical positions of each mirror described by the co-ordinate frame M_1, M_2, and M_3. If L_N denotes the laser frame, the reflected laser frame L_{N+1} with its origin on the mirror plane when leaving mirror $N+1$ ($N=0, 1, 2$) can be described as follows:

$$L_{N+1} = [L_N][Transl(x,y,z)][Rot(k, 180^\circ - 2\sigma)] \\ (N = 0, 1, 2) \tag{15}$$

where $L_0 = Transl(x_{L0}, y_{L0}, z_{L0})$ with respect to the reference co-ordinate frame, k is the axis about which the rotation is made and can be derived from the vector product of the z-axis of the laser frame to the z-axis of the mirror frame, and σ is the angle between the z-axis of the laser frame and the z-axis of the mirror frame calculated from the scalar product of the two axes.

After reflection from the third mirror, the beam is directed towards the target retroreflector. This is a simple translation along the beam and can be described as follows:

$$L_R = [L_4]\left[Transl\left(\frac{x_{PSD}}{2}, \frac{y_{PSD}}{2}, l_{\text{interferometer}} \\ - p_{\text{laser}}\right)\right] \tag{16}$$

where X_{PSD} and Y_{PSD} is the measurement recorded on the PSD when the beam does not hit the centre of the retroreflector, $l_{\text{interferometer}}$ is the beam displacement measured by the laser interferometer and p_{laser} is the length of the laser path from the source to the third mirror.

The complete model consists of parameters with corresponding geometric errors. These errors can be compensated through calibration by comparing the position found using the model with those obtained from a known reference. The difference in position can be related to the system parameters by J, the Jacobian matrix, which represents a differential change in position of the modelled transformation ${}^{W}L_{R}$ with respect to differential change in parameter p.

$$\Delta x = J\,\Delta p \tag{17}$$

where Δx is the differential change in position and orientation between the measured and the calculated value using the modelled transformation ${}^{W}L_{R}$, and Δp is the differential change in parameters p.

From Eq. (17), the generalised inverse of J can be found and by using the linear least squares method, and thus Δp can be found by the following formulation [1,15]:

$$\Delta p = J^{+}\,\Delta x \tag{18}$$

where J^{+} is the generalised inverse or commonly known as the Moore–Penrose inverse of the Jacobian matrix. With Δp determined for every parameter, the geometric errors can be reduced or eliminated.

With the geometric errors calibrated out, the uncertainty of the measurements obtained using the LIST technique can be determined by the following two approaches.

5.1.1. Uncertainties based on experimentation

The uncertainty in position measurement caused by the non-geometric errors can be obtained by first comparing the position measured in n repeated measurements of the positions of known references (generally $n > 100$) using the calibrated kinematic model to the reference values. The difference between the measured and the reference values is given by:

$$\delta_{\text{position}} = X_{\text{model}} - X_{\text{reference}} \tag{19}$$

where X_{model} is the positions determined by the kinematic model, $X_{\text{reference}}$ is the reference positions and δ_{position} is the differences between the positions determined by the kinematic model and the reference positions. From Ref. [6]

$$u^2(X_{\text{model}}) = u^2(X_{\text{reference}}) + s^2(\bar{\delta}_{\text{position}}) \tag{20}$$

where $u(X_{\text{model}})$ is the uncertainty of the position determined by the kinematic model, $u(X_{\text{reference}})$ is the uncertainty of the reference position and $s^2(\bar{\delta}_{\text{position}})$ is called the experimental variance of the mean.

For a reference with resolution r, the uncertainty of the reference is given by:

$$u(X_{\text{reference}}) = 0.29r \tag{21}$$

To find $s^2(\bar{\delta}_{\text{position}})$, the differences δ_{position} obtained from Eq. (19) are tabulated in a distribution graph. The variance of the distribution is given by:

$$s^2(\delta_k) = \frac{1}{n-1}\sum_{k=1}^{n}(\delta_k - \bar{\delta}_{\text{position}})^2 \tag{22}$$

where δ_k is the individual observation of the difference and n is the number of individual observations. $s^2(\delta_k)$ is called the experimental variance of the n observations and $\bar{\delta}_{\text{position}}$ represents the average of the n observations and is given by the following equation:

$$\bar{\delta}_{\text{position}} = \frac{1}{n}\sum_{k=1}^{n}(\delta_k) \tag{23}$$

The experimental variance of the mean, $s^2(\bar{\delta}_{\text{position}})$ is given by:

$$s^2(\bar{\delta}_{\text{position}}) = \frac{s^2(\delta_k)}{n} \tag{24}$$

The standard uncertainty of position measurement, $u(X_{\text{model}})$ is given by the following equation:

$$u(X_{\text{model}}) = \sqrt{u^2(X_{\text{reference}}) + s^2(\bar{\delta}_{\text{position}})} \tag{25}$$

5.1.2. Uncertainties based on model verification

Another approach to uncertainty calculation is based on assuming that all the geometrical errors are compensated in the calculations. With this assumption, the position of the retroreflector can be determined by translating the co-ordinate frame of the laser beam to each mirror in turn and rotating the frame about a particular axis by appropriate angles. This is shown in Fig. 14.

With this method, the position of the retroreflector with respect to the world reference can be written as:

$$^{W}A_{R} = A_0 A_1 A_2 A_3 A_R \qquad (26)$$

where

$$A_0 = transl(x_{X0}, y_{Y0}, z_{Z0})$$
$$A_1 = [transl(z_0; a_{Z1})][Roty(90°)]$$
$$A_2 = [transl(z_1; a_{Z2})][Rotx(90°)][Roty(-q_1)]$$
$$A_3 = [transl(z_2; a_{Z3})][Rotx(90°)][Roty(q_2)]$$
$$A_R = [transl(z_3; a_{ZR})][transl(x_3; a_{XR})][transl(y_3; a_{YR})]$$

a_{XR} and a_{YR} are deviations from the centre of the retroreflector when the laser beam does not hit the centre of the retroreflector. Both can be determined from the error offset detected by the PSD, as follows:

$$a_{XR} = \frac{X_{PSD}}{2} \qquad (27)$$

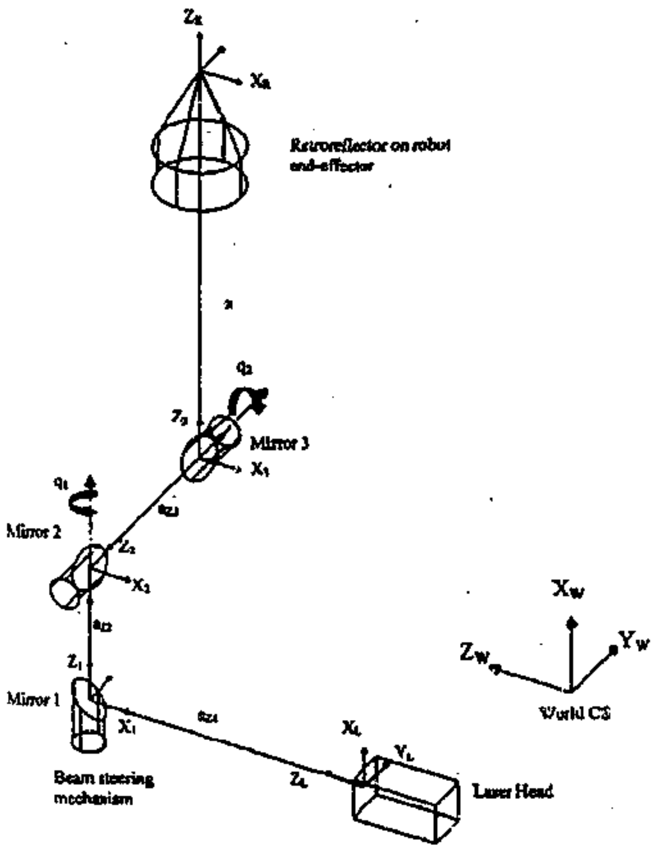


Fig. 14. Kinematic model of LIST showing path of laser beam.

$$a_{YR} = \frac{Y_{PSD}}{2} \qquad (28)$$

where X_{PSD} and Y_{PSD} are the laser beam offset error detected by the PSD along the PSD's X ana Y directions, respectively. a_{ZR} is the displacement of the beam from the third mirror to the retroreflector and can be calculated from the following equation:

$$a_{ZR} = l_{\text{interferometer}} - p_{\text{laser}} \qquad (29)$$

where $l_{\text{interferometer}}$ is the interferometer reading and p_{laser} is the path of the laser from the laser head to the third mirror.

From the $^{W}A_{R}$ matrix given by Eq. (26), the x, y and z positions of the retroreflector are:

$$x = -\sin(q_1)a_{XR} + \cos(q_1)a_{ZR} + a_{Z2} + a_{X0} \qquad (30)$$

$$y = \sin(q_1)\cos(q_2)a_{XR} + \sin(q_1)\sin(q_2)a_{ZR} + \cos(q_1)a_{YR} + \cos(q_1)a_{Z3} + a_{Y0} \qquad (31)$$

$$z = -\cos(q_1)\cos(q_2)a_{XR} - \cos(q_1)\sin(q_2)a_{ZR} + \sin(q_1)a_{YR} + \sin(q_1)a_{Z3} + a_{Z1} + a_{Z0} \qquad (32)$$

where a_{X0}, a_{Y0}, a_{Z0} are the x, y, and z positions of the laser head co-ordinate frame with respect to the reference co-ordinate frame.

By making use of Eq. (2), the variance of the position of the retroreflector can be calculated by the following equations:

$$u^2(x) = \left[\frac{-X_{PSD}}{2}\cos(q_1) - (l_{\text{interferometer}} - p_{\text{laser}})\sin(q_2)\right]^2 u^2(q_2) + \left[-\frac{1}{2}\sin(q_2)\right]^2 u^2(X_{PSD}) + \left[-\frac{1}{2}\sin(q_2)\right]^2 u^2(Y_{PSD}) + [\cos(q_2)]^2 u^2(l_{\text{interferometer}}) \qquad (33)$$

$$u^2(y) -$$

$$\begin{bmatrix} \frac{X_{PSD}}{2}\cos(q_1)\cos(q_2) - \frac{V_{PSD}}{2}\sin(q_1) \\ + (l_{interferometer} - p_{laser})\cos(q_1)\sin(q_2) - a_{23}\sin(q_1) \end{bmatrix}^2 u^2(q_1)$$

$$+ \left[\frac{-X_{PSD}}{2}\sin(q_1)\sin(q_2) + (l_{interferometer} - \right.$$

$$\left. p_{laser})\sin(q_1)\cos(q_2) \right]^2 u^2(q_2)$$

$$+ \left[-\frac{1}{2}\sin(q_1)\cos(q_2) \right]^2 u^2(X_{PSD})$$

$$+ \left[-\frac{1}{2}\sin(q_1)\cos(q_2) \right]^2 u^2(Y_{PSD})$$

$$+ [\sin(q_1)\sin(q_2)]^2 u^2(l_{interferometer}) \quad (34)$$

$$u^2(z) =$$

$$\begin{bmatrix} \frac{X_{PSD}}{2}\sin(q_1)\cos(q_2) + \frac{Y_{PSD}}{2}\cos(q_1) \\ + (l_{interferometer} - p_{laser})\sin(q_1)\sin(q_2) + a_{23}\cos(q_1) \end{bmatrix}^2 u^2(q_1)$$

$$+ \left[\frac{-X_{PSD}}{2}\cos(q_1)\sin(q_2) - (l_{interferometer} - p_{laser})\cos(q_1)\cos(q_2) \right]^2 u^2(q_2)$$

$$+ \left[-\frac{1}{2}\cos(q_1)\cos(q_2) + \frac{1}{2}\sin(q_1) \right]^2 u^2(X_{PSD})$$

$$+ \left[-\frac{1}{2}\cos(q_1)\cos(q_2) + \frac{1}{2}\sin(q_1) \right]^2 u^2(Y_{PSD})$$

$$+ [-\cos(q_1)\sin(q_2)]^2 u^2(l_{interferometer}) \quad (35)$$

where $u(q_1)$ and $u(q_2)$ are uncertainties of the encoder readings, $u(X_{PSD})$ and $u(Y_{PSD})$ are uncertainties of the error offset measured by the PSD that take into account the uncertainties caused by the PSD error and the retroreflector error, and $u(l_{interferometer})$ is the uncertainty of the laser interferometer.

The standard uncertainty of x, y, and z measurements can be determined by the square root of Eqs. (33)–(35) as follows:

$$u(i) = \sqrt{u^2(i)} \quad (i = x, y, z) \quad (36)$$

It must be noted that all the uncertainties of position determined using this approach are dependent on the instantaneous rotation of the motors and the displacement of the laser beam.

5.2. Uncertainties in orientation measurement

To determine the uncertainty of the CCD camera-based orientation measurement approach, the measured values must be compared to a known reference.

$$\theta_{orientation} = \theta_{measure} - \theta_{ref} \quad (37)$$

where $\theta_{measure}$ is the angle determined by Eqs. (4)–(6), $\theta_{orientation}$ is the recorded angular difference, and θ_{ref} is the reference angle of the known reference.

The uncertainty of the vision-based orientation measurement, $u(\theta_{measure})$ is given by:

$$u(\theta_{measure}) = \sqrt{s^2(\bar{\theta}_{orientation}) + u^2(\theta_{ref})} \quad (38)$$

where $s^2(\bar{\theta}_{orientation})$ is calculated using Eqs. (22)–(24) by replacing δ_k with θ_k and $\bar{\delta}_{position}$ with $\bar{\theta}_{orientation}$. $u(\theta_{ref})$ is determined from Eq. (21), by replacing $X_{reference}$ with θ_{ref}.

For the dual PSD-based orientation measurement approach, the uncertainty can be estimated by the method described above or it can be calcuiated using the methods described in Section 3 on Eqs. (8) and (9) as follows:

From Eq. (8), assume

$$n_1 = \tan\theta_1 = \frac{D_{2x} + D_{4x}}{2l_1 - l_2 + l_3}$$

$$u^2(n_1) = \left[\frac{1}{2l_1 - l_2 + l_3} \right]^2 u^2(D_{2x})$$

$$+ \left[\frac{1}{2l_1 - l_2 + l_3} \right]^2 u^2(D_{3x})$$

$$+ \left[\frac{2(D_{2x} + D_{3x})}{(2l_1 - l_2 + l_3)^2} \right]^2 u^2(l_1)$$

$$+ \left[\frac{-1(D_{2x} + D_{3x})}{(2l_1 - l_2 + l_3)^2} \right]^2 u^2(l_2)$$

$$+ \left[\frac{(D_{2x} + D_{3x})}{(2l_1 - l_2 + l_3)^2} \right]^2 u^2(l_3) \quad (39)$$

From Eq. (9), assume

$$n_2 = \tan\theta_2 = \frac{D_{2z} + D_{3z}}{2l_1 - l_2 + l_3}\cos\theta_1$$

$$u^2(n_2)=\left[\frac{\cos\theta_1}{2l_1-l_2+l_3}\right]^2 u^2(D_{2Z}) + \left[\frac{\cos\theta_1}{2l_1-l_2+l_3}\right]^2 u^2(D_{3Z}) + \left[\frac{2(D_{2Z}+D_{3Z})\cos\theta_1}{(2l_1-l_2+l_3)^2}\right]^2 u^2(l_1) + \left[\frac{-1(D_{2Z}+D_{3Z})\cos\theta_1}{(2l_1-l_2+l_3)^2}\right]^2 u^2(l_2) + \left[\frac{(D_{2Z}+D_{3Z})\cos\theta_1}{(2l_1-l_2+l_3)^2}\right]^2 u^2(l_3) + \left[\frac{-(D_{2Z}+D_{3Z})\sin\theta_1}{2l_1-l_2+l_3}\right]^2 u^2(\theta_1) \quad (40)$$

The uncertainty of the angles can be found from the arctan of the square root of the variances given by Eqs. (39) and (40):

$$u(\theta_i)=\tan^{-1}[u(n_i)] \quad (i=1,2) \quad (41)$$

It must be noted that the uncertainty in θ_1, which is the rotation of the gimbal about the z-axis in Fig. 8, is dependent only on the geometry of the gimbal and the PSD measurements. However, the uncertainty in θ_2, which is the rotation of the gimbal about the y-axis in Fig. 8, is dependent on θ_1 as well as the uncertainty in θ_1.

6. Conclusions and future work

In this paper, the sources of errors associated with the LIST technique have been identified. An approach to analyse the overall uncertainties of the measurement made by the LIST technique has also been presented. It is desirable that measurements obtained using the LIST technique have a very high accuracy, at least better than the accuracy of the dynamic systems to be measured in order to carry out calibration. This requires that the uncertainty of the measurements made by the LIST technique due to geometric and non-geometric or any other errors to be less than the accuracy desired. This knowledge of uncertainty and the analysis method allow the correct selection of individual sub-system devices for the LIST technique in order to improve or maintain the required accuracy.

From Eqs. (25) and (38), it can be seen that the uncertainty in position as well as in orientation measurements depends on the uncertainty in the reference used in the calibration of the LIST apparatus. Therefore, it is vital that the reference position measurement device has a high resolution. Further, the dependency on δ_{position} and $\theta_{\text{orientation}}$, which represent the difference between the measured and reference value, emphasises the importance of the repeatability of the measurement made by the LIST technique. Therefore, the apparatus has to have an acceptable repeatability that is lower than the average of δ_{position} and $\theta_{\text{orientation}}$.

Future work includes the calibration of the LIST apparatus using a coordinate measuring machine in order to verify the uncertainties determined from the analysis presented above.

Acknowledgements

This project is partly funded by Australian Research Council (ARC), and Harold Armstrong Fund.

References

[1] M. Vincze, J.P. Prenninger, H. Gander, A laser tracking system to measure position and orientation of robot end effectors under motion, Int. J. Robotics Res. 13 (4) (1994) 305–314.

[2] B. Shirinzadeh, P.L. Teoh, C.W. Foong, Y.D. Liu, A strategy for accurate guidance of a manipulator using laser interferometry-based sensing technique, Sensor Rev. 19 (4) (1999) 292–299.

[3] B. Shirinzadeh, Laser interferometry-based tracking for dynamic measurements, Ind. Robot 25 (1) (1998) 35–41.

[4] F. Demarest, High-resolution, high-speed, low data age uncertainty, heterodyne displacement measuring interferometer electronics, Meas. Sci. Technol. 9 (7) (1998) 1024–1030.

[5] B. Shirinzadeh, P.L. Teoh, A study of predictive control for laser tracking of robots, in: Proceedings of Pacific Conference on Manufacturing (PCM98), Brisbane, Australia, 1998, pp. 328–333.

[6] ISO, Guide to Expression of Uncertainty in Measurement, International Organisation for Standardisation, Switzerland, 1993.

[7] Zygo Corporation, ZMI Optics Guide OMP-0326L, Zygo Corporation, USA, 2000.

[8] W. Augustyn, P. Davis, An analysis of polarization mixing errors in distance measuring interferometers, J. Vac. Sci. Technol. B 8 (6) (1990) 2032–2036.

[9] M. Vincze, J. Prenninger, H. Gander, A model of tumbling to improve robot accuracy, J. Mech. Machine Theory 30 (6) (1995) 849–859.

[10] Melles Griot, Melles Griot 1995/96 Catalogue, Melles Griot, USA, 1996.

[11] B. Shirinzadeh, P.L. Teoh, C.W. Foong, Orientation measurement using vision and non-vision based techniques in laser tracking system, in: 30th International Symposium on Robotics, Tokyo, Japan, 1999, pp. 317–324.

[12] J. Prenninger, H. Gander, M. Vincze, Contactless position and orientation measurement of robot end-effectors, IEEE Conf. Robotics Autom. 1 (1993) 180–185.

[13] Y. Bao, N. Fujiwara, Dynamic measurement orientation by LTS, in: Proceedings of the Japan/USA Symposium on Flexible Automation, Vol. 1, 1996, pp. 545–548.

[14] J. Craig, Introduction to Robotics, 2nd Edition, Addison-Wesley, 1991, pp. 391.

[15] A. Ben-Israel, T.N.E. Greville, Generalised Inverses: Theory and Applications, Wiley-Interscience, 1974, pp. 7–38.

P. L. Teoh, B. Shirinzadeh, G. Alici, "Experimental analysis of laser interferometry-based sensing and measuring technique for a 3D dynamic positioning system", Proceedings of Pacific Conference on Manufacturing (PCM2002), Vol. 2, pp. 882-887, November 2002.

Experimental Analysis of Laser Interferometry-based Sensing and Measuring Technique for a 3D Dynamic Positioning System

P.L. TEOH, B. SHIRINZADEH and Gürsel ALICI
Robotics and Mechatronics Research Laboratory
Department of Mechanical Engineering
Monash University
Clayton, VIC 3800
Australia
Email: bijan.shirinzadeh@eng.monash.edu.au

ABSTRACT

Kinematic model is generally used in the operation of robotic devices such as a robot manipulator to accurately position the end-effector onto a commanded position. However, the accuracy of this model is generally low, as it is dependent on the accuracy of the parameters used in the model. Different methods of calibration have been developed to improve the accuracy of the model. Laser Interferometry-based Sensing and Measuring (LISM) Technique was originally proposed for this purpose. In this paper, a closed-loop control of robotic device using LISM technique is presented. Feedback from the LISM unit is used to provide a position compensation specification to the robotic device used to ensure high accuracy positioning. Experimental implementation of the strategy on a 3D positioning system is presented and the results are analysed to examine the effectiveness of the strategy.

1. INTRODUCTION

In the past two decades, there have been significant increases in the automation of operations using programmable robotic devices such as robot manipulators in the manufacturing and service industries. These devices are used to perform high precision and complex operations. The increase in complexity of these operations has emphasised the need for improved positioning accuracy of the robotic devices used. This is especially true in high-tech manufacturing industries where higher accuracy in the manufacturing process will provide a better competitive advantage.

Robotic devices consist of rigid links that are connected to each other with joints (revolute or prismatic). Actuators are attached to these joints to provide relative motion of neighbouring links by changing the joint angles for revolute joint and joint offset for prismatic joint. These motions move the end-effector to a desired location in space. Sensors such as encoders are usually instrumented to each joint to allow the actual joint motion to be measured [1]. In normal robotic operation, the end-effector (i.e. the free end) of the robotic device is required to be positioned accurately at a desired location in space or along a specific path with respect to a reference point in the work cell. The desired locations in space are usually specified in Cartesian co-ordinates (i.e. x, y, and z). This leads to the use of kinematic models in the robotic control system to convert the user-input Cartesian units into a set of joint angles. The kinematic model uses the joint positions determined from the axes encoders to provide the three dimensional Cartesian co-ordinates of the manipulator. The accuracy of such a model relies heavily on the accuracy of the parameters used such as link lengths. The accuracy is also affected by the geometric (e.g. bending) and non-geometric (e.g. temperature) errors. Further, joint backlash can also play an important role in such errors. Therefore, it is well known that, in general, the absolute accuracy of today's robots is at least a magnitude lower than its repeatability [2]. Accurate position measurements and calibration techniques are often required to improve the accuracy. Laser Interferometry-based Sensing and Measurement (LISM) technique has been proposed for this purpose. This technique can provide dynamic position measurement in real time and has high accuracy, a large operation range, a high sampling rate and automatic target tracking [3, 4].

However, only some geometric errors can be calibrated, as these are the errors that can be measured and modeled into the kinematic model. Non-geometric errors such as joint backlash, tumbling, and effect of environmental conditions on material properties as well as geometric error such as bending are not known or difficult to model. An external sensing unit with high accuracy is, therefore, required to provide the end-effector's position data with a closed-loop position control system in order to improve the positioning accuracy of the robotic devices.

In this paper, a closed-loop control methodology using LISM technique is presented. An experimental implementation of the strategy on a 3D positioning system is provided and the results are analysed to examine the effectiveness of the strategy.

2. PRINCIPLE OF LISM TECHNIQUE

LISM technique generally involves the dynamic acquisition of the three dimensional position of an end-effector in its workspace relative to a world reference [3, 4, 5, 6, 7]. The LISM technique uses the angular and beam displacement data, obtained from the beam steering mechanism and the interferometer, respectively, to provide the position of the target retroreflector attached to the end-effector of the manipulator. It maintains tracking of the target by sensing the offset of the incident and reflected beam. The tracking is carried out by adjusting the angles of a beam steering mechanism. The LISM apparatus used in this study is a Leica LT500 Laser Tracker. A functional layout of the tracker is provided in Figure 1.

In this layout, the laser beam (heterodyne type) generated by the HeNe laser head is split into a reference beam and a measurement beam. The reference beam is directed to the measurement board in the laser interferometer controller via the receiver. This beam will later be compared with the returning measurement beam, whose frequency will be Doppler-shifted, to determine the distance between the target (i.e. a retroreflector mounted on the robot end-effector) and the laser head. The measurement beam emitted from the interferometer travels through a main beam splitter before entering the two-axes beam steering mechanism. The beam steering mechanism consists of a rotating mirror, which is driven by two motors – one rotate about the vertical axis and the other about the horizontal axis. Encoders are attached to the motors to provide the angular displacement of the motors. The beam is directed to the target retroreflector by rotating the appropriate motors. Once the beam hits the target retroreflector, the beam is reflected through 180° and it travels back parallel with the incident beam. There will exist an offset between the incident and the reflected measurement beam if the beam does not hit the centre of the retroreflector.

The reflected beam then travels through the mirrors and back to the main beam splitter. Part of the beam power is diverted to a two-axes photosensor (PSD). The reflected beam position from the centre of the PSD is acquired and the offset, referred to as the tracking error, is recorded. The remaining portion of the beam power will be combined with the reference beam in the interferometer and the displacement of the beam can be determined. The Leica control system minimises the tracking error by rotating the mirror by corrective angles calculated using the tracking error and the beam displacement, thus following the arbitrary movements of the target. Measurement of the position of the target in 3D space is obtained from the interferometer measurement, tracking errors, angular displacements of the two motors, and the kinematics of the LISM apparatus. The tracker is capable of position measurement of target moving up to 6m/s at an accuracy of ±10μm/m in real time [8].

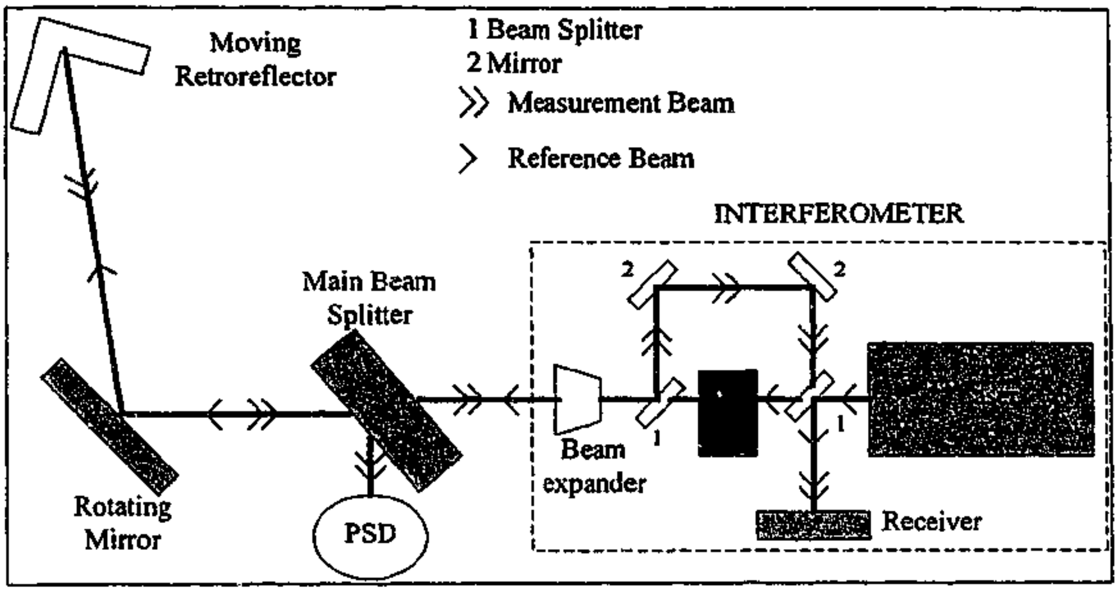


Figure 1: Functional layout of Leica LT500 Laser Tracker

3. CLOSED-LOOP CONTROL OF ROBOTIC DEVICES

3.1 Principle

The closed-loop control proposed is a control methodology where the LISM apparatus is used as a position sensor to measure the Cartesian-based position of the end-effector in real time. The LISM apparatus will maintain tracking of the robot end-effector until it reaches the final position or via points along a path. There may be discrepancy between this final position commanded by the robot controller and the desired position. Moreover, there may be vibration along the path of motion, which deviates the position of the end-effector from the via points. The LISM apparatus is implemented as a processing unit to determine the offset error and a signal will be sent to the robot controller to move the robot by the calculated amount. This can ensure that the position of the end-effector is within a certain error-limit and thus can improve the positioning accuracy of the robotic device. A flowchart of the control algorithm is provided in Figure 2.

3.2 Control Algorithm

In this methodology, the robotic device will first be commanded to the home position. The current robot position measured by the LISM control unit will be initialised to be the zero reference. The LISM control unit will then be provided the desired position P_D for the manipulator. The current position P_C of the robot manipulator will be compared with the desired position to determine the absolute offset error $|(P_D-P_C)|$. When the offset error is more than a predefined error-limit, the LISM control unit will send the offset error to the robot controller. The robot controller will calculate the necessary control signals for the actuators in order to position the end-effector to the desired location. When the robot has moved, the LISM control unit will maintain tracking of the robot end-effector. The current position P_C is updated through the measurements made by the LISM apparatus and the process is repeated until user intervention

Another feature of the control algorithm is to check the motion status of the robotic device continuously. This is called the motion status detection processes. A shown in Figure 2, there are two status that this process is monitoring. There are whether the robot has stopped or whether a via point has been reached. In both cases, an interruption signal will be sent to the LISM control unit to update the current position P_C. If the robot has stopped, the offset error represents the discrepancy between the robot final position P_{FP} and the desired position P_D. This offset error will be sent to the robot controller for position correction if it is greater than a predefined error-limit. This is repeated until the robot stops at the position where the offset error is less than the predefined error-limit. This follows that any steady-state error in the end-effector for position is eliminated. If a via point has been reached, the offset error will then represent the discrepancy between the via points' position P_{VP} and the robot current position P_C. If this error is greater than a predefined error-limit, this offset error will be sent to the robot controller and correction is made while the robot is moving to the next via point. This is to minimize deviation of the end-effector from the desired path due to vibration or bending of the link.

The tracking process will maintain tracking of the robot manipulator even when the robot has stop at position within the predefined error-limit. This is to account for subsequent environmental effect on the robot that will move the end-effector away from the desired position.

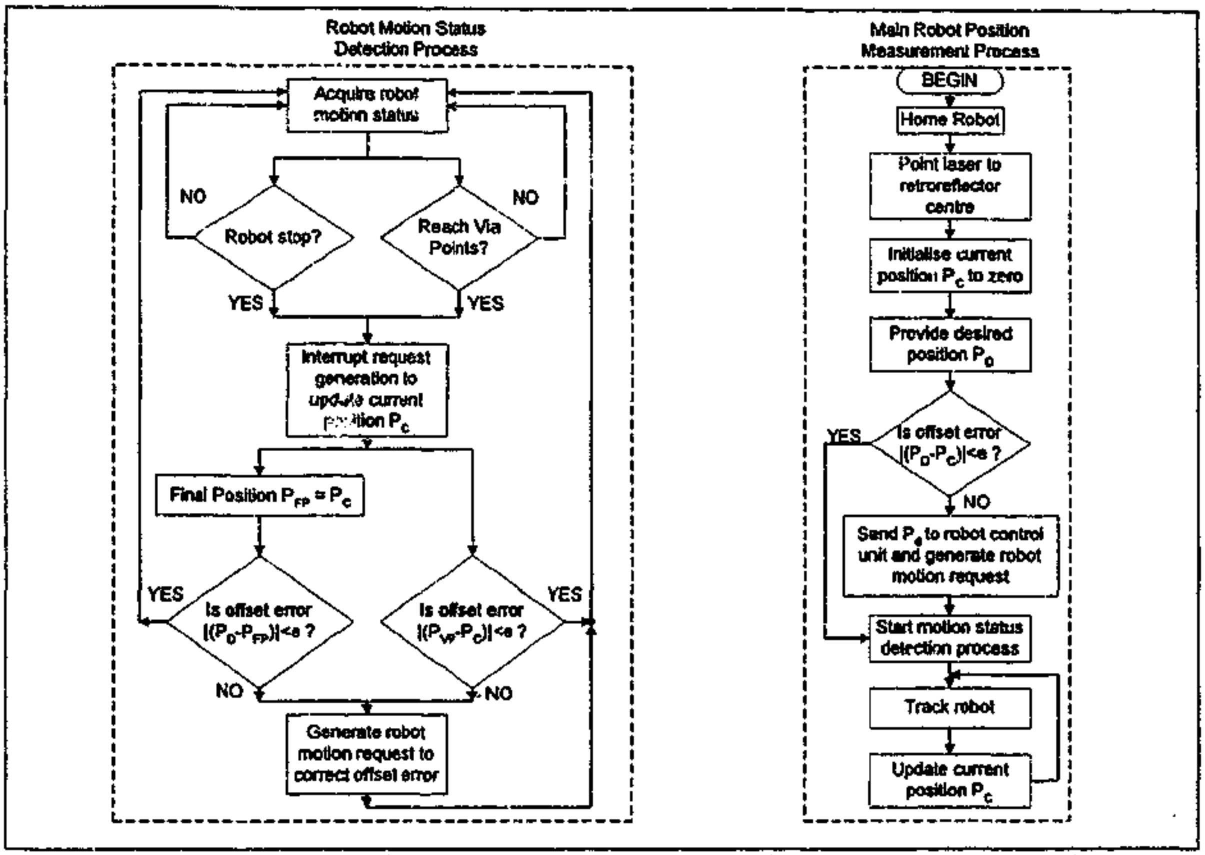

Robot Motion Status Detection Process
Acquire robot motion status
NO
Robot stop?
NO
Reach Via Points?
YES
YES
Interrupt request generation to update current position P_c
Final Position P_{FP} = P_c
YES
Is offset error $|(P_D-P_{FP})|<e$?
Is offset error $|(P_{VP}-P_C)|<e$?
YES
NO
NO
Generate robot motion request to correct offset error
Main Robot Position Measurement Process
BEGIN
Home Robot
Point laser to retroreflector centre
Initialise current position P_C to zero
Provide desired position P_D
YES
Is offset error $|(P_D-P_C)|<e$?
NO
Send P_e to robot control unit and generate robot motion request
Start motion status detection process
Track robot
Update current position P_C

Figure 2: Flowchart of closed-loop control algorithm using LISM technique

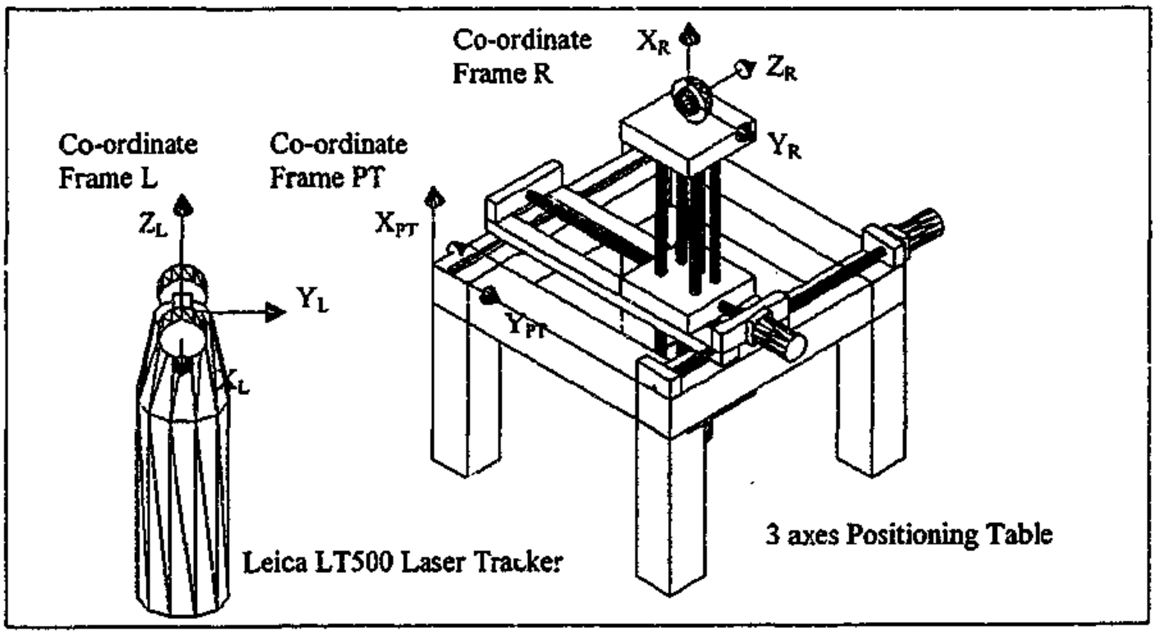
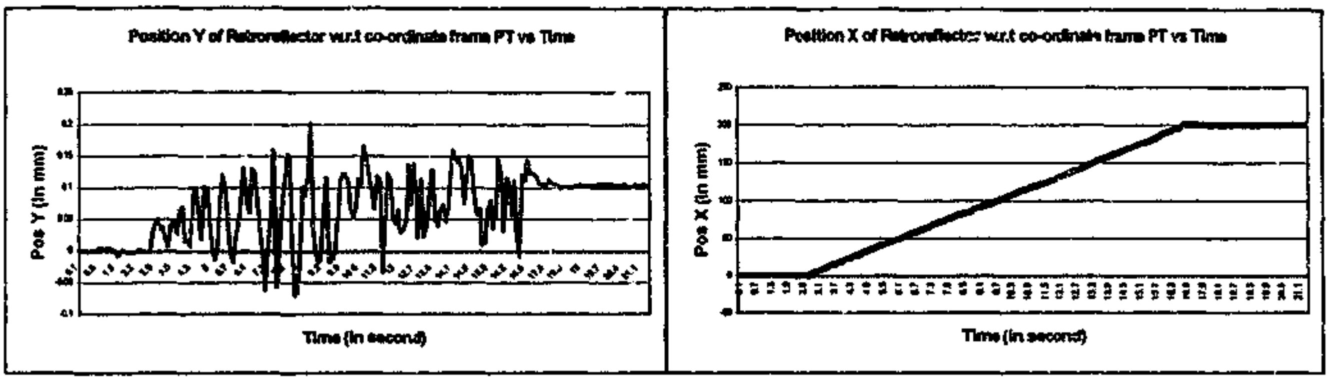


Figure 3: Experimental setup showing the co-ordinate frames

4. EXPERIMENTAL SETUP AND RESULTS

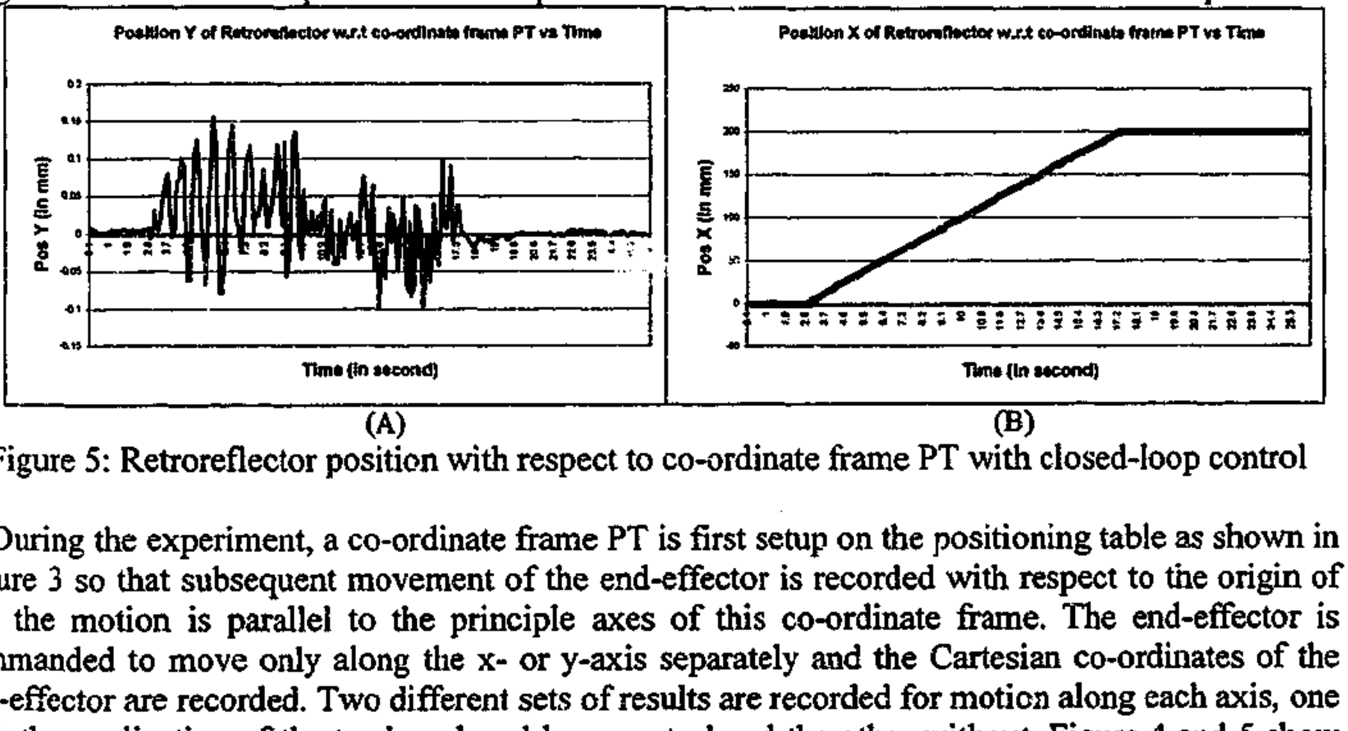
Experiments have been performed to determine the capability of the control algorithm for closed-loop control of robotic devices. The experimental set-up, shown in Figure 3, consists of the mentioned Leica LT500 Laser Tracker and a 3D positioning table that resembles a gantry robot manipulator. Position data in spherical co-ordinates can be sampled into the control program via the parallel interface provided by the Leica controller at a maximum sampling rate of 1000 Hz. The positioning table consists of 3 leadscrew-driven axes driven by servo motors. The resolution of each axis is 0.125μm.

4.1 Experimental Results



(A) (B)

Figure 4: Retroreflector position with respect to co-ordinate frame PT without closed-loop control



(A) (B)

Figure 5: Retroreflector position with respect to co-ordinate frame PT with closed-loop control

During the experiment, a co-ordinate frame PT is first setup on the positioning table as shown in Figure 3 so that subsequent movement of the end-effector is recorded with respect to the origin of and the motion is parallel to the principle axes of this co-ordinate frame. The end-effector is commanded to move only along the x- or y-axis separately and the Cartesian co-ordinates of the end-effector are recorded. Two different sets of results are recorded for motion along each axis, one with the application of the tracker closed-loop control and the other without. Figure 4 and 5 show the results of the motion.

From Figure 4A and 4B, it can be observed that the normal motion of the end-effector along the x-axis produces a considerable amount of fluctuation in the y co-ordinates. This is due to the vibration of the end-effector while in motion. The mean of the fluctuation is increasing with time which represents a deviation of the end-effector away from the desired path. The increase in the mean value are due to twisting and bending of the moving link under the load of the end-effector and this introduced a final steady state error of Δx = 0.01mm and Δy = 0.10mm. With the implementation of closed-loop control algorithm, it can be seen from Figure 5A and 5B that the steady state error along the y-axis has been reduced to 0.01mm while Δx remain unchanged as it is already at the maximum accuracy that can be sampled by the Leica Tracker. The mean value of the fluctuation can be observed to be close to zero, which shows that the path end-effector is within a limit of ±0.15mm off the desired path. The small fluctuations at the beginning of the motion and at steady state are due to the noise in the environment.

5. CONCLUSION AND FUTURE WORK

From the experimental results, it can be concluded that the closed-loop control algorithm using LISM technique can improve positioning accuracy of a robotic device. This is accomplished by compensating the position offset error caused by the deviation of the end-effector from the desired path mainly due to vibration and bending. The final steady state error is close to the predefined error limit that is the maximum accuracy that can be measured by the Leica Tracker.

Future work includes the experimentation involving the third axis. Experiments will also be conducted based on the via points approach with all three axes running at the same time. Moreover, the algorithm will be tested on more complex manipulator such as the long-reach manipulator, which generally has significant amount of bending and vibration due to long link length.

ACKNOWLEDGEMENT

This project is partly funded by Australian Research Council (ARC) and Harold Armstrong Fund.

REFERENCES

[1] John J. Craig, Introduction to Robotics 2nd edition, Addison Wesley, pp. 391. 1991

[2] B. Shirinzadeh, P. L. Teoh, C. W. Foong, Y. D. Liu, "A strategy for accurate guidance of a manipulator using laser interferometry-based sensing technique", Sensor Review, Vol. 19, No. 4, pp. 292-299, 1999.

[3] M. Vincze, J.P. Prenninger, H. Gander, "A laser tracking system to measure position and orientation of robot end effectors under motion", The International Journal of Robotics Research 13(4), pp. 305-314. 1994.

[4] B. Shirinzadeh, "Laser interferometry-based tracking for dynamic measurements", Industrial Robot, Vol. 25. No. 1, pp. 35-41, 1998.

[5] B. Shirinzadeh, P. L. TEOH, "A study of predictive control for laser tracking of robots", Proceedings of Pacific Conference on Manufacturing (PCM98), Brisbane, Australia, pp. 328-333, August 1998.

[6] J.P. Prenninger, H. Gander, M. Vincze, "Contactless position and orientation measurement of robot end-effectors." IEEE Conference on Robotics and Automation. Vol. 1 pp. 180-185, 1993.

[7] P. L. Teoh, B. Shirinzadeh, C. W. Foong, G, Alici, "The measurement.uncertainties in laser interferometry-based sensing and tracking technique", Journal of Measurement, in press.

[8] Leica Geosystems AG, Leica Laser tracker system, Leica Geosystems AG Switzerland, 1999.

P. L. Teoh, B. Shirinzadeh, "3D external ground truth feedback sensing for robot manipulators using laser interferometer-based sensing and measurement technique", Proceedings of the 8th IEEE Conference on Mechatronics and Machine Vision in Practice (M2VIP), Hong Kong, pp. 261-265, August 2001.

8th IEEE Conference on Mechatronics and Machine Vision in Practice
Hong Kong 2001

3D External Ground Truth Feedback Sensing for Robot Manipulators using Laser Interferometer-based Sensing and Measurement Technique

P. L. Teoh and B. Shirinzadeh
Department of Mechanical Engineering
Monash University
Clayton, VIC 3800
Australia
TEL: 613 9905 3510
Email: pek.teoh@eng.monash.edu.au or bijan.shirinzadeh@eng.monash.edu.au

***Abstract*-- Robot manipulators have been used in many manufacturing industries. In many cases, the use of manipulator involves the interaction between the robot's end-effector and the objects in the physical environment. Due to an increase in complexity and higher quality requirements in production operations, a more accurate positioning of robot's end-effector along a desired path is required. However, due to the geometric and non-geometric errors associated with the robot position and orientation (pose) control, the repeatability of today's industrial robots is at least an order of magnitude better than their absolute accuracy. Laser Interferometry-based Sensing and Measurement (LISM) technique has been proposed to track and perform position measurements of dynamic systems such as robot manipulators. These measurements can be used to perform accurate calibration and to develop performance measures. The technique can provide dynamic robot position measurements in real time, has high accuracy, a large working space, a high sampling rate and automatic target tracking. LISM technique can be further modified to provide a closed-loop control of the robot manipulator to improve accuracy. In this paper, an external ground truth feedback sensing control methodology using the LISM technique will be presented. . This is a control strategy where an external position-sensing unit is used to perform dynamic measurements of the robot manipulator during robot manipulation to perform compensation or error adjustment on-line. The physical make-up, measurement and analysis techniques, together with the control algorithm for the above control methodology will be described.**

***Keywords*— Laser Interferometry-based Sensing and Measurement technique (LISM), external ground truth feedback sensing, closed loop control, cartesian-based position measuremen, position sensing unit**

I. Introduction

Manipulators consist of rigid links that are connected to each other with joints (revolute or prismatic). Actuators are attached to these joints to provide relative motion of neighbouring links by changing the joint angles for revolute joint and joint offset for prismatic joint. These motions move the end-effector, which is the free end of the manipulator, to a desired location in space. Sensors such as encoders are usually instrumented to each joint to allow the actual joint motion to be measured [1]. However, in normal robotic operation, the desired locations in space are usually specified in Cartesian co-ordinates (i.e. x, y, and z). This leads to the use of kinematic models in the robotic control system to convert the user-input Cartesian units into a set of joint angles. The kinematic model uses the joint positions determined from the axes encoders to provide the three dimensional Cartesian co-ordinates of the manipulator. The accuracy of such a model relies heavily on the accuracy of the parameters used such as link lengths. The accuracy is also affected by the geometric (e.g. bending) and non-geometric (e.g. temperature) errors. Further, joint backlash can also play an important role in such errors. Therefore, it is well known that, in general, the absolute accuracy of today's robots is at least a magnitude lower than its repeatability [2]. Accurate robot position measurements and calibration techniques are often required to improve the robot accuracy.

Laser Interferometry-based Sensing and Measurement (LISM) technique has been proposed for this purpose. This technique can provide dynamic robot position measurement in real time and has high accuracy, a large working space, a high sampling rate and automatic target tracking [3, 4]. However, only geometric errors can be calibrated, as these are the known error that can be modeled into the kinematic model. Non-geometric errors such as joint backlash, tumbling, and effect of environmental conditions on material properties and laser are not known or difficult to model. High accuracy external sensing unit is required to provide a closed-loop control. LISM technique can be modified to provide a closed-loop control of the robot manipulator to improve accuracy

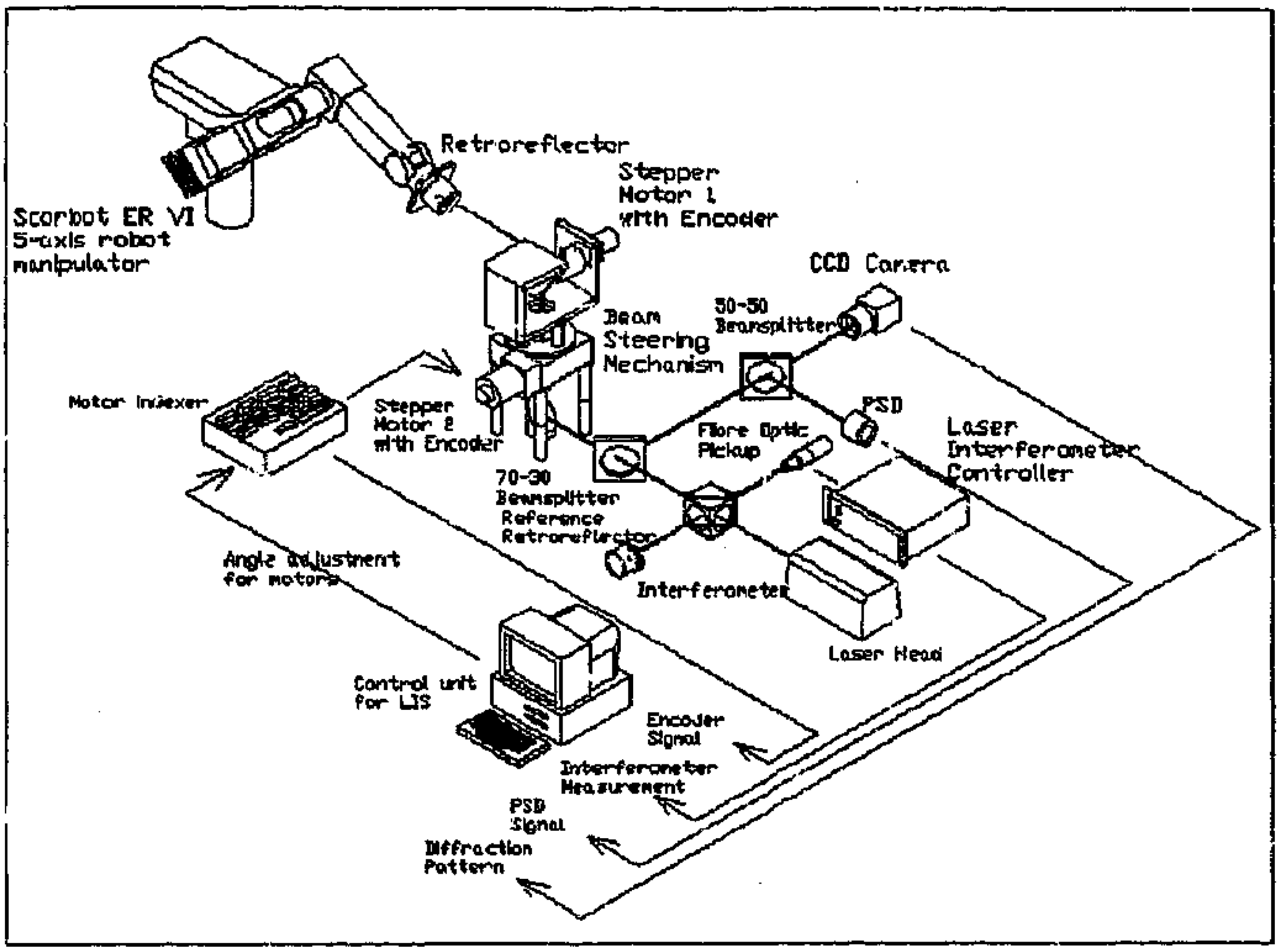


Figure 1: Functional layout of LISM Technique

[2, 5]. This is a control methodology where an external position-sensing unit is used to perform dynamic measurements of the robot manipulator during robot manipulation. Feedback from the measuring unit will provide true Cartesian-based position measurement for robot control. When there is a discrepancy between the desired and the actual position detected by the LISM apparatus, a signal will be sent to the robot controller to move the robot manipulator by the calculated amount. This will ensure high accuracy robot positioning in complex manipulator operation.

In this paper, an external ground truth feedback sensing control methodology using LISM technique will be proposed. This approach will provide accurate control of the robotic devices without modification to the mechanical unit or its control architecture.

II. LASER INTERFEROMETRY-BASED SENSING AND MEASUREMENT PRINCIPLE

Laser Interferometry-based Sensing and Measurement (LISM) technique generally involves the dynamic acquisition of the three dimensional position of an end-effector in its workspace [1, 4, 6]. It can also be used to measure the orientation of robot end-effector [7, 8, 9, 10]. The LISM technique uses the angular and distance data, obtained from the beam steering mechanism and the interferometer, respectively, to provide the position of the target retroreflector attached to the end-effector of the manipulator. It maintains tracking of the target by sensing the offset of the incident and reflected beam. The tracking is carried out by adjusting the angles of a beam steering mechanism. A LISM apparatus was developed for this study. A functional layout of the overall design of the LISM apparatus is provided in Figure 1.

In this layout, the laser beam generated by the HeNe laser head travels to the interferometer where it is split into a reference beam and a measurement beam. The reference beam is directed to the measurement board in the laser interferometer controller via the fibre optic pickup. This beam will later be compared with the returning measurement beam, whose frequency will be Doppler-shifted, to determine the distance between the target (i.e. a retroreflector mounted on the robot end-effector) and the laser head. The measurement beam travels through a 70-30 percent beam splitter before entering the two-axis beam steering mechanism. The beam steering mechanism consists of three 45° beam-steer mirrors, with one being stationary and the other two driven by two stepper motors – one rotate about the vertical axis and the other about the horizontal axis.

Two high-resolution optical encoders are attached to the stepper motors to provide the angular displacement of the stepper motors and thus the rotational angles of the beam-steer mirrors. The beam is directed to the target retroreflector by rotating the stepper motors attached to the appropriate beam-steer mirrors. Once the beam hits the target retroreflector, the beam is reflected through 180° and it travels back parallel with the incident beam. There will exist an offset between the incident and the reflected measurement beam if the beam does not hit the centre of the retroreflector.

The reflected beam then travels through the beam-steer mirrors in the beam steering mechanism and back to the 70-30 percent beam splitter. 30 percent of the beam power is diverted through a 50-50 percent beam splitter, where it is split equally and directed to a Charged Coupled Device (CCD) camera and a Position Sensitive Diode (PSD). The CCD camera, which is connected to a high-data-acquisition-rate frame grabber board, captures the diffraction pattern of the retroreflector, so that analysis can be performed to determine the orientation of the target retroreflector. The PSD is attached to a data acquisition card and it detects the offset of the beam from the centre of the PSD sensor. The reflected beam position from the centre of the PSD will be acquired by the card and the offset, referred to as the tracking error will be stored in the LISM control unit. The remaining 70 percent of the beam power will be combined with the reference beam via the interferometer and the fibre optic pickup. The Doppler shift of the reflected beam can be detected and used by the processing electronics within the laser interferometer controller to determine the displacement of the beam and the velocity.

The LISM control system minimises the tracking error obtained from the PSD acquisition system by signaling the motor controller which in turn rotates the axes of the beam steering mechanism by a corrective angles calculated using the tracking error, thus following the arbitrary movements of the target. Measurement of the position of the target in space is obtained from the interferometer measurement, tracking errors, angular displacements of the axes of the beam steering mechanism, and the kinematics of the LISM apparatus. The tracking algorithm utilises a predictive control algorithm that allows estimation of future position of the target from the previous position, velocity and acceleration values [6].

III. EXTERNAL GROUND TRUTH FEEDBACK SENSING

External ground truth feedback sensing is proposed to ensure high accuracy robot positioning. It is a control methodology where the LISM apparatus is used as a position sensor to measure the position of the end-effector in real time. The LISM apparatus will maintain tracking of the robot end-effector until it reaches the final position calculated by the robot controller. However, there may be discrepancy between this final position commanded by the robot controller and the desired position. The LISM apparatus will then acts as a processing unit to determine the offset error and a signal will be sent to the robot controller to move the robot by the calculated amount. This will improve the positioning accuracy of a robot manipulator. A flow chart of the control algorithm is provided in Figure 2.

A. Control Algorithm

In this methodology, the robot manipulator will first be commanded to the home position. The laser will be pointed to the center of the retroreflector attached to the end-effector of the manipulator by manually rotating the motors on the beam steering mechanism. When the laser is in position, the current robot position measured by the LISM control unit will be initialised to be the zero reference. The LISM control unit will then be provided the desired position P_D for the manipulator. The current position P_C of the robot manipulator will be compared with the desired position to determine the offset error (P_D-P_C). When the offset error is more than a predefined error-limit, the LISM control unit will send the offset error to the robot controller. The robot controller will calculate the required rotations of each actuator. Motion of these motors will be initiated to position the end-effector to the desired location.

When the robot has moved, there will be tracking error in the reflected beam from the centre of the retroreflector. This tracking error will be detected by the PSD acquisition sub-system. The LISM control unit will maintain tracking of the robot end-effector by rotating the beam steering mechanism to correct the tracking error. The current position is also updated through the measurements from the PSD acquisition sub-system, angular displacements of the axes of the beam steering mechanism, the interferometer measurements and the kinematics of LISM. A new offset error is calculated from the updated P_C and the process is repeated until user intervention or interrupts from the two other concurrent processes that will be described in the following paragraphs.

Two other processes are running concurrently during the tracking of the robot manipulator. There are the motion detection and motion direction detection processes. In the motion detection process, the LISM controller will continuously check for the motion status of the robot manipulator. When the robot has stopped as the encoders attached to each robot actuator recorded the target joint angles, an interrupt will be sent to the LISM control unit to stop the tracking process. The current position P_C will be updated and the offset error recalculated. When the offset error, which now represents the discrepancy between the robot final position P_{RF} and the desired position P_D is greater than a predefined error-limit, the LISM control

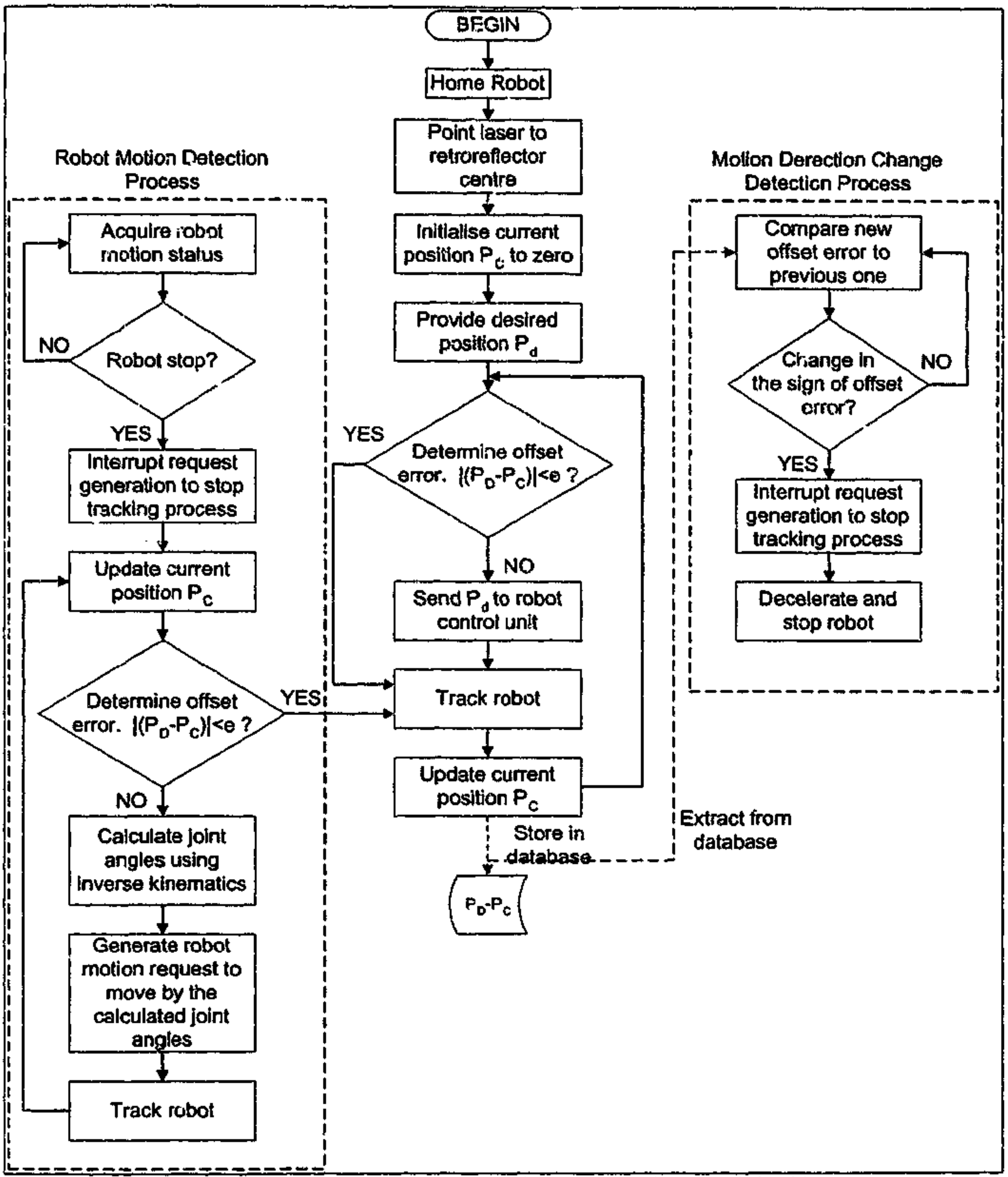

BEGIN
Home Robot
Point laser to retroreflector centre
Robot Motion Detection Process
Motion Derection Change Detection Process
Acquire robot motion status
Robot stop?
NO
YES
Interrupt request generation to stop tracking process
Update current position P_C
Determine offset error. $|(P_D-P_C)|<e$?
YES
NO
Calculate joint angles using inverse kinematics
Generate robot motion request to move by the calculated joint angles
Track robot
Initialise current position P_c to zero
Provide desired position P_d
YES
Determine offset error. $|(P_D-P_C)|<e$?
NO
Send P_d to robot control unit
Track robot
Update current position P_C
Store in database
P_D-P_C
Extract from database
Compare new offset error to previous one
Change in the sign of offset error?
NO
YES
Interrupt request generation to stop tracking process
Decelerate and stop robot

Figure 2: Flowchart of external ground truth feedback sensing control algorithm

unit will calculate the required rotations of each actuator using the robot kinematics. These calculated angles will then be sent to the robot controller to further correct the offset error. The new offset error is again updated and the process is repeated until the robot stops at the position where the offset error is less that the predefined error-limit. The tracking process will be resumed when this occurs. This process is to eliminate error due to the inaccuracy of the robot controller in positioning the end-effector of the robot.

The motion direction detection process is to prevent the overshooting of the robot manipulator. Overshooting can occur due to several factors such as inherent oscillation of long reach manipulator. It can also occur in high load condition due to high inertia. This process is aimed to damp the oscillation and reduce settling time of the robot end-effector. This can be done by comparing the motion direction of the current offset error to the previous one from the database. The robot has overshot if there is a change in motion direction represented by the change of sign in the offset error. When this occurs, another interrupt will be sent to the LISM control unit to decelerate the robot to a halt immediately. The motion detection process will then be activated.

The tracking process will maintain tracking of the robot manipulator even when the robot has stop at position within the predefined error-limit. This is to account for subsequent environmental effect on the robot that will move the end-effector away from the desired position.

IV. CONCLUSION AND FUTURE WORK

In this paper, the principle of external ground truth feedback sensing of dynamic system using LISM technique has been presented. The control algorithm of such approach was described. The above algorithm has to be performed in real time and thus, the speed of the LISM apparatus in tracking of the robot end-effector and updating the position of the robot is the most important factor. The LISM apparatus presented in this paper is now capable of position measurements at speeds of about 0.5m/s with accuracy of better then ±0.005mm.

The proposed closed-loop control using LISM technique is aimed to provide for improvement of accuracy in the control of robot manipulators and other dynamic systems. The positioning of the end-effector of robotic systems such as long reach manipulator, where their accuracy is affected by vibration and bending can be benefit from this approach. Further, this control methodology can be used as a basis for the development of laser interferometry-based guidance technique. In this approach, dynamic measurements are obtained and the robot is guided to position the target retroreflector precisely to a desired location or along a desired path based on these measurements, thus improving accuracy and repeatability.

However, there are limitations on the proposed methodology such as small working range due to the small incident angle range of target retroreflector. Only laser beams that are incident at an angle of ±30° with respect to the vertical plane of the retroreflector will be reflected. Second limitations are the incapability of performing orientation sensing. Thus, orientation offset error compensation cannot be performed.

Future work includes the incorporation of orientation sensing into the control algorithm to provide 6D position and orientation (pose) offset determination and compensation. A cat-eye retroreflector with an incident angle range of ±65° will also be used to increase working range. An upgrade of the hardware such as data acquisition card and the motor controller card will also being considered to improve the tracking speed. Further, experimentation of the proposed control methodology on a long reach manipulator will be established.

V. ACKNOWLEDGEMENT

This project is partly funded by Australian Research Council (ARC) and Harold Armstrong Fund.

VI. REFERENCES

[1] John J. Craig, Introduction to Robotics 2nd edition, Addison Wesley, pp. 391. 1991

[2] B. Shirinzadeh, P. L. Teoh, C. W. Foong, Y. D. Liu, "A strategy for accurate guidance of a manipulator using laser interferometry-based sensing technique", Sensor Review, Vol. 19, No. 4, pp. 292-299, 1999.

[3] M. Vincze, J.P. Prenninger, H. Gander, "A laser tracking system to measure position and orientation of robot end effectors under motion.", The International Journal of Robotics Research 13(4), pp. 305-314. 1994.

[4] B. Shirinzadeh, "Laser interferometry-based tracking for dynamic measurements", Industrial Robot, Vol. 25. No. 1, pp. 35-41, 1998.

[5] B. Shirinzadeh, H. C. Chong, K. C. Lee, P. L. Teoh, "Issues and Techniques for Interferometry-Based Laser Guidance of a Manipulator" The Fifth International Conference on Control, Automation, Robotics and Vision, Singapore, Vol. 1, pp. 271-275, 1998.

[6] B. Shirinzadeh, P. L. TEOH, "A study of predictive control for laser tracking of robots", Proceedings of Pacific Conference on Manufacturing (PCM98), Brisbane, Australia, pp. 328-333, August 1998.

[7] J.P. Prenninger, H. Gander, M. Vincze, "Contactless position and orientation measurement of robot end-effectors." IEEE Conference on Robotics and Automation. Vol. 1 pp. 180-185, 1993.

[8] B. Shirinzadeh, P. L. Teoh, C. W. Foong, "Orientation measurement using vision and non-vision based techniques in laser tracking system", 30th International Symposium on Robotics, Tokyo, Japan, pp. 317-324, 1999.

[9] Y. Bao, N. Fujiwara, "Dynamic measurement orientation by LTS." Proceedings of the Japan/USA Symposium on Flexible Automation. Vol. 1, pp. 545-548, 1996.

[10] B. Shirinzadeh, P. L. Teoh, C. W. Foong, Y. D. Liu "Orientation measurement technique in laser interferometry based tracking system for robot manipulator calibration", Proceedings of Pacific Conference on Manufacturing (PCM2000), Detroit, USA, Vol. 2, pp. 788-793, September 2000.

B. Shirinzadeh, P. L. Teoh, C. W. Foong, Y. D. Liu, "A strategy for accurate guidance of a manipulator using laser interferometry-based sensing technique", Sensor Review, Vol. 19 No. 4, pp. 292-299, 1999.

Research articles

A strategy for accurate guidance of a manipulator using laser interferometry-based sensing technique

Bijan Shirinzadeh
Pek Loo Teoh
Chee Wei Foong
YongDong Liu

The authors

Bijan Shirinzadeh, Pek Loo Teoh and **Chee Wei Foong** are in the Department of Mechanical Engineering, Monash University, Clayton, Australia.
YongDong Liu is at the State Key Laboratory of Precision Measurement Technology and Instruments, TsingHua University, Beijing, China.

Keywords

Lasers, Robots, Position measurement

Abstract

Laser interferometry-based sensing (LIS) technique has been proposed and established recently to track and perform dynamic measurements on a moving end-effector of a robot manipulator. In this paper, a technique using LIS system to perform guidance of a manipulator is proposed. The LIS system is used as a sensor to guide the end-effector of a robot manipulator. This is to be accomplished through the implementation of guidance error determination and compensation, and path generation in the control algorithm. This technique can be used to accurately guide the manipulator's end-effector to a specified location or along a specified path with a high level of accuracy. The structure and various components within the system and the control strategy are also presented.

Electronic access

The research register for this journal is available at **http://www2.mcb.co.uk/mcbrr/sr.asp**

The current issue and full text archive of this journal is available at **http://www.emerald-library.com**

Sensor Review
Volume 19 · Number 4 · 1999 · pp. 292–299
© MCB University Press · ISSN 0260-2288

Introduction

The increase in the number of complex applications of robots in manufacturing and service industries has emphasised the need for higher levels of positioning accuracy from robot manipulators. It is well-known that the repeatability of today's industrial robot is at least an order of magnitude better than its absolute accuracy (Vincze *et al.*, 1994). This is due to the fact that the position and orientation (i.e. pose) of a robot manipulator are described using a kinematic model. This kinematic model uses the joint positions determined from the axes encoders to provide the three-dimensional co-ordinates of the manipulator. The accuracy of such a model relies heavily on the accuracy of the parameters used (e.g. link lengths). The accuracy is also affected by the geometric (e.g. bending) and non-geometric (e.g. temperature) errors. Further, joint backlash can also play an important role in such errors. Therefore, accurate robot pose measurement and calibration techniques are required to improve the robot accuracy.

Recently, a single beam laser interferometry-based sensing (LIS) technique, capable of performing dynamic measurements of robot end effector's pose, has been proposed and established for such purposes (Parker and Gilby, 1982/3; Gander *et al.*, 1993). This technique provides high accuracy, a large working space, a high sampling rate, and automatic target sensing capability in real time (Spiess *et al.*, 1996). In this paper, a technique using the LIS as the pose sensor to accurately guide the manipulator's end-effector to a specified point or along a specified path is proposed. This approach is expected to provide accurate laser-based remote control of fixed based robots, mobile robots, and long reach manipulators. The physical make-up, measurement and analysis techniques, together with the control algorithm for the above technique will be described. Preliminary results of such an approach will also be presented.

This research project was partly funded by grants from the Australian Research Council (ARC) and the Harold Armstrong Research Fund. The authors wish to thank H.C. Chong and K.C. Lee for their work on the research project.

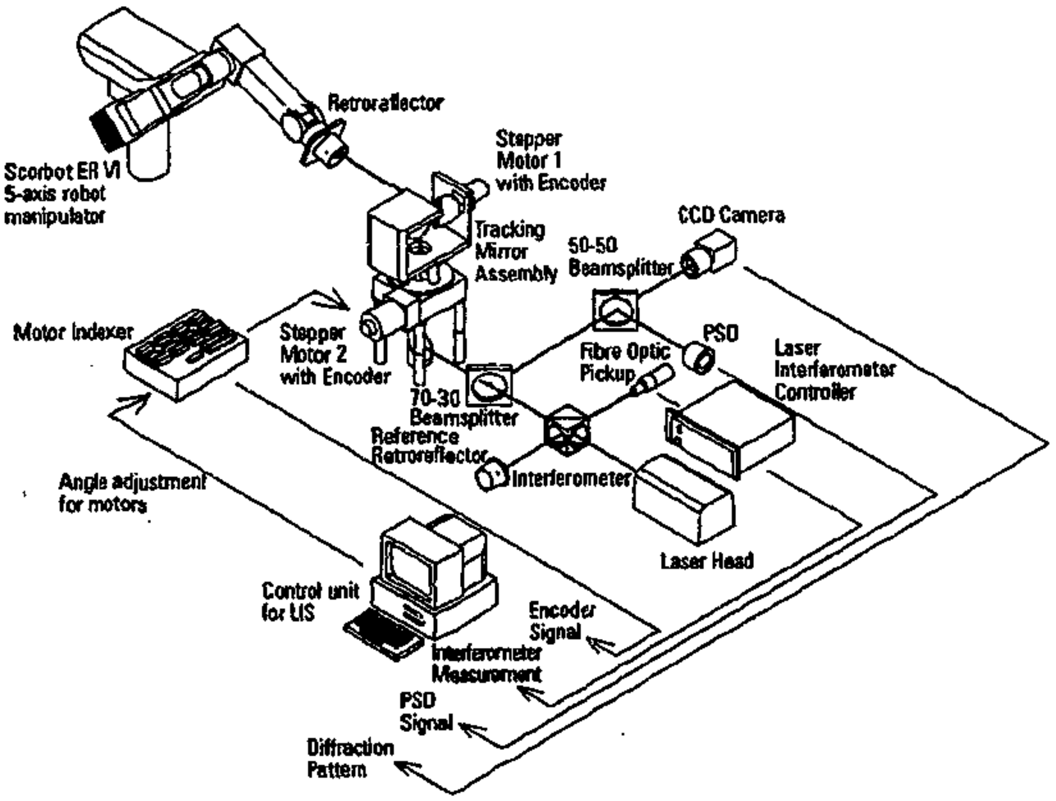
Laser interferometry-based sensing principle

Laser interferometry-based sensing (LIS) generally involves the dynamic acquisition of the 3D pose of an end-effector in its workspace. The LIS system uses the angular and distance data, obtained from the beam steering mechanism and the interferometer, respectively, to provide the pose of the target retroreflector attached to the end-effector of the manipulator. It maintains tracking of the target by sensing the offset of the incident and reflected beam. It subsequently performs offset corrections by adjusting the angles of the beam steering mechanism. A LIS system was developed for this study. A functional layout of the overall design of the LIS is provided in Figure 1.

In this layout, the laser beam generated by the HeNe laser travels to the interferometer where it is split into a reference beam and a measurement beam. The reference beam is directed to the measurement board in the laser interferometer controller via the fibre optic pickup. This beam will later be compared with the returning measurement beam, whose frequency will be Doppler-shifted, to determine the distance between the target (i.e. a retroreflector mounted on the robot end-effector) and the laser head. The measurement beam travels through a 70-30 per cent beam splitter and is directed to the target by the beam steering mechanism. Once the beam hits the target, the beam is reflected through 180° and it travels back parallel with the incident beam. There will exist an offset between the incident and the reflected measurement beam if the beam does not hit the centre of the retroreflector.

The reflected beam then travels through the mirror assembly and back to the 70-30 per cent beam splitter. Of the beam power, 30 per cent is diverted through a 50-50 per cent beam splitter, where it is split equally and directed to a charged coupled device (CCD) camera and a position sensitive diode (PSD). The CCD camera, which is connected to a high-data-acquisition-rate frame grabber board, captures the diffraction pattern of the retroreflector, so that analysis can be performed to determine the orientation of the retroreflector. The PSD detects the offset of the beam from the centre of the PSD sensor. This offset error is referred to as the tracking error. The remaining 70 per cent of the beam power will be combined with the reference beam via the interferometer and the fibre optic pickup. The Doppler shift of the reflected beam can be detected and used by

Figure 1 Set-up of the laser interferometry-based system



the processing electronics within the laser interferometer controller to determine the distance travelled and the velocity.

The LIS control system minimises the tracking error obtained from the PSD acquisition system by signalling the motor controller which in turn rotates the axes of the beam steering mechanism, thus following the arbitrary movements of the target. Measurement of the position of the target in space is obtained from the interferometer measurement, tracking errors, angular displacements of the axes of the beam steering mechanism, and the kinematics of LIS. The tracking algorithm utilises a predictive control algorithm that allows estimation of future position of the target from the previous position, velocity and acceleration values (Shirinzadeh and Teoh, 1998).

Laser interferometry-based guidance technique

Laser interferometry-based guidance (LIG) is the technique of positioning the end-effector of a robot manipulator accurately to a desired point in Cartesian space along a predefined trajectory by steering the laser beam. In this technique, the LIS system is used as a pose sensor to measure the dynamic pose of the end-effector. It also acts as a processing unit for path generation and guidance error compensation in the control algorithm. When the beam steering mechanism has directed the laser beam to a new desired pose, the offset will be detected by the LIS system. It subsequently performs offset corrections by adjusting the joint angles of the robot manipulator.

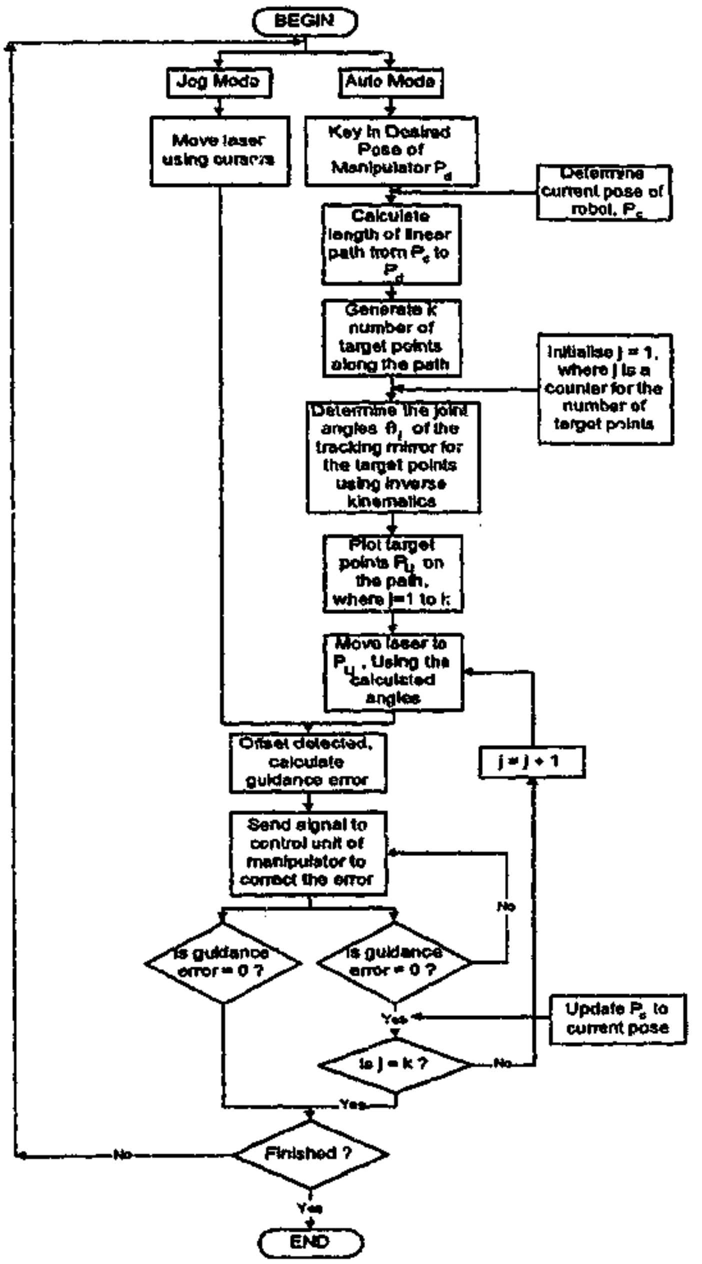
Control algorithm

A flow chart of the control algorithm is provided in Figure 2. The control software is designed to have two modes of operations:

(1) a jog mode; and
(2) an auto mode.

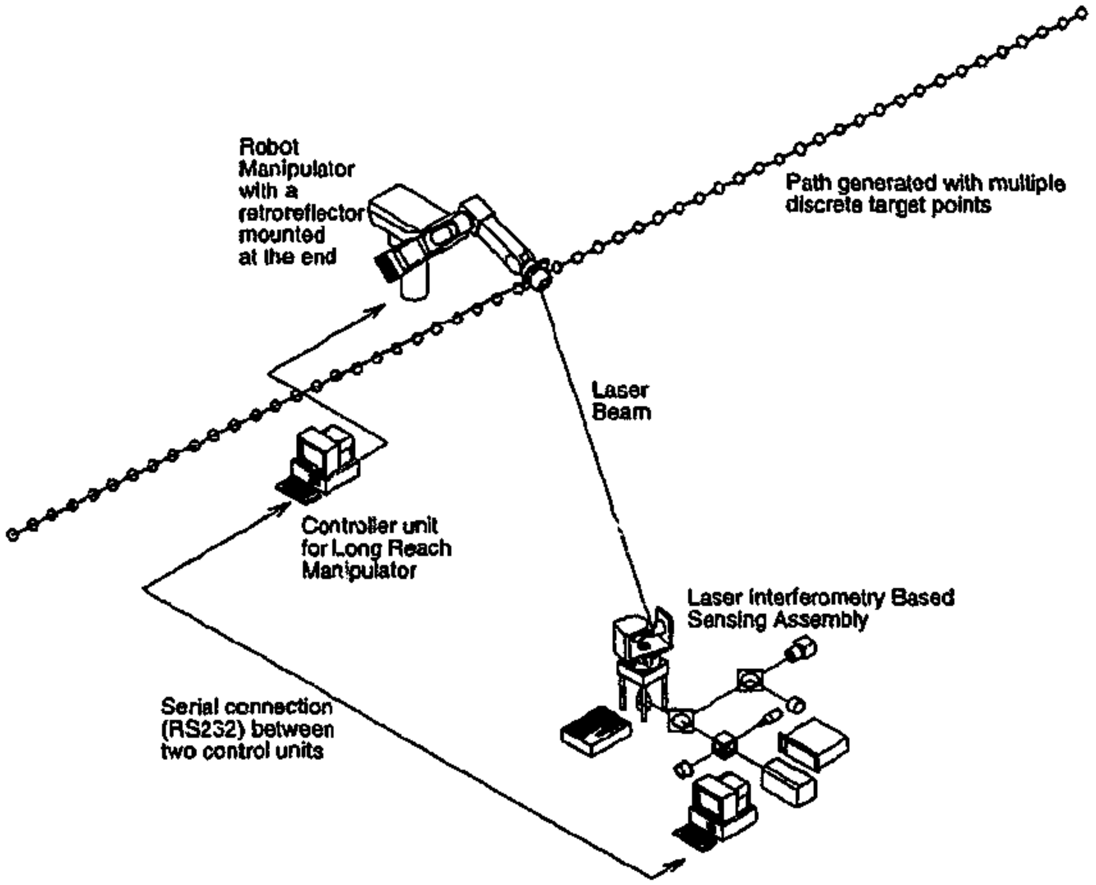
In the jog mode, the operator will manually control the rotation of the axes of the beam steering mechanism using the keyboard or an input device (e.g. joystick). The laser beam direction will be displaced by a predefined distance and thus generate the guidance error. The robot controller will be automatically commanded to move in the required direction to reduce this error (this will be described in detail later).

Figure 2 Flowchart of the control software


BEGIN
Jog Mode
Auto Mode
Move laser using cursors
Key in Desired Pose of Manipulator P_d
Determine current pose of robot, P_c
Calculate length of linear path from P_c to P_d
Generate k number of target points along the path
Initialise j = 1, where j is a counter for the number of target points
Determine the joint angles θ_j of the tracking mirror for the target points using inverse kinematics
Plot target points P_{tj} on the path, where j=1 to k
Move laser to P_{tj}, Using the calculated angles
Offset detected, calculate guidance error
j = j + 1
Send signal to control unit of manipulator to correct the error
Is guidance error = 0 ?
Is guidance error = 0 ?
No
Yes
Update P_c to current pose
Is j = k ?
No
Yes
Finished ?
No
Yes
END

In the auto mode, the system will first prompt the operator to enter the desired pose for the manipulator. The control software will determine the total offset of the desired pose with respect to the current pose. An appropriate path with multiple target points, which is to be followed by the laser beam, is generated. This will be discussed further in the path generation section. The control software will command the motors to direct the beam to the required pose, increment by increment, following the target points along the path. There will be a guidance error when the laser beam has moved at each increment. This methodology can be seen in Figure 3.

Figure 3 Robot following a path


Robot Manipulator with a retroreflector mounted at the end
Path generated with multiple discrete target points
Laser Beam
Controller unit for Long Reach Manipulator
Laser Interferometry Based Sensing Assembly
Serial connection (RS232) between two control units

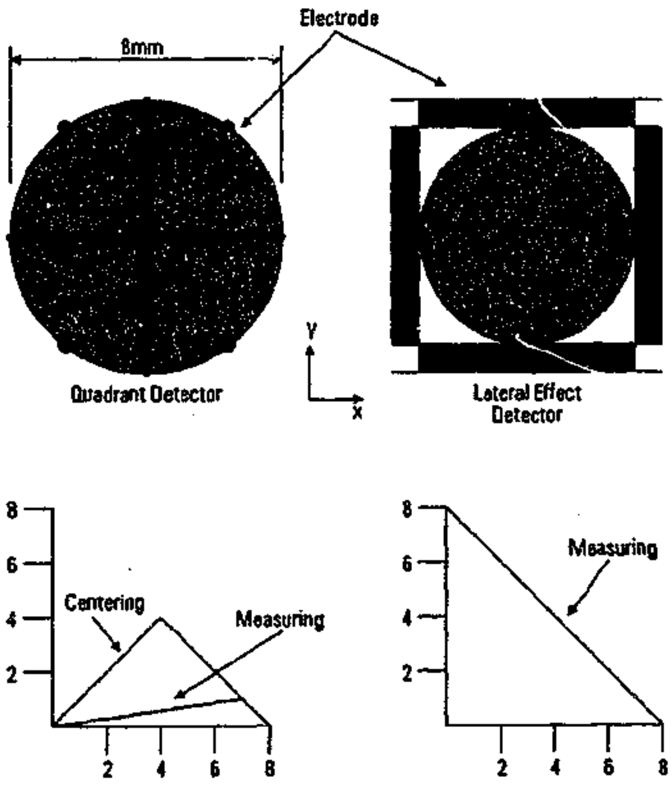
Guidance error determination and compensation

Two different sub-systems are used to determine the guidance error of the manipulator. This section describes these sub-systems and the method of compensation.

Position-sensitive diode (PSD) acquisition sub-system

To detect the guidance error in the plane perpendicular to the laser beam, the PSD acquisition sub-system in the LIS system (Shirinzadeh, 1998) is the focus of attention. There are two main types of PSDs, the quadrant detector and the lateral effect detector. The detector used in this study is the Melles Griot 13PSL001 Lateral Effect Detector System (Melles Griot, 1999). It has four electrodes connected equidistant around its perimeter, as shown in Figure 4. The electrodes are connected such that opposite pairs yield photo currents that are processed to give the displacement of the beam in the *x* and *y* directions. It can accurately measure beam position with a position resolution of

Figure 4 Quadrant and lateral effect detector



± 1μm across its entire surface due to software controlled calibration algorithm. This algorithm linearises the detector's photo currents; thus it is possible to accurately determine the magnitude of the offset error of a beam. The initial data update rate was 10Hz. However, that has been modified to the current update rate of 18Hz.

When the laser beam has moved by one increment, there will be an offset error in the reflected beam. This offset error will be detected by the PSD acquisition sub-system, and the guidance error of the laser beam from the centre of the PSD can be determined. The controller will minimise the guidance error by signalling the control unit of the manipulator, allowing the manipulator control unit to calculate appropriate joint angle rotation using inverse kinematics. It will then position the end-effector to the target point. Dynamic measurements are still acquired to update the PSD readings. A new offset will be detected when the robot has moved and the process is repeated until the guidance error detected is within the acceptable zero-limit range.

Laser interferometer sub-system

To determine the guidance error of the end-effector in the plane parallel to the direction of the laser beam, the laser interferometer (Shirinzadeh, 1998) is the focus of attention. The laser interferometer currently being used is the Zygo ZMI1000 high-velocity interferometer system. This laser interferometer is based on a He-Ne heterodyne (dual frequency) laser with a beam diameter of 6mm. It employs the Zeeman split to generate two frequency components separated by 20MHz. Each component has an opposite circular polarisation that allows each frequency to be separated by optical polarisers. It relies on Doppler shifts caused by the movement of the target retroreflector to generate interference fringes. The structure of the sub-system is provided in Figure 5. The laser beam consists of two separated frequencies, F_1 and F_2, which are polarised at right angles to one another. As the beam passes through the interferometer, F_2 is directed to the reference retroreflector while F_1 passes through the polarising beam splitter to the target retroreflector. As the target retroreflector moves, the frequency F_1 will be shifted based on the Doppler principle. F_1 will increase or decrease by ΔF_1 as the target retroreflector moves towards or away from the interferometer respectively. F_1 and F_2 are recombined in the laser interferometer to give the measurement signal of:

$$F_2 - (F_1 \pm \Delta F_1) \qquad (1)$$

A reference signal of $F_2 - F_1$ is created by the laser head and combined with the measurement signal, leaving only ΔF_1, the rate of change of position of the target retroreflector. This can then be integrated to yield the relative motion of the target retroreflector. The HeNe heterodyne laser interferometer has a resolution of 0.16μm.

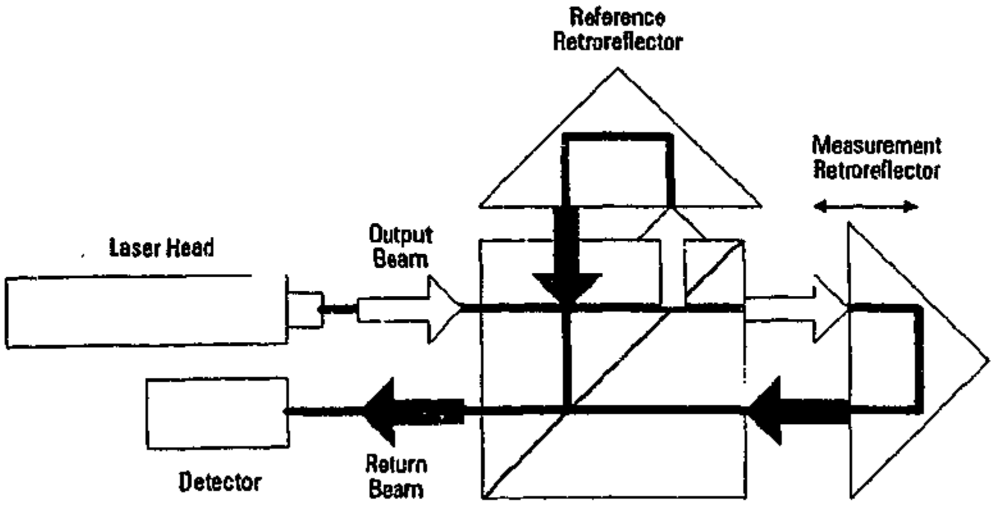
Using the above sub-system, the offset of the next target point from the current one is calculated. The offset is sent to the control unit of the manipulator to calculate the required joint angle rotation using inverse kinematics to position the retroreflector at the desired point. Dynamic measurements are acquired at the same time to update the Doppler shift readings. When the robot has moved, the new offset is detected and the process is repeated until the desired point is reached.

Path generation

In order to guide the retroreflector to the desired pose in the auto mode described above, the path of the guiding laser has to be first determined. Linear or non-linear path can be used. For the purpose of this study a linear path is used. Further, it is also assumed that there are no obstacles within the working space of the robot.

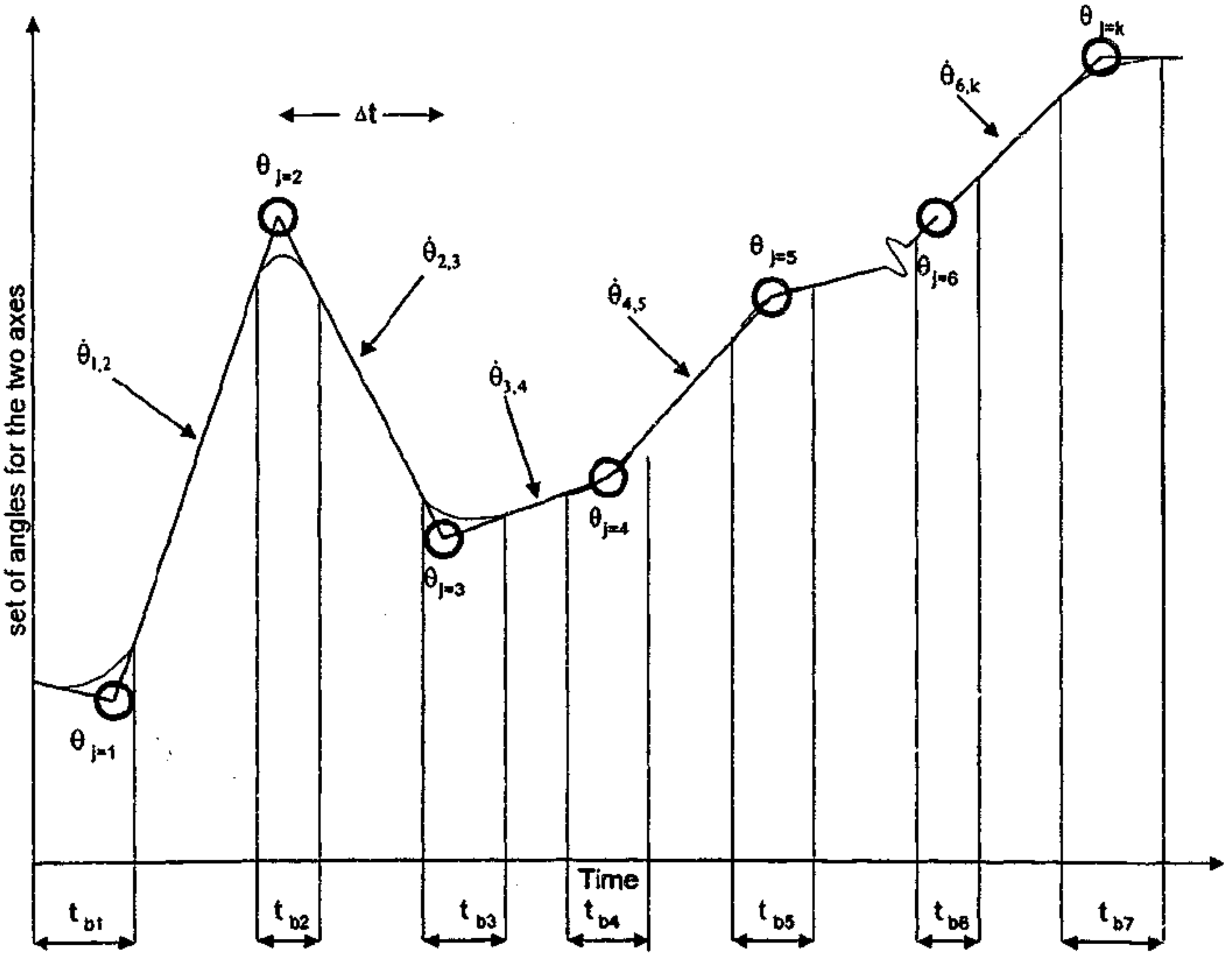
When a desired pose is selected through the computer software, a linear path between the desired pose, P_d $(x_d, y_d, z_d, \varphi_d, \theta_d, \phi_d)$, and the current pose, P_c $(x_c, y_c, z_c, \varphi_c, \theta_c, \phi_c)$, is generated. A discrete number of target points, that are to be followed by the laser beam, are generated and plotted along the path. The distance between any two adjacent target points is defined to be less than the PSD's range. This is to ensure that the laser beam will not be out of the PSD's range to maintain dynamic measurement of the retroreflector. The pose of each target point in polar configuration for the beam steering mechanism can be determined using inverse kinematics. A plot of the angles for both axes of the beam steering mechanism, θ_i and ϕ_i (where i is the target point number), for every target point can then be constructed against time. Figure 6 shows a sample plot for one of the axes.

Figure 5 Structure of laser interferometer



Another important issue that must be considered is the sudden changes in direction and angular velocity. These will in time damage the motors and affect the accuracy. Therefore, a parabolic blend between changes in velocity can be implemented. The time step, Δt, between two joint angles (the desired and the current joint angles) can be determined from the desired average velocity of the manipulator and the distance between the

Figure 6 Plot of joint angle versus time



adjacent target points. The blend time, t_{bi} can be calculated with the known values of θ_i and ϕ_i, and the desired values of acceleration at the target point angles, $\ddot{\theta}_i$ and $\ddot{\phi}_i$. The equations are as follows (Craig, 1991):

$$\dot{\theta}_{i,\, i+1} = \frac{\theta_{i+1} - \theta_i}{\Delta t} \qquad (2)$$

$$t_{bi} = \frac{\dot{\theta}_{i,\, i+1} - \dot{\theta}_{i-1,\, i}}{\ddot{\theta}_i} \qquad (3)$$

The first and the last segments are handled differently since the blend time must be included in the time step. The first and the last blend times t_{bm}, where m =1 or m = n = last target point, can be calculated by equating the velocities in the linear phase segment. The initial and final angular velocities can be easily calculated using the blend times. The following relationships are utilised (Craig, 1991):

$$\ddot{\theta}_m = \frac{(\theta_m - \theta_{m-1})/(\Delta t - \frac{1}{2} t_{bm})}{t_{bm}} \qquad (4)$$

$$t_m = \Delta t - \sqrt{\Delta t^2 - \frac{2(\theta_m - \theta_{m-1})}{\ddot{\theta}_m}} \qquad (5)$$

$$\dot{\theta}_{m-1,\, m} = \frac{\theta_m - \theta_{m-1}}{\Delta t - \frac{1}{2} t_{bm}} \qquad (6)$$

Preliminary experiment

Preliminary experiments have been performed to determine the capability of the apparatus for LIG. The experimental set-up consists of the mentioned LIS system and a Scorbot ER VI robot manipulator. The properties of interest are the response time of the LIS system and the response time of the communication between the LIS system and the robot manipulator. This leads to the following equation that represents the total response time for the LIG technique:

$$T_{response} = T_{robot} + T_{LIS} \qquad (7)$$

where T_{robot} is the response time of the robot to move and acknowledge the command given by the LIS system, and T_{LIS} is the response time of the LIS system to perform dynamic measurements as well as controlling the axes of the beam steering mechanism.

In the current set-up, the response time of the LIS system is found to be in the order of 0.4 seconds (Shirinzadeh *et al.*, 1998). The accuracy of pose sensing is ± 5μm. It must be emphasised that the laser interferometer sub-system provides for distance measurement with an accuracy of 0.16μm and the PSD acquisition sub-system provides for offset error measurement in the order of 1μm. The beam steering mechanism makes use of three 45° beam steers to direct the beam towards the target retroreflector. Both axes of the steering mechanism currently have a maximum rotational velocity around 1 rev./second. The vertical axis has a resolution of around 0.02°, using a motor step rate of 400 steps/rev. and a precision rotary table with a 45:1 gear ratio – i.e. one revolution of the vertical axis requires 18,000 steps. Further, the resolution can be improved by using the micro-stepping capability of the drive system (up to 50,000 steps/rev.). However, this reduces the maximum speed. There is a negligible amount of backlash in the precision rotary table utilised for the vertical axis. The horizontal axis has a direct drive and currently has a resolution of 0.007° using a step rate of 51,200 steps/rev. The stepper motors on each axis have rotary shaft encoders mounted on the rear shaft. A 500-line DRC encoder is mounted on the rear shaft of the motor responsible for rotating the beam about the vertical axis. This encoder has a maximum resolution of 0.18°. Further, the gearing used increases the resolution of this sensing device to 0.004°. The other encoder used is the 1,000-line B57 encoder that is mounted on the rear shaft of the motor responsible for rotating the beam about the horizontal axis. It has a maximum resolution of 0.09°.

The communication between the LIS and the robot controller requires 1.3 seconds to send the signal and acknowledge a new command. Including the LIS system response time, this leads to a total of 1.7 seconds. This is considered to be too slow for LIG technique. The impediments to a faster response include the following:

(1) Slow PSD acquisition sub-system (18Hz).
(2) Slow motor controller.
(3) Long delays in communication between robot and LIS system.

Steps are being taken to remedy the shortcomings of the system. These will be described in the next section.

Conclusion and future work

In this paper, the principle and strategy of the laser interferometry-based guidance technique have been proposed. The overall structure and the required sub-systems for such an approach were presented. The path generation for literally painting the way using a laser was described. The laser interferometer-based guidance apparatus developed has currently a poor response time of 1.7 seconds but with an excellent accuracy of ±5μm.

The initial results are promising. However, some major modifications to the system must be carried out to improve the response time and thus the guidance speed. A faster PSD acquisition sub-system capable of 2,000Hz is currently being developed. Further, a high speed motor controller is also being incorporated into the system. Therefore, the LIS system is expected to achieve a response time in the order of 0.004 seconds. Further, an open-architecture controller is being developed for the robot. This open-architecture controller will be directly integrated within the LIS system, thus removing the communication delay associated with external linking of these systems. A predictive algorithm for the robot manipulator to determine its future position from the past positions, estimated velocity and estimated acceleration values is also being investigated. Further, the laser beam used has a low power (i.e. 1mW). This limits the range of the technique to a maximum of 10m. The beam power detected by the PSD acquisition sub-system when the target is further than 10m will be the same as the average ambient light power. This will result in the disruption of PSD measurements in the LIS system. A laser interferometer with higher laser power is also being considered. The above technique is developed based on a linear path generation. Non-linear path generation method will be investigated to account for obstacles in the workspace.

References

Craig, J.J. (1991), *Introduction to Robotics: Mechanics and Control*, 2nd ed., Addison-Wesley, Reading, MA.

Gander, H., Vincze, M. and Prenninger, J.P. (1993), "An external 6D-sensor for industrial robot", *IEEE Proceedings International Conference on Intelligent Robots and Systems*, pp. 974-8.

Melles Griot (1992), *Optical Beam Position Measurement System*, Melles Griot, USA.

Parker, G.A. and Gilby, J.H. (1982/3), "An investigation of robot arm position measurement using laser tracking techniques system to measure robot arm performance", *Sensor Review*, October, pp. 180-4.

Shirinzadeh, B. (1998), "Laser-interferometry-based tracking for dynamic measurements", *Industrial Robot*, Vol. 25 No. 1, pp. 35-41.

Shirinzadeh, B. and Teoh, P.L. (1998), "A study of predictive control for laser tracking of robots", *Proceedings Pacific Conference on Manufacturing*, Brisbane, Australia, pp. 328-33.

Shirinzadeh, B., Chong, H.C., Lee, K.C. and Teoh, P.L. (1998), "Issues and techniques for interferometry-based laser guidance of a manipulator", *International Conference on Control, Automation, Robotics and Vision*, Singapore, Vol. 1, pp. 271-5.

Spiess, S., Vincze, M., Krautgartner, P. and Filz, K. (1996), "On modelling the kinematics and optics of a laser tracking system for contactless robot measurement", Institute of Flexible Automation, Technical University of Vienna, Austria.

Vincze, M., Prenninger, J.P. and Gander, H. (1994), "A laser tracking system to measure position and orientation of robot end effectors under motion", *Robotics Research*, Vol. 13 No. 4, pp. 305-14.

P. L. Teoh, PhD Professional Disputation Thesis, Department of Mechanical Engineering, Monash University, Australia, August 2002.

Thesis

In a point-to-point operation, the addition of Cartesian position feedback from a Laser interferometry-based sensing and measurement (LISM) technique into a joint-based PID controller improves the static positioning accuracy of robotic devices (e.g. manipulators) based on a joint-based PID controller alone.

Introduction

In robot control, the main objective is to allow the end-effector's position to be controlled so that a desired position can be reached or a desired trajectory can be followed with high accuracy and stability (John J. Craig, 1991, L. Sciavicco, 2000, C. H. An *et. al.* 1988). A general control architecture is shown in Figure 1. This prospectus focuses on the establishment of the proposed Cartesian position feedback control using the LISM technique to improve static positioning accuracy of manipulators controlled by a joint-based PID controller. The following outlines the limitation of the existing joint-based PID controller and the development of the proposed control architecture based on feedback from LISM apparatus.

Arguments for Thesis

1. The current application of joint-based PID control scheme is simple but has low accuracy, as it does not take into account the dynamic model of manipulator and the real position of the manipulator's end-effector.
2. More computation power is needed for more complicated model-based control scheme and it is very difficult, in some cases impossible, to obtain an accurate model. The model inaccuracy creates high uncertainty, which makes the analysis of the control scheme more difficult and time consuming. The uncertainty may also nullify the benefit of such approach.
3. The proposed LISM technique can provide a real time, non-contact, dynamic measurement of Cartesian position with high accuracy. It has a large working range, high sampling rate and automatic target tracking capability. The use of feedback from LISM apparatus maintains the simplicity and improves the positioning accuracy of a PID controller.

Background

In Figure 1, the control system is used to compute appropriate actuators command to realise desired motion. The actual output is fed back into the control system to reduce disturbance and improve stability. The task specification (end-effector motion) is usually performed in Cartesian space whereas control actions are performed in joint space.

Problem with Joint space control

A general joint space control scheme is as shown in Figure 1. In this scheme, the manipulator's trajectory generally specified in Cartesian space is first converted into joint space. This is then followed by a joint space control scheme.

The simplest joint space control scheme is a Proportional, Integral, Derivative (PID) type controller shown in Figure 2. The actuator torque needed can be calculated from:

$$\tau = \ddot{\vartheta}_d + K_v \dot{E} + K_p E + K_i \int E dt$$

where $E = \vartheta_d - \vartheta$ and $\dot{E} = \dot{\vartheta}_d - \dot{\vartheta}$. K_p, K_v, and K_i are the proportional, derivative and integral gain, respectively.

It must be noted that this controller is completely error driven. No model of the manipulator is used at all. Each joint can be controlled as a separate control system. The gains can be selected to critically damp the response and stabilise the system. This simple controller is the most common controller used in the present-day manipulator (John J. Craig, 1991). However, the performance of a manipulator controlled in this way is dependent on the mechanical design of the robot. Since no decoupling is being done, the motion of each joint may affect the other joints. These interactions cause errors that are then suppressed by the error driven controller. These errors change with the configuration and only by the increase of the gains in the feedback loop can the errors be suppressed subsequently. However, there are practical limits to how high gains can be set (John J. Craig, 1991, C. H. An *et. al.* 1988) due to actuator saturation and stability problems. Moreover, there is no single set of gain values which will critically damp the response in various configurations. An average value is usually chosen and this will cause overdamped or underdamped in various extreme configurations.

One way of overcoming the above drawbacks is the use of the dynamic model of the robot in the controller to evaluate the necessary torque output. The inertial parameters of the links and load, the interaction between the links as well as the effect of friction and gravity can be included in the dynamic model. This dynamic model is expected to represent the actual robot dynamics accurately under different loads and positions configurations so that only the small unmodeled dynamics are being corrected by the feedback controller. A complete model-based controller is shown in Figure 3. The actuator torque output can be calculated as:

$$\tau = M(\vartheta)\ddot{\vartheta}^{*} + V(\vartheta,\dot{\vartheta}) + G(\vartheta) + F(\vartheta,\dot{\vartheta})$$

$$\ddot{\vartheta}^{*} = \ddot{\vartheta}_{d} + K_{v}\dot{E} + K_{p}E$$

where $M(\vartheta)$ is the inertia matrix of the manipulator, $V(\vartheta,\dot{\vartheta})$ is a vector of centrifugal and Coriolis terms, $G(\vartheta)$ is a vector of gravity terms and $F(\vartheta,\dot{\vartheta})$ is the friction model. Each element in these models is complicated functions depending on ϑ and $\dot{\vartheta}$, which represent the current joint angle and joint velocity.

The entire dynamic equations must be computed within the loop of this controller. These computations are quite complex which require higher computational power. The resulting system may run at a lower sampling frequency which in general, would degrade the stability and disturbance-rejection capabilities of the system. (John J. Craig, 1991, C. H. An *et. al.* 1988, G. Alici *et. al.* 1993). With the reduction in cost of computer power, this drawback can be eliminated as shown in F. Reyes, 2001. Since joint space controller does not influence the real position of the manipulator's end-effector directly, more emphasis is placed on the accuracy and completeness of the dynamic model. However, the complete and accurate dynamic model in this control scheme is not often known exactly. Uncertainty of structure such as design error, poor calibration, gear backlash, elasticity, friction and etc. are difficult if not impossible to model accurately. Moreover, most robots will be used to pickup various parts and tools. The inertia parameter of the tools changes the dynamics of the manipulator. All these will cause a loss of end-effector position accuracy and thus nullify the extra computations necessary.

Closed loop control using LISM

Recently, laser interferometry-based sensing technique has been proposed for measurement of robot's position in real time (H. Gander *et. al.* 1993; J. P. Prenninger *et. al.* 1993; M. Vincze *et. al.* 1994; Shirinzadeh *et. al.* 1997). Figure 4 shows a diagram of the set-up of LISM apparatus. In this method, reflected laser beam is detected by sensors and the resulting data are used to maintain automatic tracking of the target by adjusting the beam steering mechanism. The LISM technique utilises real time angular and distance data obtained from the high-resolution encoders and the interferometer respectively to provide the position of the target by direct kinematics.

The simple mechanism of a LISM technique allows a simple kinematic model to be used to convert the angular and distance data obtained from the encoders and interferometer, respectively, into Cartesian position of the target. (S. Spiess *et. al.* 1996, 1998). The high coherence of laser beam and the nature of light to travel in straight line eliminate the elasticity problem in mechanical measuring device and can accommodate a large measuring volume. With a proper design and inertia balancing, there will be close to zero effective inertial seen by the actuators and thus minimum amount of actuator uncertainty. These have led to low uncertainty in the position data obtained using the LISM technique (P. L. Teoh *et. al.* Sept. 2002). The retroreflector weights only about 150g and the LISM apparatus has no contact with the robot. Therefore, no constraints are being placed on the robot. Accuracy higher than 50μm can also be achieved as all the equipment used has accuracy in the order of a tenth of a micron. The sampling rate of equipment can reach more than 1000 samples per second. Therefore, real-time dynamic measurements can be achieved. Further, the measurement process is automated with the target tracking capability. As a result, real-time dynamic Cartesian position of the target can be measured at high accuracy.

By adding a feedback loop using the position sampled by the LISM apparatus, a closed-loop control methodology where the LISM apparatus is used as a real time Cartesian position sensor can be established (P. L. Teoh *et. al.*, 2002). The LISM apparatus will maintain tracking of the robot end-effector until it reaches the final position. When there is a discrepancy between the position commanded by the robot controller and the position measured using LISM apparatus, a centre processing unit will be used to determine the offset error. This offset error can be converted to compensation joint angles and fed into the joint space controller as shown in Figure 5. This feedback loop will ensure a higher positioning accuracy as the uncertainty of the structure of the robot can now be detected accurately using LISM apparatus.

Experimental Verification

Experiments have been performed to determine the capability of the LISM technique for closed-loop control of robotic devices. The experimental set-up, shown in Figure 6, consists of the mentioned LISM apparatus and a 3D positioning table. The resolution of each axis of the positioning table is 0.125μm per encoder count. During the experiment, a co-ordinate frame is first setup on the positioning table so that subsequent movement of the end-effector is recorded with respect to the origin and the motion is parallel to the principle axes of this co-ordinate frame. The co-ordinate frames are given in Figure 3. The end-effector is commanded to move along a single axis and the Cartesian co-ordinates of the end-effector are recorded. Two different sets of results are recorded for motion along each axis, one with the application of the LISM closed-loop control and the other without. Figure 7 and 8 show the results of a particular motion. The thicker line in the graphs shows the desired position. Table 1 shows the tabulated results obtained at different configurations and along different path. The results of only two axes are shown as the servomotor controlling the third-axis (i.e. x) is not functioning at the moment.

From Figure 7A and 7B, it can be observed that the normal motion of the end-effector along the z-axis produces a considerable amount of fluctuation in the y co-ordinates. This is due to the vibration of the end-effector while in motion. The mean of the fluctuation is increasing with time which represents a deviation of the end-effector away from the desired path. The increase in the mean value is due to twisting and bending of the moving link under the uneven load of the end-effector and this introduced a final steady state error of 0.06mm in the z-axis and -0.07mm in the y-axis. From Table 1, it can be observed that the steady state error varies with the configuration and the path taken. It is also noted from experimentation that the error is random. A constant offset value cannot be used to minimise the error. The encoder's reading at steady state shows the desired joint rotations commanded, which indicate that it is the structural uncertainty that is causing the error. Since the encoders do not observe the steady state error, a higher gain values will not eliminate the error in this error-driven controller.

With the implementation of closed-loop control algorithm, it can be seen from Table 1 and Figure 8 that the steady state error along the x and y axes has been reduced to 0.01mm, which is the maximum accuracy that can be sampled by the LISM apparatus. The small fluctuations at the beginning of the motion and at steady state are due to the noise in the signal.

Conclusion and Future Work

From the experimental results, it can be concluded that the closed-loop control algorithm using LISM technique improves the static positioning accuracy of a robotic device compare to a joint-based PID controller in a point-to-point operation. This is accomplished by compensating the position offset error caused by the deviation of the end-effector from the desired position due to structural uncertainty such as uneven loading and bending. The final steady state error is close to the maximum accuracy that can be measured by the LISM apparatus. In this approach, high positioning accuracy can be achieved in the simple joint-based PID controller.

Future work includes the study of the performance of the control scheme at various velocities and accelerations. Experimentation of the control scheme will also be performed on more complex manipulator such as the long-reach manipulator, which generally has significant amount of bending and vibration due to long link length and a 6-axis Motoman robot manipulator. Due to the limitation of the controller of the positioning table, compensation can only be performed on non-moving axes. By using a controller where an offset can be added into the control loop of the moving axes, this study can be extended to include a proposed real time LISM trajectory generating technique where via points are generated along the desired path. In this approach, the position offset of the via points are compensated while the manipulator is in motion to improve the trajectory following capability.

References

[1] John J. Craig, Introduction to Robotics 2nd edition, Addison Wesley, pp. 299-360. 1991

[2] M. Vincze, *el. al.*, A laser tracking system to measure position and orientation of robot end effectors under motion, *The International Journal of Robotics Research* Vol. 13 No. 4, pp. 305-314. 1994.

[3] B. Shirinzadeh, "Laser interferometry-based tracking for dynamic measurements", *Industrial Robot*, Vol. 25 No. 1, pp. 35-41, 1998.

[4] J.P. Prenninger, H. Gander, M. Vincze, Contactless position and orientation measurement of robot end-effectors, *IEEE Conference on Robotics and Automation*, Vol. 1 pp. 180-185, 1993.

[5] P. L. Teoh, B. Shirinzadeh, C. W. Foong, G, Alici, The measurement uncertainties in laser interferometry-based sensing and tracking technique, *Journal of Measurement*, Vol. 32 No. 2, pp. 135-150. September, 2002.

[6] S. Spiess, M. Vincze, P. Krautgartner, K. Filz, On modelling the kinematics and optics of a laser tracking system for contactless robot measurements. Institute of Flexible Automation, Technical University of Vienna, 1996.

[7] S. Spiess, M. Vincze, M. Ayromlou, On the calibration of 6-D laser tracking system for dynamic robot measurements, IEEE transactions on instrumentation and measurement, Vol. 47 No. 1, February 1998.

[8] H. Gander, M. Vincze, J. P. Prenninger, An external 6D-sensor for industrial robots, *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Yokohama, Japan, pp. 974-997, July 1993.

[9] B. Shirinzadeh, Alvin Y.L. KOH, Laser tracking and robot dynamic measurements, *Proceedings of ISPE/IEE International Conference. on Robotics & Factories of Futures*, Vol. 2, pp. 886-891, 1997.

[10] L. Scivicco and B. Siciliano, Modelling and control of robot manipulators, Springer, pp. 213-268, 2000.

[11] C. H. An, C. Atkeson, J. Hollerbach, Model-based control of a robot manipulator, MIT Press, pp. 1-110, 1988.

[12] G. Alici and R. Daniel, Experimental comparison of model-based robot position control strategies, *Proceedings of the 1993 IEEE/RJS International Conference on Intelligent Robots and Systems*, Yokohama, Japan, pp. 76-83, July 1993.

[13] F. Reyes and R. Kelly, Experimental evaluation of model-based controllers on a direct-drive robot arm, *Journal of Mechatronics*, Vol. 11 No. 3, pp. 267-282, 2001.

[14] P. L. Teoh, B. Shirinzadeh, G. Alici, Experimental Analysis of Laser Interferometry-based Sensing and Measuring Technique for a 3D Dynamic Positioning System, *Proceedings of Pacific Conference on Manufacturing (PCM2002), In Print.*

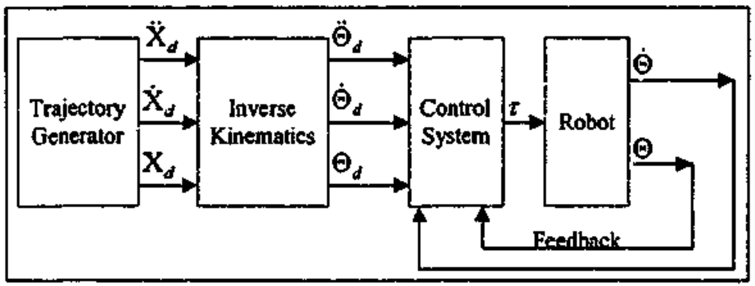


Figure 1: Block diagram of a general joint space position controller

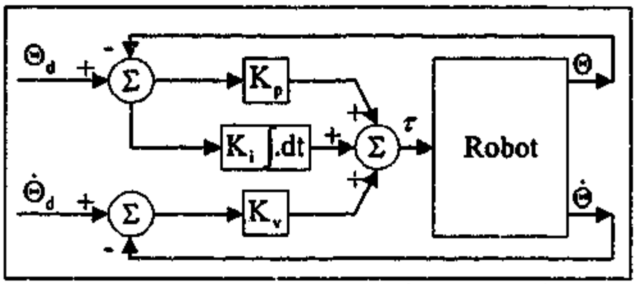


Figure 2: PID position controller

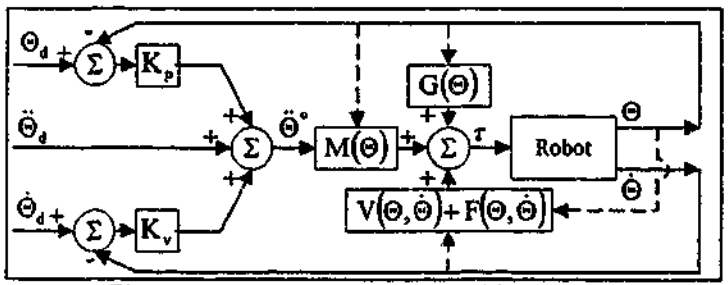


Figure 3: Computed Torque position controller

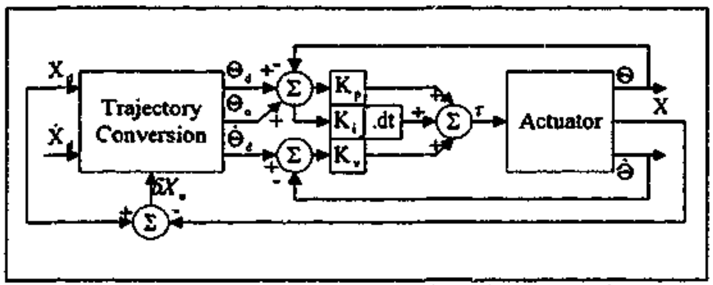


Figure 5: Joint space PID controller with LISM feedback

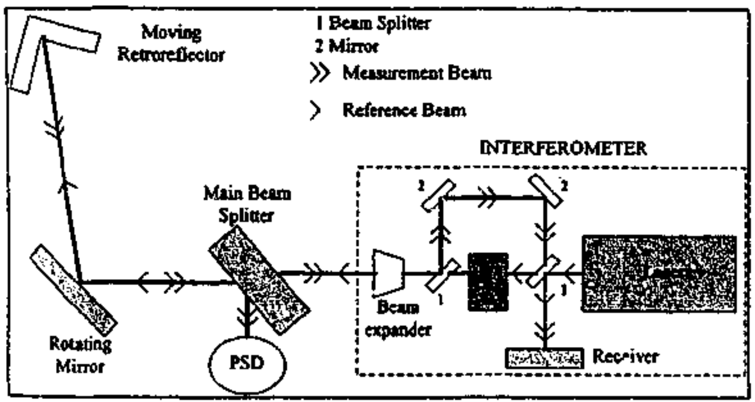


Figure 4: Set-up of LISM apparatus

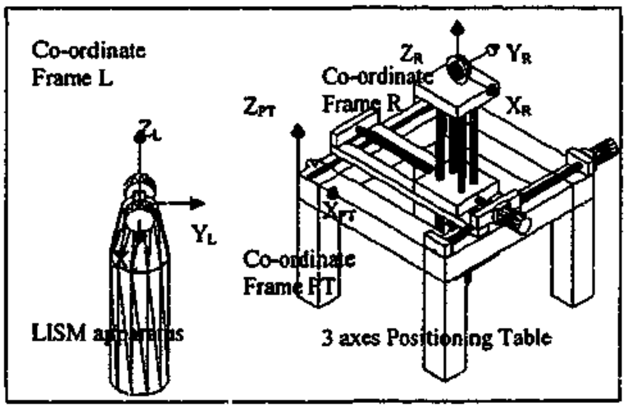
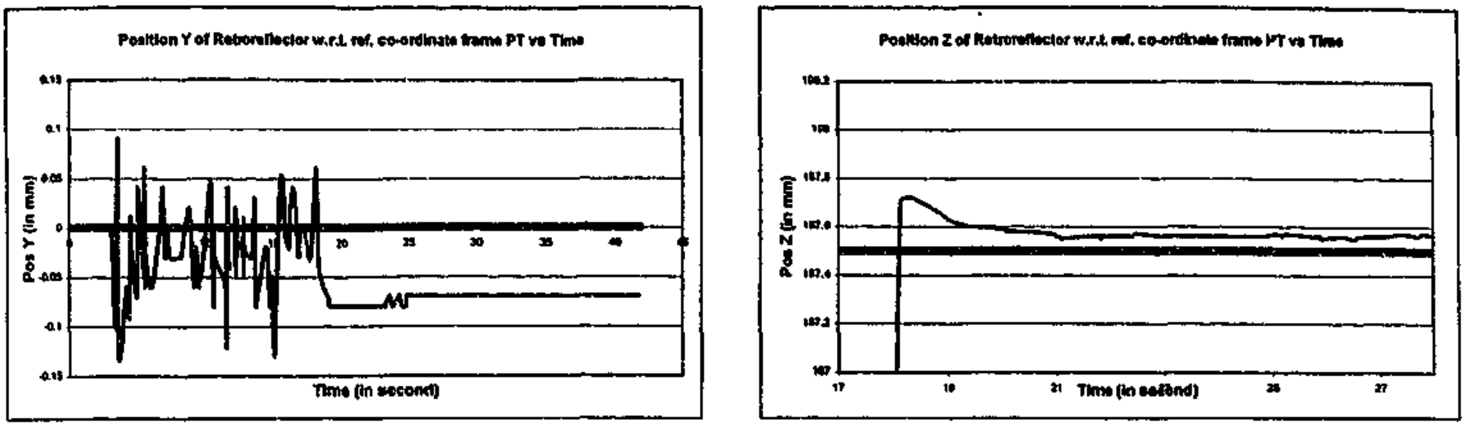
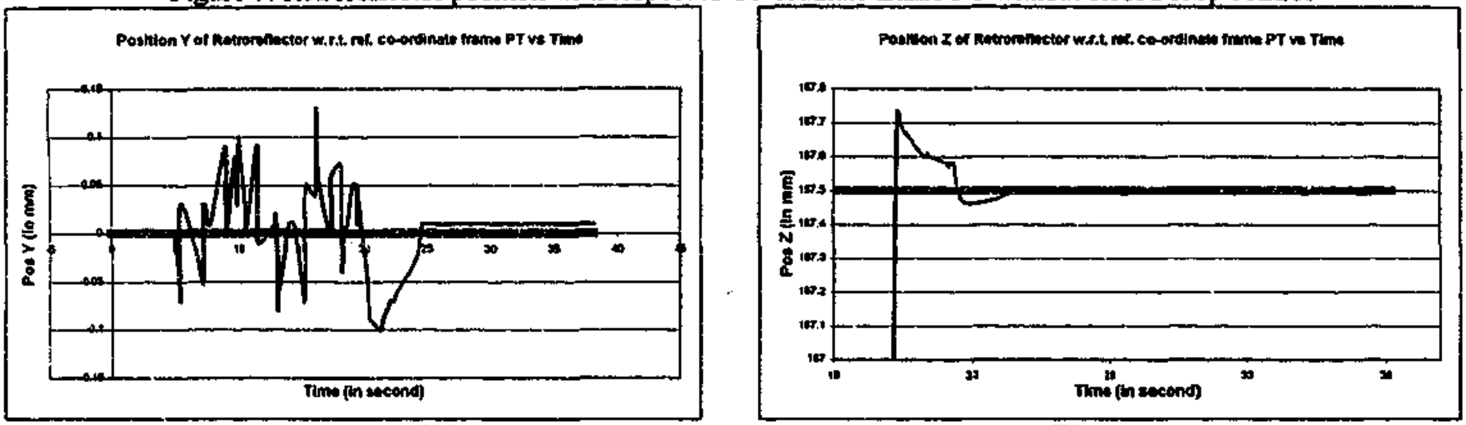


Figure 6: Experimental set-up showing the appropriate Co-ordinate frame



(A) (B)

Figure 7: Retroreflector position with respect to Co-ordinate frame PT without closed-loop control



(A) (B)

Figure 8: Retroreflector position with respect to Co-ordinate frame PT with closed-loop control

Table 1: Steady state error of the manipulator at different configurations moved along different axis

		Without LISM				With LISM	
Current	Commanded	Steady State error		Y- Encoder	Z-Encoder	Steady State error	
Position	Position Increment	y-axis	z-axis	at steady state	at steady state	y-axis	z-axis
y = 0 z = 0	Dy = 0 Dz = 187.5	-0.19	-0.04	0	-150000	0	0.01
y = 0 z = 337.5	Dy = 0 Dz = -187.5	-0.07	0.06	0	-120000	0.01	-0.01
y = 0 z = 0	Dy = 187.5 Dz = 0	-0.49	0	-150000	0	0.01	0.01
y = 0 z = 250	Dy = 187.5 Dz = 0	-0.32	0	-150000	-200000	0	0