2009

Laser assisted telerobotic control for remote manipulation activities

Karan Khokar
University of South Florida

Laser Assisted Telerobotic Control for Remote Manipulation Activities

by

Karan H. Khokar

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Mechanical Engineering
Department of Mechanical Engineering
College of Engineering
University of South Florida

Major Professor: Rajiv Dubey, Ph.D.
Redwan Alqasemi, Ph.D.
Susana Lai-Yuen, Ph.D.
Craig Lusk, Ph.D.

Date of Approval:
July 13, 2009

Keywords: robotics, teleoperation, manipulator, arm, sensor, trajectory, path planning, machine intelligence, Supervisory Control, Traded Control, autonomous, unstructured environment

Dedication

To mummy, papa, Ishan and Guruji

Acknowledgements

First and foremost, I would like to thank my parents and my brother for supporting me throughout my life and giving me the encouragement and motivation to go on and on to achieve higher and higher. I would like to thank Guruji for the blessings bestowed upon us.

This work would not have been possible without the direction, encouragement and support of Dr. Rajiv Dubey. I would like to thank Dr. Dubey for giving me the opportunity to earn a Master’s degree in United States, to work in the field of Robotics that I always wanted to, for all his guidance and for believing in me that I could deliver the goods.

I would like to thank Dr. Eduardo Veras for being my mentor in the research project and teaching me valuable skills in Robotics. I would also like to thank Dr. Kathryn De Laurentis and Dr. Redwan Alqasemi for their supervision of my work at various stages of the project and for their encouragement in obtaining valuable publications. I am grateful to my committee members, Dr. Redwan Alqasemi, Dr. Lai-Yuen and Dr. Craig Lusk for their time and review of my literature.

I am also thankful to my current and former lab members from the Rehabilitation Robotics Lab and Motion Analysis Lab for their kind association and to the office persons of Mechanical Engineering. I would especially like to thank Charlie Baumann from Tampa prep for helping me out with videos. I would like to thank Merry Lynn Morris for helping me enhance my dancing talent at USF. This was something out of the blue!

Lastly, I would like to thank my aunts, uncles, cousins, grand moms, late grand fathers; friends in Tampa, USA and in other parts of the globe and my friends back home in India for their love and support.

Table of Contents

List of Tables

List of Figures

Laser Assisted Telerobotic Control for Remote Manipulation Activities

Karan H. Khokar

ABSTRACT

The effort in this work has been to innovatively use range information from a laser sensor mounted on the end effector of a remote robotic arm in a telerobotic system to assist the user in carrying out remote tasks in unstructured environments. Assistance is provided in the form of Traded Supervisory Control where the human is involved in high level activities such as decision making and the machine generates task plans and executes tasks autonomously using laser data and machine intelligence. In this way human planning and high level decision making capabilities are combined with machine computational and precision task execution capabilities in an optimal way. Laser range data has been used in a novel way to generate trajectories and virtual constraints that assist the user either by executing trajectories autonomously or by guiding the user in teleoperation along specific virtual constraints. The ability of the laser to generate path plans and execute them autonomously, and generate 3D geometry information is another novel feature of the project. This has been achieved without using sophisticated sensory suite and extensive computer processing. The user simply points to certain locations in the unstructured environment by teleoperating the remote arm using the master arm and presses certain keys on the keyboard. The machine using laser data and its intelligence generates the appropriate trajectories, virtual geometric surfaces and path plans which assist the user in executing the task. Time and accuracy results in executing a remote manipulation tasks on a real-time telerobotic system with master and slave arms, with and without laser based assistance have been

generated and compared to validate the hypothesis that laser based assistance improves task performance and reduces the cognitive load on the user.

To improve dexterity of the arm and to enable smooth and stable control of the arm, singularity avoidance techniques have been implemented and results in simulation have been presented.

Accuracy results to validate the motion control algorithms of the robot by comparing trajectories in simulation and on the robot have been generated.

Chapter 1: Introduction

In this chapter we introduce telerobotics and its related concepts. An introduction to the assistance concept is given. The need for this research thesis, thesis objectives and hypothesis, thesis project overview, proposed thesis results and thesis document outline are also presented in this chapter.

1.1. Introduction to Telerobotics and Assistance Concept

As defined by Sheridan [1] a teleoperator is a machine that extends a person's sensing and manipulation capabilities to a location remote from the person. A teleoperator has artificial sensing elements, a vehicle for carrying these around in the remote environment, communication channels to and from the human and actuators for applying force and for manipulating the remote environment. The manipulation is usually achieved by means of a manipulator also known as a robotic arm or simply an arm. Teleoperation is defined as a direct and continuous control of a teleoperator. Teleoperation also has a literal meaning of operating at a distance. There may be some form of feedback from the remote environment to aid the human operator to perform the remote task, however there is no automation or computer control involved in this case. Telemanipulation and telechirics (etymological meaning of 'remote hand') are synonyms of teleoperation.

This was the way remote tasks were executed before the advent of computer technology. Dangerous tasks in hostile environments that were inaccessible to humans were mainly performed using telemanipulation. These tasks included handling of radioactive materials and space or undersea manipulation applications. With the advent of computing technology, computer intermediation came to be used to facilitate remote manipulation tasks. High computing power not only provided rich interfaces with real-time sensor data processing for a realistic feel of the remote environment to the human operator but computer algorithms also infused intelligence into the system which enhanced the way the humans did the tasks. Thus, with the advent of computing technology emerged the field of telerobotics.

Sheridan [1] defined telerobotics as an advanced form of teleoperation in which a remote manipulator is controlled by a human supervisor through a computer intermediary. The remote manipulator is semi-autonomous with some level of local autonomy but the human is always in the loop controlling the robotic arm. The human conveys the plans, orders, goals, constraints to the robot or to the computer and the computer executes these based on the information from the human and from its own sensing. It conveys back information about accomplishments and issues to the human based on which the human takes the next steps.

The addition of computing technology into teleoperation systems brought with itself autonomy to the robot, intelligence and decision making capabilities, sensor data processing, knowledge generation capabilities and so on. All these additional features complimented the human user in executing the task. The abilities of the human user were enhanced. In other words the human was assisted in executing remote tasks. This enhancement of the human ability to execute remote tasks by means of computer intermediation is known as the assistance concept.

Computer based assistance can be provided to the human in several ways. One way is to merge commands from a computer with those from a master device in a Shared Control type telerobotic control. In Shared Control strategy both the computer and the human execute parts of the same task simultaneously. The computer component of the control either relieves the human from a difficult part of the task or helps the human by enhancing human's input by simultaneously executing the task with the human. Some inputs out of the total are from the human and some from the computer control. Either way the human is assisted in task execution. Another way of assisting the human in telerobotic task execution is where the computer executes tasks low level tasks, those that are monotonous or computationally challenging. The human on the other hand executes high level tasks that require human cognitive abilities or human planning and decision making skills. This could be Shared Control or Traded Control. In Traded Control the computer and the human relinquish control to each other at specific instants of task execution whereas in Shared Control both the human and the computer give necessary inputs and simultaneously execute the task. Computer assistance is also provided when the computer generates geometric virtual constraints in the remote environment that guide the human operator manipulating the remote arm along specific trajectories or along certain surfaces of interest. Along with motion, force effects that help the user to feel the geometries kinesthetically provide additional assistance. Potential fields that either attract the human operator controlling the remote arm along certain targets or repel the human operator from certain obstacles also provide assistance to the human.

In this work the machine assistance is in the form of the ability of the computer to generate trajectories, virtual geometries and virtual constraints. These either enable the computer to execute tasks or parts of the tasks autonomously and thus assist the user who is controlling the remote robot in Traded Control. They also provide assistance by guiding the master and hence the user along specific trajectories by scaling their motion along the trajectory (known as scaled teleoperation) or virtually constraining the user along the reference trajectory when the user is executing the task by teleoperating the remote arm. The computer is able to generate the trajectories, virtual constraints and 3D geometries by using the laser range data. For generating results, only Traded Control in which autonomous machine control is combined with human teleoperation is considered. Other forms of machine assistance that of scaled teleoperation and virtual constraints are demonstrated but are not integrated into a task and hence are not considered for comparison in the results section. Now we continue with discussion on the theory of telerobotics and its related ideas.

A term that is closely related to telerobotics is Human Supervisory Control or simply Supervisory Control. As per Sheridan [1], Supervisory Control means that the human is intermittently programming or controlling and continually receiving information from a computer that itself closes an autonomous control loop through artificial effectors and sensors to control a process at the remote environment. Supervisory Control is more of a general term when any machine or process (not necessary a telerobot) is controlled whereas telerobotics is used when specifically a teleoperator and more specifically a manipulator is controlled.

In Supervisory Control or telerobotics, the human is like a supervisor of a process and its involvement depends on how intelligent the process is or the degree of autonomy of the process. On one extreme of Supervisory Control the human is a mere observer and acts only in case of emergencies. Here, the computer has almost complete autonomy. Going by the strict definition, in Supervisory Control the human gives high level commands, acts as per the information from the remote environment and observes the process that the computer executes autonomously the rest of the time. On the other extreme the computer has no autonomy and is only for transferring the information to the human. This is the case of teleoperation in which the human has direct and continuous control of the remote arm or process.

The computer could be involved in simply processing the information from the sensors to be made available to the human (here the computer plays the role of a subordinate). It may be involved in doing certain low level tasks when the human does high level tasks and issues commands (this is form of Shared Control – ‘relieve’ in which both the computer and the human are doing parts of the same task and simultaneously) or may take over the control intermittently depending on the type of sub-task (this is a form of Traded Control – ‘replace’ in which a task is split into sub-tasks, some of which are executed by humans and others by the computer and at a time only one of the two acts).In these forms of control the computer plays the role of a collaborator. The computer may also take a different role than being a collaborator or a subordinate to the human. The computer can take the role of an ‘expert’ also where it advises the human supervisor on the next steps of the task.

In this project, the role of the computer is that of a collaborator. More specifically the control form is that of Traded Control where the human takes over direct control for certain parts of the main task and remains a supervisor at other times when the computer control is active.

Collaborative control between human and robot is mainly of two types, Traded Control and Shared Control. In Traded Control, the human and the robot carry out sub-tasks of one main task at different times taking turns depending on the sub-task. *When* the control is to be taken over by the human or robot is important in this case. In Shared Control, the human and the robot perform parts of the same task simultaneously. *How* the task should be split between the human and the robot is important in this case.

From the human-in-the-loop point of view, the Traded Control and Shared Control modes can be considered as forms of Human Supervisory Control. The computer control provides assistance to the human in executing the task by making the task easier for the human to execute it. Thus, in Shared Control or Traded Control mode the computer may handle tasks that are monotonous and repetitive (dull in nature), computationally intensive (difficult in nature) or too simple for the human to perform. These are generally called low level tasks. Other than the low level tasks, tasks or sub-tasks that require precision or automatic execution are also executed by the computer. On the other hand the human acts as a supervisor and directs the task execution. The human controls the portions of tasks that require human decision making, cognitive and sensorimotor skills. The human also takes the control over from the computer in case of emergencies. These are called high level tasks. Thus apart from assistance to the human, the computer control also improves task performance by speeding up the task execution and executing it more precisely. Other terms related to teleoperation and telerobotics are defined next.

Telepresence is defined as the immersion of the human operator in the remote environment in a manner that the human operator feels than she is virtually present at the remote site. Situation Awareness is another term for telepresence. The immersion of the human is achieved by means of interfaces that project visual, auditory, tactile and kinesthetic information from the remote site to the operator site. These information forms ultimately stimulate the senses of the human operator. Only if the information is delivered in the proper form does the remote operator feel present as if she is at the remote site. Sensors at the remote site capture the information which is processed and converted to a form that can be projected onto the human interface and which is compatible to the humans. Telepresence is the key first step in remote manipulation. Only when the information from the remote environment is available to the human operator in the format that the human sensory organs can perceive or decode the information will they be able to take the appropriate steps to manipulate in the remote environment.

Other terms used in the realm of telerobotics are teleproprioception and telekinesthesis. Teleproprioception refers to the knowledge of the human operator regarding the position and orientation of the teleoperator and its parts with respect to its base and other objects in the environment. Telekinesthesis is teleproprioception and also includes the operator’s knowledge about the dynamics (forces and torques) of the teleoperator and the dynamics of the operator in relation to that of the teleoperator.

1.2. Need and Motivation

Over the years researchers have developed telerobotic systems that provide assistance to the remote user in executing manipulation tasks. Some have made notable contributions in providing assistance to the user by implementing Traded Control or Shared Control modes [2], [3], [4][, [5] , [6], [7], [8] and [9]. Many of these implementations [2], [3], [4] and [5] have Shared Control where the human is always in control of the remote manipulator and the machine inputs are only superimposed over human inputs. The machine never takes complete control of the arm. As a result of this these systems have been used in executing tasks in unstructured environments without generating any 3D environment models. In some of the telerobotic implementations, [7]and [10], there is continuous teleoperation control from the human and the human is assisted either by Forward Simulation [7] or by motion scaling [10]. Again because of continuous teleoperation control, these systems have been able to execute tasks in unstructured environments. For those implementations where machine controls the arm for certain periods, like in Traded Control or in autonomous task plan and path generation implementations, [6], [7], [8] and [11], task execution in remote unstructured environments has been possible either by generating extensive 3D models of the remote environment, [6], [8] and [11], or by generating path plans using Teach and Play techniques [7] and [8]. Generating 3D environment models has been known to be a time consuming process that requires extensive computing resources or extensive sensors and processing of their information. Other methods like Teach and Play techniques have been predominantly slow and have known to be more suitable for tasks in structured environments. Most telerobotic systems are also complex for humans to operate and the way the autonomous task plans can be generated.

Thus there is a need for a telerobotic system with an easy and simple to use interface that is able to define task plans and execute tasks autonomously with minimal human involvement. It should be able to operate in unstructured environments by using simple sensors and basic 3D environment information and should able to provide assistance to the user in manipulating the remote environment.

The assistance via Traded Control in which human intervenes only when it is required and in other cases the human acts as a supervisor is desirable as it relieves the human from continuous involvement on the task. The human is engaged in high level activities while the machine performs low level activities. Moreover there is a need for a system that optimally uses human planning and decision making capabilities and the computational, precision task execution and low level task execution capabilities of the computer or the robot. The human should be relieved while executing cognitively challenging tasks and the machine should be relieved from planning and decision making activities which humans can perform very well.

The concept of human-in-the-loop is a very important concept and the author has used it in this work. Even with the advances in sensors, computing technology and algorithms, and their ability to create models of environments from sensor information and from pre stored data, human is still an integral part of the system and cannot be removed. This is because human is the key decision making and judgment rendering entity that no computer, till date, possesses the capability to [12]. The cognitive skills of humans and its application with their sensor motor skills are incomparable to any robotic system [1]. All these features that humans possess, those of decision making, judgment rendering, cognitive skills and sensorimotor skills are especially important in changing and unstructured environments. Programmable robots and robotic manipulators are not able to perform in complex, unmodeled, unplanned and unstructured environments [13]. Human-in-the-loop is the only solution in these circumstances. Complexity of tasks is another factor that disables autonomous control to perform tasks successfully [4].

Thus we see that human is necessary part in a machine control system especially in telerobotic applications as the human possess the important planning and decision making skills that no machine possesses. Embedding machine enabled planning and decision making has been attempted but at the expense of a lot of programming and a lot of computer processing capability. These systems known as Artificial Intelligence (AI) based systems are still not able to match humans and are nowhere even close to humans. However, machines possess the ability to execute a task with high precision, computationally intensive tasks and other low level task which are repetitive and monotonous. Humans lack the attributes to execute tasks with these characteristic efficiently. The author believes in optimally using the human and the machine such that the positive features of both the human and machine will ultimately improve the task performance and reduce the cognitive load on the user. Speed and accuracy of the task will also improve. A better distribution of tasks between the human and the machine enables the human to better plan tasks and generate the necessary commands. It enables the human to view results, act on them and make corrections online in the event future aborts are likely to occur. On the other hand the machine executes tasks with its own local autonomy, conveys results to the human operator and accepts commands. An example of an optimal interaction between the human and the machine is given in [7], “humans can envision and generate a command sequence much faster than a machine (or robot) can effect manipulation” and this is something that the humans can use to their advantage by ‘typing ahead’ i.e. planning and generating commands while the robot executes low level tasks and catches up.

Another driving factor has been to operate in unstructured environments by creating basic yet reliable 3D environment information, online and quickly, from simple sensor data and without using extensive computing resources. Providing maximum assistance to the user in proper form from simple sensor data that will enable the user to execute complete tasks is another area of motivation.

1.3. Thesis Objective and Hypothesis

The overall goal of this thesis is to evaluate the hypothesis that laser based assistance provided to a user teleoperating a remote manipulator while executing a remote manipulation task improves the task performance and reduces the cognitive load on the user when compared to executing the remote task without any machine assistance i.e. by simply teleoperating the robot. Another objective of this work has been to validate the robot control algorithms that are responsible for generating trajectories and executing motions on the robot. Contribution towards improving the stability and smoothness of control of the robot by implementing techniques like singularity avoidance has also been made.

1.4. Thesis Project Overview

This work is part of an ongoing effort to develop a telerobotic system that can be used to execute tasks in remote unstructured environments by providing assistance to the user in executing the task. Before this thesis work began, a real-time telerobotic test bed with basic control algorithms was in place [14].

The major contribution of this thesis work has been to innovatively use the range information from the laser sensor in a Traded Control to provide assistance to the user in executing a manipulation task in a remote unstructured environment. The novelty of this work is the manner in which range information from the laser sensor mounted on the end effector of the robotic arm has been used to generate trajectories, virtual geometries and virtual constraints as also path plans that assist the user in executing a complete task. The assistance is either by combining autonomous control with teleoperation control in a Traded Control formulation or by directing the user along certain trajectories when the user teleoperates the robot. The manner in which basic 3D environment information and task plans are generated using the laser range information, without using sophisticated sensory suite, extensive computer processing and slow Teach and Play methods is another novel feature of this project. This has resulted into the evolution of a very simple and easy to use telerobotic system in which the user is involved in high level tasks like planning and decision making and the robot with its machine intelligence develops the necessary 3D geometries and task plans for task execution. The robot also executes the task autonomously. The user simply points to certain locations in the unstructured environment by teleoperating the remote arm with the master arm and presses certain keys on the keyboard. The machine generates the necessary trajectories, virtual geometric surfaces and path plans that assist the user in executing the task. At the moment the user takes the help of the overhead view of the remote environment by viewing with its naked eyes, the environment being right at the user site, but in case of a distant environment the idea is to use overhead stereo vision cameras.

Trajectories are generated by using laser range data when the laser is pointed to objects of interest and appropriate key on the keyboard is pressed. Once this is done the machine records the laser range data, carries out the computations of generating the necessary transformation matrix of the point to which the laser is pointing and also executes the trajectory autonomously. The trajectory generated is a linear trajectory from the current PUMA location to the target location. The machine uses the forward kinematics on the PUMA and the recorded range data from the laser for the computations. For more details the reader is suggested to refer to chapter 6. Thus the user simply selects objects and generates commands and the machine generates and executes the trajectories autonomously. This way the user is assisted in doing the task. The objects could be anywhere in the environment i.e. the environment is unstructured.

In a manner similar to autonomous trajectory generation, once the user points to a particular object in the environment and presses a certain key on the keyboard, the machine generates a trajectory. But instead of executing the trajectory in autonomous mode the user teleoperates the arm towards the target and the machine assists the user by scaling up user's motion along the generated trajectory and scaling down the motion input in other directions. Thus if the user deviates away from the trajectory, its motion is scaled down and is scaled up when the user follows the trajectory closely. Alternatively using the force feedback capabilities of the haptic master device the user is pulled into alignment along the desired trajectory while following teleoperating towards the target. Stick forces along the trajectory prevent the user from deviating away from it. This is a form of virtual constraint along a straight line trajectory generated from the laser information.

Task plans generation in the form of path planning is also demonstrated using the laser data. This feature enables the machine to traverse the arm autonomously from any location in the workspace where the arm might be at due to the specific task requirements to the point of interest. The path comprises of three straight line trajectories, one which takes the arm to a certain pre determined height from the current point, the second which makes the arm traverse in the horizontal plane to over the point of interest and the third that makes the arm traverse vertically downwards to the destination point. The height to which the arm end effector rises is such that it is clear of any obstacles that might come in its way. In other words it rises above the tallest obstacle. The process is that the user points the laser to a destination point and on pressing a key the system records the co-ordinates of that point from the laser range information and PUMA forward kinematics. After this when the user gives the command to the robot to execute the trajectory, the machine does the computations of the necessary transformation matrices and also executes the path autonomously. Thus the user's involvement is only in making high level decisions and generating high level commands in the initial stages and later on the human is only a supervisor. This also demonstrates the capability of generating a 3D environment information from basic range data of the laser and the simplicity of the system in executing a task in the unstructured environment.

Another demonstration of environment information generation using laser data and machine intelligence is the autonomous alignment of the robot end effector or the hand over a flat surface. For this the user points the laser pointer to three points on a flat surface of interest (a door or a table top) and presses a certain key on the keyboard. The machine generates the equation of the plane i.e. of the flat surface. Here the machine uses the range data from the laser and PUMA forward kinematics along with vector algebra to generate the equation of the plane surface. The press of the key results in the machine computing the necessary transformation matrices that align the hand with the perpendicular to the plane surface. Here too we demonstrate assistance to the user from sensor and machine as the user generates commands and the robot executes them autonomously in an unstructured set up.

Another contribution to the project has been to improve the accuracy and repeatability of the robotic system by improving the laser data accuracy and reproducibility. The former method of laser data acquisition led to errors in measurement of co-ordinates and in formulation of surface equations due to the inaccurate calibration model. Moreover, lack of data filtering led to its poor reproducibility. This directly impacted the task execution as data from laser is one of the primary sources for computations in the computer to enable autonomous control, generation of virtual geometries, generation of virtual constraints and task plans. Recalibration of laser resulted in new accurate characteristics and filtering of its data led to its better reproducibility. Significant improvement was observed in robotic task execution due to the new and improved laser data acquisition and processing.

Initial part of the project was concerned with the validation of the algorithms of robot motion control, specifically those for trajectory generation, inverse kinematics, forward kinematics, motion mapping from Omni to the PUMA in teleoperation and for communication between the master and the slave robot. Trajectories were generated in simulation and then the same were generated on the actual arm for comparison.

In order to improve the dexterity of the remote arm, singularity avoidance measures were implemented. The implementation of these measures has made the robot motion more stable and smooth in teleoperation which ultimately has improved the task execution.

1.5. Thesis Proposed Results

To evaluate the hypothesis, laser data and machine intelligence based assistive control has been implemented on a real-time telerobotic system and a remote manipulation task has been executed. The time to execute the task and the accuracy with which the task is executed with this assistive control is compared with unassisted mode of control in which the user executed the remote task simply by teleoperating the robot. Accuracy results of the actual arm in executing trajectories when compared to those executed in simulation have been presented. Trajectories generated with and without the implementation of singularity avoidance techniques have been compared with each other in simulation. Results from calibration of laser for new and improved characteristics, improvement in accuracy and reproducibility of laser and the improvement in robotic task execution due to improved calibration model and data processing are presented.

1.6. Thesis Outline

Chapter 1 introduces the reader to the thesis. It starts with an introduction to the broad topic of telerobotics and introduction to the concept of assistance. This introduction is by no means exhaustive and the reader is suggested to refer to the relevant literatures. This is followed by the explication of the need for such a work and as to what motivated the author to do the work. Next comes the thesis objective and hypothesis followed by the project overview. The reader gets an idea of the scope of the project in these sub-chapters. Finally, the results that the author proposes are described.

Chapter 2 presents in detail the previous work in the area of telerobotics, machine assistance, use of laser sensors in the field of robotics and generation of motion, paths and virtual geometric constraints using various sensors.

Chapter 3 introduces the reader to the design of the telerobotic system that the author has used for testing his hypothesis. Though this is not a part of the implementation of this thesis, nevertheless it is important to mention it here in detail as the author had implemented his algorithms over this system and it required a thorough understanding of the system. Improved laser calibration model generation, laser data filtering methodology and the steps undertaken to interface the laser to the Omni PC has been described.

Chapter 4 gives in detail the kinematics of the robot manipulators. This is useful as it contains the basic building blocks that the sensor based control algorithms described in chapter 6 will be based upon.

Chapter 5 is on the implementation of singularity avoidance techniques on the PUMA controller so that there is smooth and stable operation of the arm. We introduce the reader to singularity, various techniques for singularity avoidance and the SR inverse of Jacobian technique of singularity avoidance that was implemented. We also discuss the procedure based on heuristics that was used to determine the parameters used in the implementation of the SR inverse of Jacobian based singularity avoidance technique.

Chapter 6 presents the implementation of the laser based assistance concepts on the telerobotic system. It presents in detail the concepts behind the various capabilities of the laser based control like autonomous trajectory generation, virtual constraint generation, virtual geometry generation, and path plan generation. The chapter also demonstrates the application of these laser based capabilities in the execution of a task and the manner in which they assist the user in executing a task.

Chapter 7 presents the results related to the experiments conducted to evaluate the hypothesis that laser based assistance improves the task performance and eases the task execution for the human operator. Results to validate the telerobotic test bed motion control algorithms and those related to singularity avoidance implementation are also presented. Moreover, results showing the improvement in accuracy and reproducibility of the laser distance measurements due to its new characteristics and data filtering, and the improvement in laser based environment information and task plan generation in the telerobotic set up have been demonstrated.

Chapter 8 concludes the thesis and lists the items for future implementation.

Chapter 2: Literature Review

In this chapter, the reader is given a historical perspective of research in the area assisted telerobotics and laser/sensor based robotic control. The state of the art in this area is presented.

2.1. Computer Assistance in Telerobotics

As mentioned in chapter 1, computer assistance in telerobotic control enhances the ability of the human operators in executing the task. We had also mentioned the various forms of computer assistance provided to the humans. One form was that of Traded Control in which there is distribution of tasks so that each of them, the human and the machine perform tasks that they are good at. The other was Traded Control in which machine enhances the human input. Geometric virtual constraints for guiding the user along specific trajectories or surfaces and potential fields for attracting (or repelling) the users to specific targets (or obstacles) were other forms of machine assistance that were mentioned. Here we give a historical overview of these and other forms of machine assistance that not only improve task performance but also ease the task execution for the user.

We also present a historical perspective on the use of the lasers in robotics since we have claimed to have used the laser sensor in a novel way in this work. Moreover, the state of the art in the use of sensors in motion planning, trajectory generation, virtual geometry generation and virtual constraint generation is presented.

2.1.1. Assistance by Combining Autonomous Control with Teleoperation

Hayati and Venkataraman [2] proposed Traded Control of remote tasks on a hybrid motion/force controller for space applications with time delay. Here they have used autonomous control for parts of the task that have force control and have used teleoperation for parts of the task that require motion control. Backes [3] implemented the Traded Control architecture that Hayati and Venkataraman developed. Autonomous mode enhances the teleoperation mode by controlling the motion and force in contact tasks autonomously. This is especially useful in space applications where there is a considerable time delay in the feedback reaching the operator on the Earth. While teleoperating the operator may inadvertently move the end effector into a surface because of delay in feedback but the autonomous controller keeps the forces at safe levels by controlling the motion and force. Teleoperation enhances autonomous control by providing real-time and accurate positioning. This is especially true in case of poorly modeled unstructured environments where complete autonomous control is not possible. Thus in a task those end effector degrees of freedom that require force control are controlled autonomously whereas those that do not are controlled by teleoperation. They use ‘shared selection’ vectors that specify which Cartesian or end effector degrees of freedom (DOF), out of the six available, will have the sharing between the autonomous and teleoperation modes. Weighing matrices then specify how much of the control proportion is teleoperation and how much is autonomous. They claim to have captured the positive features of both autonomous and teleoperation control modes in this way.

Yokokohji et al. [4] implemented Traded Control as well as Shared Control on a 1 DOF arm using a bilateral kinesthetically coupled control framework that they had designed. They named Traded Control as ‘serial mode’ and Shared Control as ‘parallel mode’. They also termed the combined human and robot control as ‘semi-autonomous’ form of control. Their main contribution is in determining and validating a sequence of mode change for safe and stable operation and determining the best mode for remote task execution. Safety was validated by proving that the change of the amount of force applied at the slave side is smooth and not sudden. For determining the best mode out of bilateral teleoperation, autonomous, shared (autonomous and bilateral teleoperation) and shared (autonomous and unilateral teleoperation) modes they conducted two experiments. In one of the experiments the remote operator had to apply a certain force to a constantly rotating object. In bilateral teleoperation the force applied at the master was transferred to the slave. However with Traded Control modes the minimum necessary force was applied by the autonomous control and when there was a need to apply more force the operator would intermediate. They proved that autonomous combined with teleoperation mode is the best mode as it informs the operator at the master if more than necessary force has been applied and so the operator can take the appropriate action. This way the task efficiency is improved. In the other experiment in which a set of LEDs were blinking in a pre defined pattern and the arm goes to the blinking LED but an operator input was necessary if an irregular LED blinked, they proved that with autonomous combined with teleoperation, the burden on the operator is reduced as the operator intervenes only when necessary (when an irregular LED blinks) and at other times is only a supervisor.

Tarn et al. [5]have tried to develop Traded Control scheme at task level based on their earlier developed event based planning and control theory. They claim that Traded Control is a must in certain applications, like multi-arm co-ordination and compliance control, in which neither the human nor an autonomous planner/controller can do the task alone. They also claim that Traded Control is difficult to execute as it is time based and many unexpected events that require mode change happen at any time. This untimely transition in mode also leads to instability. More specifically autonomous control is time based and needs to be reset every time a switch is made to teleoperation control mode. However, we have shown in this work that the transition between modes is smooth and the task performance is unaffected by transitions in Traded Control. In fact we have shown that the task performance improves when we use Traded Control. We have ensured system stability. This has been made possible because of the modularity in task execution. A switch is made from one mode to the other only when the current mode completes its iota of task. In case of unexpected events, such as unexpected obstacles, the control is transferred to the teleoperation mode and the autonomous path is replanned. The replanning is laser based and hence is not time consuming.

Tarn et al. have tested their implementation on an obstacle avoidance task and force/impact control task. In obstacle avoidance task, human intervenes when an unexpected obstacle is encountered and the human input is superimposed with autonomous planner motion to avoid the obstacle. Once the obstacle is cleared 'time optimal recovery plan' is used replan the path. The force/impact control application is similar to that implemented by Hayati [2], Backes [3] and Yokokohji [4]. They claim that this fusion of human and machine intelligence brings about intelligent control, robust performance and reducing the requirement of operator skill. They have also shown the application of Shared Control in a dual arm telerobotic control in which co-ordination between the two arms is performed by the autonomous controller and manipulation by human operator via teleoperation.

2.1.2. Assistance by Interactive Task Execution

In [6] Hirai et al. present an intelligent, co-operative and interactive telerobotic system with a knowledge base that will assist the users to execute tasks. The co-operation is at various levels like motion level and intelligent level.

Their knowledge base consists of position and orientation (POSE) information of models, their affixment relationship to other objects in the environment, the handling procedures like grasp POSE, approach POSE etc. which have all been implemented using object oriented programming (OOP) approach. Inheritance concept has been used to describe handling procedures for general and specific handling. They have been able to utilize the features of OOP to create a hierarchical and modular database of models and procedures. So if a specific approach needs to be computed, the corresponding procedure is executed. This will be a sub-step of the general POSE calculation method. A lot of software writing for object models, their properties and procedures, rule base in 'motion understanding system' and so on has been carried out.

At the intelligent level they claim that the knowledge base information of object properties and handling procedures will assist the system by invoking methods to execute next sub-tasks and will assist human users by providing global information about task outcomes and next steps via audio interface as also information about next POSE that the robot is likely to achieve via simulation. This will provide co-operative control. It also informs the human operator of any errors, asks for confirmation to proceed with autonomous tasks and requests the operator to teach certain sub-tasks which is cannot execute from knowledge base data. Thus the authors provide co-operative control by interactive task execution keeping human in the control loop for necessary intermediation while the intelligence comes from the knowledge base. This is different from the Shared Control and Traded Control l methods. It is a form of Supervisory Control with need based human intervention through interactive user interface.

At the motion level the co-operative control is more like the conventional Shared Control. This is provided by superposing the motion generated by language command (autonomous) to the motion generated by master manipulator or teach pendant (teleoperation). Virtual fixture (mentioned as software jigs here) constraint motion of certain joints while others are teleoperated, teleoperation mode is simply superimposed over autonomous when needed and so on.

2.1.3. Assistance by Forward Simulation

In [7] the authors have presented the concept of tele-autonomous control in which sharing of manipulative and cognitive activities between the human and the robot takes place. In telerobotics, where human manipulation is projected to the remote environment, the concept proposes projection of human cognition as well. In AI based autonomous system, where human supervisory intervention is primarily at cognitive level, the authors propose intervention for manipulative tasks as well. This way the intend to mirror the two technologies and merge them into what they call as tele-autonomous systems. They intend to build a bridge between telerobotics and intelligent autonomous systems by providing methods to make real-time transitions between human and machine control of remote events. These methods involve decoupling the master and slave to relax certain constraints. Relaxation of the time and position constraints enables the operator to plan tasks with forward simulation assistance faster and utilize the time when the slave catches up to do other high level tasks. The humans can envision and generate a command sequence much faster than a machine (or robot) can effect manipulation. This way there is a distribution of high level and low level tasks between human and machine with the advantage of improving manipulation time performance.

Their basic work revolves around relieving the operator from 'time slaving' in the case of telerobotic systems with time delay (like space teleoperation) where the operator is restricted to carry out large movements of the master and has to wait for the slave to catch up with the master POSE. Even with the forward simulation and predictor display technologies proposed by Sheridan et al. the operator cannot make large movements as in case of an erroneous move the command buffer has to be cleared and the slave has to be brought at the previous good POSE. By removing the time synchrony, the operator can 'type ahead' by making large moves and get involved in other high level tasks while the slave catches up. They do this by disengaging what they call Time Clutch (basically depressing and releasing a control pedal). Once the Time Clutch is disengaged the command buffer that sends the control signals from the master to the slave is no more populated (so the slave does not get any new commands) but the path planning is still done with the user getting assistance from forward simulation which simulates the slave to move like the master without any time delay. Once the user is satisfied with the path, the clutch is engaged and the command buffer starts getting populated to move the slave to next position. While the slave is moving the operator can again disengage the clutch to type ahead thus saving time and doing other critical tasks while the slave catches up.

They have also introduced, what they call as Position Clutch in which the position synchrony that constraints the operator to move the master only as fast as the slave can move is removed. Also like the Time Clutch disengagement, the command buffer is not queued any more. As a result the operator can move the slave at any speed (unlike the Time Clutch disengaged mode, the forward simulation is not constrained by the slave speed). Once the Position Clutch is engaged again (and the Time Clutch is engaged also if it was disengaged) the path is planned automatically and command buffer is queued to move the slave to the destination POSE. This can be useful where the operator is required to do a complex maneuver or locate the end effector in a complex orientation. Again the operator can type ahead and do other high level tasks while the slave catches up.

Other novel concepts like Time Ratio Controls in which the speed of the slave in forward simulation with Time Clutch disengaged can be changed and Time Brake in which the command buffer is cleared and the robot begins to move to the last good POSE before the Time Clutch (or Position Clutch) was disengaged. This is done in case an unexpected obstacle comes in the way of the planned trajectory with time (or position) clutch disengaged and needs to be replanned.

The authors performed experiments to measure time performance in a task completion based on Fitt's law. The task comprised of moving the manipulator to five locations on a 2D surface. The distance & target width parameters of Fitt's law and simulated communication delays were varied and three modes teleoperation, teleoperation with forward simulation and teleoperation with forward simulation and Time Clutch disengaged were tested and the last mode was the fastest in completing the task and the operators could reach, stop to a goal point and move to the next one more accurately and finely in the last mode.

2.1.4. Assistance by 3D Models and Graphical Techniques

In [8] Anderson has demonstrated the usage of graphical techniques along with Supervisory Control (Traded Control and Shared Control) that assist the user in executing a task. The graphical techniques are in the form of programming techniques, intelligent 3D graphical modeling of the environment and a virtual environment for task simulation.

The graphical programming techniques are in the form of automatic code generation by selecting icons of various functional modules and connecting them in a sequence to a form a task. Once the modules are selected and connected the system automatically generates the code for each module and also assigns processors for each module and so on. This way the operator can design any process with either autonomous, teleoperated, Shared Control or Traded Control while working at a high level while the computer system generates task plans. The intelligent 3D graphical modeling of the environment is achieved using commercially available robot simulators and then applying Teach and Play on the virtual environment so that the computer stores the tag points. These tag points are later used by the graphical programming system to generate path plans and execute task. The 3D modeling of the environment is a time consuming process though.

This way the graphical techniques allow the user to focus on high level tasks as the system generate task plans from high level user specifications and then the user uses Supervisory Control for task execution.

In [9], Hamel et al. have developed a real-time controller and have used the concept of HMCTR (Human Machine Co-operative Telerobotics) in which the human is assisted but never superseded by machine intelligence. Here they provide machine intelligence by sensor and model information. This is then used with computer assisted modes of operation, like autonomous combined with teleoperation for remote task execution.

For executing a task, a task plan is generated. The task plan is a description of the sequence of atomic actions that are to be executed for completing the task. The task plan is generated from the 3D task scene model which in turn generated from the sensor and 3D model data. The task plan along with teleoperation mode is then used by the HMCTR controller which is based on the real-time ControlShell to execute the task.

The results show that telerobotic mode is faster than the pure teleoperation mode. Moreover, the workload on the human operator is significantly reduced, remote work efficiency is improved, dependence of human skill on effective remote operations is reduced and training time can be reduced.

2.1.5. Assistance by Human Machine Co-operative Control

In [10] Everett and Dubey have presented a form of sensor and model based machine assistance in which they augment the operator's motions depending on the accuracy of environment information from sensor and model data. For this they adjust system parameters not under direct control of operator like position and velocity mapping depending on the accuracy of environment information from sensor and model data. From another point of view, the operator’s inputs are altered depending on the accuracy of information from sensor and model. Inaccurate sensor and model information implies that the operator's input should be left almost unchanged shifting more responsibility towards the operator for obstacle avoidance or goal attainment. This is more like unassisted form of teleoperation. On the other hand when the sensor and model information is more reliable and accurate, the operator is assisted in a telerobotic mode to execute the task.

To demonstrate the concept, an approach and impact task of the end effector with a surface was performed. In this task it is desirable to move quickly towards the surface and minimize impact probability. The burden on the operator to achieve this is alleviated by adjusting the scaling between the master and slave velocities which is further based on the accuracy of the range data from a laser range sensor mounted on the end effector. As long as the sensor information is accurate, the velocity of the slave with respect to the master is scaled up and as the end effector approaches the surface and the accuracy drops, the velocity is scaled down until there is no scaling. Results from experiments show reduction in approach time and impact force and increase in operator confidence when performing the task with this method.

2.1.6. Assistance by Geometric Virtual Constraints

Joly and Androit [15] argue that for certain tasks less than 6 DOF are required. In fact for these tasks less than 6 DOF improve task performance. Like in the inspection of pipes, the cameras need to move only along cylinders around pipes. If the cameras are constrained along these cylinders, task execution i.e. recording proper images improves and there would be fewer collisions. They present the idea of constraining the user's motion in teleoperation to virtual geometric constraints by using virtual mechanisms and the master and slave being connected to these virtual mechanisms by virtual springs and dampers. They have developed suitable control law. They conducted experiments and compared the motion of master and slave for orientation, screw, cylinder and a complex curve constraint. Their experiments showed that the control law they have developed was successful in constraining the slave along desired motion trajectories and surfaces even if the master motion was disturbed.

2.1.7. Assistance by Potential Fields

Aigner and McCarragher [16] present a Shared Control type of framework to allow sharing of human commands with an automatic controller. Here they have used attractive and repulsive potential fields to assist the human in moving towards a target (attractive potential) and move away from a target (repulsive potential). The human motion is superimposed with the motion from an automatic controller but if the human tries to go towards an obstacle or outside the boundary of the workspace, the human is restricted from doing so by the potential field.

The potential fields are generated by adding individual attractive (targets) and repulsive (obstacles and boundary) to form a composite potential field. The individual potential fields are specific geometries like ellipsoid to represent an obstacle, a conical well to represent a target and a rectangular parallelepiped to represent the workspace and its boundary. The equations of these geometries are such that potential values are generated depending on where we are in that geometry. For example, as we go towards the center f the conical well the value of potential increases. On the other hand as we go towards the boundary of the rectangular parallelepiped, the potential is reduced. Then the derivative of the potential of the composite field gives the velocity from the autonomous controller over which the velocity input from the human is superimposed.

2.2. Use of Lasers in Robotics

Laser pointers and range finders have been used predominantly in robotics as a pointing device, as a proximity sensing device, as a device for determining the 2D POSE of mobile robotic platforms and for determining virtual force on the end effector of a manipulator when the end effector is at a certain distance from a surface as if the end effector is touching the surface.

Laser has been used to solve the problem of localization and is considered powerful and accurate sensors for the localization application [17]. Localization is determining the POSE of a mobile robot at a point in its trajectory and is achieved by using a laser range finder by a method called triangulation. The figure shows the apparatus required for the method that is mounted on the mobile robot platform. It consists of a laser scanner system which basically consists of a laser diode, servomotor with its axis vertical, photo detectors and a computer. Also reflectors mounted on the walls at regular points along the trajectory of the robot are part of the set up. The (x, y) positions of reflectors are known with respect to the world co-ordinate system. As the servomotor rotates the head on which the laser diode is mounted by 360 degrees, the laser beam is reflected by reflectors on the walls. These reflected beams are captured by the photo detector. The computer based on the information from the photo detector and the servomotor encoders computes the angular positions of the reflectors with respect to the robot axis. Once this is known the following set of equations are used to determine the position (x, y) and the angular position (θ) of the robot with respect to the world co-ordinates.

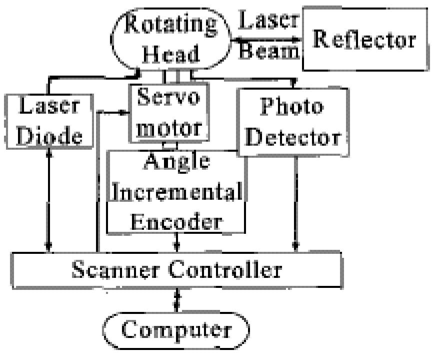


Figure 1: Hardware for laser based triangulation [17]

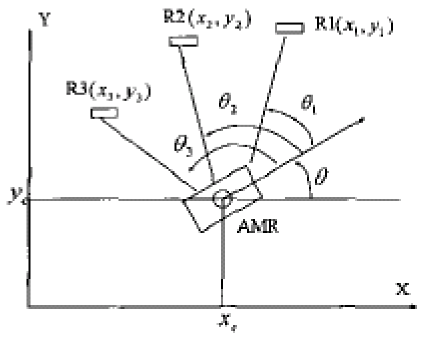


Figure 2: Co-ordinates for triangulation [17]

In [18], Rizun and Sutherland have developed a telerobotic system for robotic surgery applications wherein they provide a sense of touch to the remote operator at the Omni master with the laser mounted on the slave end effector simply based on the laser range information. When the laser beam is focused onto a real surface and when the focal point of the laser beam is coincident with the surface (which would happen within a certain distance of the surface) it gives an impression to the operator of touching the surface even though the remote arm is physically not touching it. With this system they intend to provide haptic feedback with tools that are non-contacting and non-haptic in nature. They have also developed a sense of friction by using Coulomb friction model and compliance (hardness) of the surface by using spring model. The laser tip penetration and velocity is used to determine spring force and friction force.

Referring to figure below, n is a unit vector pointing in the direction of the laser beam, e is a unit vector whose direction is along n but the sense of direction depends on whether the laser focus penetrates the surface or not. If the laser penetrates the real surface then the virtual surface is as shown in the figure and e is in the direction of n.

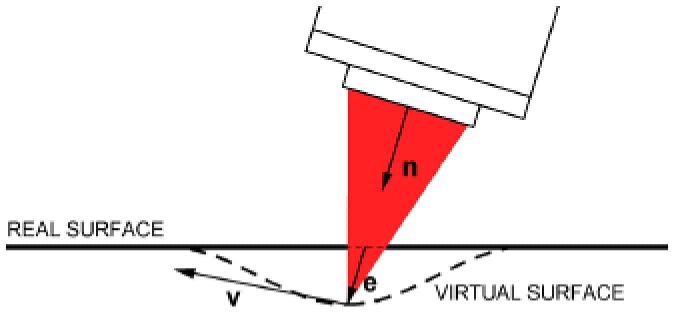


Figure 3: Laser based haptic feedback for non-contact devices [18]

If the laser focus is on the surface then e is zero and of the laser focus is above the real surface then the virtual surface is also above the real surface with the direction of e opposite to that of n. The direction of e is determined by the laser system from the pixel information of the photo detectors and the dot product of e and n is calculated accordingly. Then the following force equations are used to compute the force that is sent to the Omni haptic device. The first component of the force is the spring force for compliance of the surface and the second one is the Coulomb friction term. The sign is negative for friction as it acts in a direction opposite to the motion and for hardness it acts in the direction opposite to that of application.

$$\mathbf{f}_c = \begin{cases} 0\ , & \mathbf{e}\cdot\mathbf{n} < 0 \\ -(k\mathbf{e} + \dfrac{\mu k e}{v}\mathbf{v})\ , & \mathbf{e}\cdot\mathbf{n} > 0 \end{cases}$$

Figure 4: Force equations in laser based haptic feedback [18]

This is an example that shows the use of Phantom Omni as a device used for haptic transformation i.e. transformation of non-haptic information at remote site into haptic information at master site via the use of computerized robotic system, sensors and haptic feedback capabilities of Phantom.

Laser range finders have also been used in the digitization of the large 3D indoor environments [19]. 3D laser range finders have been used in these cases. The laser scanners scan the environment and this information is used by the robot for localization, computation of its POSE for the next step in its path and for avoiding obstacles by computing their POSE.

Hasegawa et al. [11] developed a model based robot system in which a 3D model of the environment was created interactively with the operator and by sensor information. This 3D model was then used to achieve autonomous robot control in uncertain environments.

They used a laser sensor with triangulation technique to determine the 3D co-ordinates of certain points on the objects. The system would then take the laser sensor closer to the object and determine more 3D points on the object. This information along with the information of object's CAD models from a data base was used by the system to provide the operator with alternatives of object POSE. An overlaid object model on the video feed was shown to the operator. The operator would select the right POSE. This way the system would generate the 3D environment information interactively and from the available information in the database. The system would then create path plans for task execution. The laser pointer is controlled using a track ball or a program and the control is different from the manipulator.

Takahashi and Yashige [20] presented a simple and easy to use laser based robot positioning system to assist the elderly in doing daily pick-and-place activities. Their basic concept of target selection and trajectory generation from laser data is similar to this work however it differs from our implementation in many respects. Firstly, it is not an anthropomorphic arm but an x-y-z linearly actuated mechanism mounted on the ceiling and that moves downwards in the z direction to grasp an object. Because it is not an anthropomorphic arm it is incapable of being oriented in all possible configurations that an anthropomorphic arm can attain and this limit not only the variety of Activity of Daily Living (ADL) tasks that it can do but also the type and number of objects that it can grasp. It has to be taken to a certain home position every time a new object needs to be picked up or placed. The laser is mounted separately from the robot and is not integrated with the robot control system. Therefore the user controls the laser pointer independently from the robot. Very basic trajectory generation algorithm has been shown and no task execution has been demonstrated. The system is limited in the tasks that it can do, that are generally expected from a robotic system to perform.

2.3. Sensor based Motion Generation and Virtual Constraint Generation

In [21] Cheung and Lumelsky present an algorithm that enables the arm covered with sensorized skin layer to autonomously move around unexpected obstacles while traversing on a trajectory to reach a certain target. The path around the obstacle is planned based on the sensors mounted on its skin. Arrays of infrared proximity sensors are mounted on the arm. When an obstacle is detected local normals to the obstacle are computed at the refresh rate of the arm movement. These local normals determine incremental local motions at each time step that the arm would execute so as to avoid obstacles. The arm either moves away from the obstacle along local normals or along perpendiculars to the local normals. The particular plane in which the arm follows the contour around the obstacle is determined by certain criteria. The arm continues to follow the contour until it meets the original trajectory. From this point it continues along its original path. This work is more on the lines of detecting obstacles from proximity sensors and generating incremental motions until the original path is connected to.

In [22] Nayak and Ray present a weld seam tracking robotic system for welding unknown seams using a laser range finder. The system is real-time, adaptive, intelligent and suitable for unstructured environments.

The controller they have developed is a two tiered control system with a high level controller and a low level controller. Also the end effector moves over the seam twice. In the first run, it locates the seam in the 3D space. The arm movement in this case is guided by the operator. In the second run, it simultaneously generates the seam model from range image generated by the laser sensor and imbedded intelligence and also welds using information from the generated model and POSE information for the end effector generated from the laser data.

The simultaneous model creation and welding is possible as the laser sensor is followed by the welding torch and the information from the sensor continuously positions and orients the torch. In this sense the system is adaptive.

The high level controller creates seam model in real-time from range image generated by laser data and using a reasoning system. The distorted range image that may have been created due to inaccurate movement of the end effector (and thus the sensor) over the seam (as the seam co-ordinates are not known initially) is transformed to a co-ordinate system different from that of the sensor. The reasoning system is the intelligence imbedded in the system to formulate the seam model from the range data.

The low level positions and orients the torch correctly over the seam based on co-ordinate information from the laser and the information from the seam model and also manages the correct speed of the movement of the end effector to generate a weld.

Here the laser sensor has been used to generate a range image as well as position and orient the torch and hence the end effector in every cycle based on the seam location information stored in the system from the first run.

Chapter 3: System Design

This chapter introduces the reader to the hardware and software of the telerobotic system that the author validated and executed his algorithms on. It also describes the modification in the system for new and improved laser characteristics and laser data filtering. User interfaces that connect the user to the telerobotic system are also described.

This telerobotic system is a general master-slave system. PUMA560 manipulator from Unimation is the slave robot or the remote manipulator and Phantom Omni from SensAble Technologies ® is the master device. These are interfaced to separate PCs through their controllers. The communication between the PCs is established via Ethernet. Sensors mounted on PUMA are also interfaced to PCs.

3.1. PUMA560 – Remote Manipulator

PUMA560 manufactured by Unimation Incorporated, a Westinghouse company (now owned by Staubli) is the remote manipulator or the slave robot. It consists of six revolute joints and hence is a six DOF manipulator. Servomotors through a system of gears move and control each joint of the PUMA. Each joint has an individual drive system through its servomotor and gear system. Potentiometers and incremental optical encoders provide feedback for the closed loop control of the motors.

PUMA has joint limits on each joint. Joints 1 and 2 are provided with hardware limits in the form of hard stops in the bull gear of the joints. These can be changed (reduced but not increased) by opening the cover of that joint and changing the hole in which the bolt is placed on the gear. Other joints have software limits i.e. limits set by the system software. Some important specifications of PUMA are:

1. Position Repeatability: +/- 0.1 mm.
2. Payload: Either 2.5 kg or 4 kg
3. Clearance volume required: 0.92 m radius with center at shoulder
4. Weight: 63 kg
5. Mounting orientation required: Vertical

3.1.1. PC based Integration of the PUMA

The PUMA originally comes with a control chassis for control and peripherals for the user to program and command it. The control chassis has I/O boards (similar to analog and digital cards of Programmable Logic Controllers or PLCs) that convert information format for sensors (encoders) and digital circuitry, a processor to process user information and send appropriate information to various modules, power supplies and amplifiers. With this set up, the system is not an open system. The robot can be controlled in only the specific language and with the specific user interface devices. Scaling the system and programming it to execute more applications is not possible.

In order to make the system open and with it bring all the advantages of an open system, the PUMA has been interfaced to a PC. With this interfacing it is now possible to write programs in general purpose languages like C++ on general purpose operating systems like Microsoft Windows. Moreover interfacing of various devices is easier which can be done by using various PCI (Peripheral Component Interconnect) cards compatible with the particular device. So the system is open and is scalable in terms of hardware and software. More heterogeneous devices can be interfaced to the system with ease and the processing power of the PC can be utilized for programming and computations. Thus the system can be based on the PC for computations, communication and connections (or interfacing) among devices. Again PC is a commonly available resource. The success of the conversion of the system from a closed to a PC based open system is credited to Eduardo Veras [14].

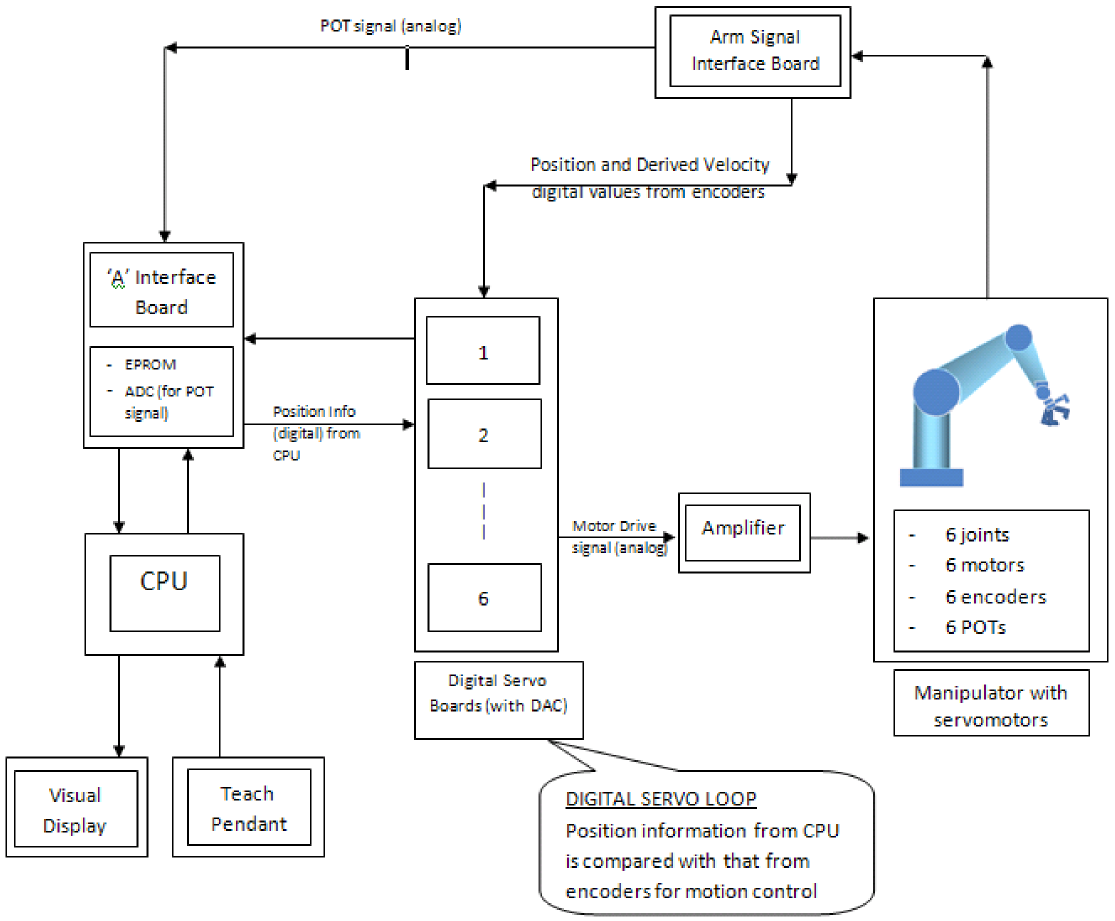

POT signal (analog)
Arm Signal Interface Board
Position and Derived Velocity digital values from encoders
'A' Interface Board
- EPROM
- ADC (for POT signal)
Position Info (digital) from CPU
1
2
6
Motor Drive signal (analog)
Amplifier
- 6 joints
- 6 motors
- 6 encoders
- 6 POTs
CPU
Digital Servo Boards (with DAC)
Manipulator with servomotors
Visual Display
Teach Pendant
DIGITAL SERVO LOOP
Position information from CPU is compared with that from encoders for motion control

Figure 5: Schematic block diagram of original PUMA interfaced to its controller

The PC based system has been developed in the following manner. The encoders and servomotors of the PUMA have been interfaced to the TRC205 controller built by Mark V Automation Corporation. This controller has amplifiers and digital and analog cards. The controller is interfaced to the 666 MHz Pentium II PC through TRC-S8-2 motion board manufactured by Servo To Go Inc. The TRC-S8-2 motion card is like a PCI card that interfaces various extraneous devices to a PC (like an IBM compatible PC). In this case it interfaces to the TRC205 robot amplifier. It has to be built by Servo To Go Inc. and is not easily available since it is not a standard PCI card (interfacing robot amplifiers is not as common as interfacing serial port devices).

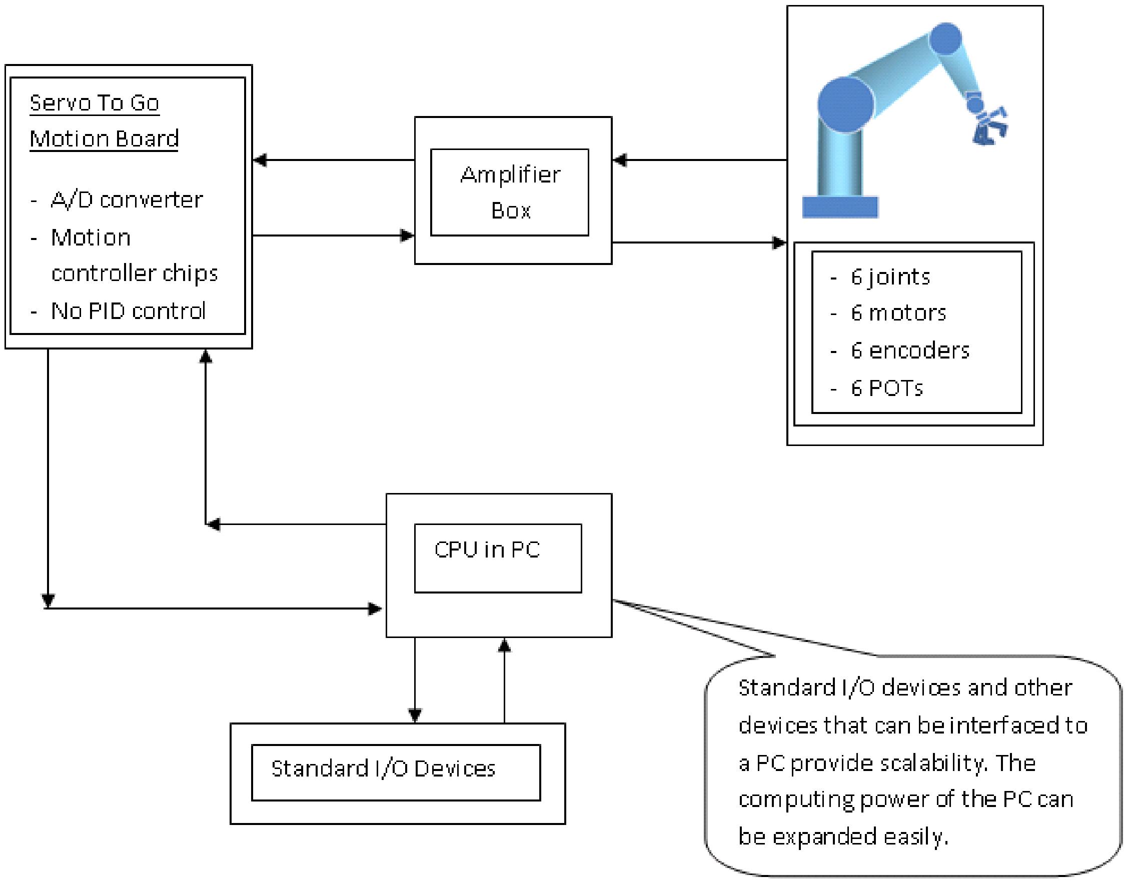

Servo To Go
Motion Board
- A/D converter
- Motion controller chips
- No PID control
Amplifier Box
- 6 joints
- 6 motors
- 6 encoders
- 6 POTs
CPU in PC
Standard I/O Devices
Standard I/O devices and other devices that can be interfaced to a PC provide scalability. The computing power of the PC can be expanded easily.

Figure 6: Schematic block diagram of the PC based PUMA system

Figure 5 and figure 6 give the comparison of the closed and open systems. In figure 5, a schematic control diagram of the PUMA with the original controller from Unimation is shown. The digital servo boards are the central control module and control the motion of the PUMA. They receive the position commanded (in digital form) from the CPU, and also the current position and derived velocity digital values from the encoders on the arm. It compares these and sends the appropriate drive signals in analog form (the digital values are converted to analog values by the DAC – digital to analog converter in the digital servo boards) to the amplifiers that ultimately go to the PUMA motors. Apart from the I/O devices, the user is also limited by the computational power of the CPU.

Figure 6 shows the modified system. Here the central motion controller is the Servo To Go card. It has ADCs, DACs and motion controller chips. However, it has no PID control which has to be implemented in the software that interfaces the Servo To Go card. All the high level information is processed in the CPU and position commands are sent to the Servo To Go card.

After making the PUMA interface to a PC, other devices have been added to make a complete telerobotic system. An attempt had been made by Eduardo to make the whole system PC based so that general software either Microsoft environment based or open source based can be used to program the system. The advantages of using the more common Microsoft software – like MS Visual C++, MS Visual C#, MS Visual Basic – and open source are:

1. They are easily available.
2. Pre-compiled, pre-tested libraries are available which reduce the coding efforts.
3. They are more widely used and so the programmer knows them and this saves time

3.2. Phantom Omni – Haptic Master Device

The next device that we are going to discuss here is the master device, the Phantom Omni haptic feedback device from SensAble Technologies. Like the PUMA it is also a six degree of freedom device and has DH parameters like the PUMA. . Its key features are given below:

1. It has six encoders at its six joints that provide positional sensing in x, y and z directions (by forward kinematics which it does internally)
2. It has potentiometers to determine roll, pitch and yaw of its gimbal
3. Its first three joints have motors that apply torques so that the user feels force
4. It has two buttons on its stylus that can be programmed
5. It can be interfaced to a PC via an IEEE-1394 FireWire port

6. It comes with an OpenHaptics toolkit which is basically a set of APIs in the form of function calls that can be used to query different states of the device
7. It has almost zero impedance (friction and damping) and can be moved very freely in space. This not only makes it easy to use but also gives a true indication of forces delivered to the user. It is a very important property of the Omni.

For other specifications, refer to the Omni data sheet in Appendix A. The master device is thus also interfaced to the PC.

3.3. Sensor and Hand

The sensor used in this project is a SICK laser range finder, DT60-P111B0253 and is interfaced to the serial port of the same PC to which the PUMA is interfaced to. For the data sheet of the laser range finder the user should refer to Appendix B. The laser range finder has the following key features:

1. It has a measurement range of 0.2 to 5.3 m
2. The range is in the form of 4mA to 20mA current output which has to be calibrated

The signal from the laser range finder has to be converted into digital format so that it can be sent to the serial port. The schematic diagram for the circuit that converts the laser signals into RS232 digital format is shown below.

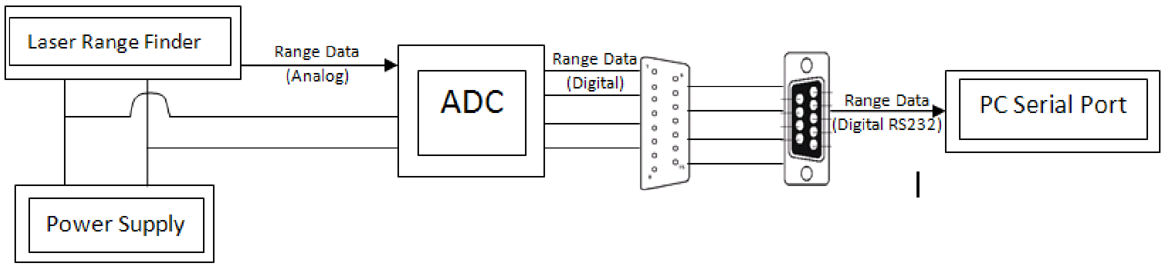


Figure 7: Interfacing of sensor to the PC

Thus we see that the slave robot, the master device and the sensor are all interfaced to the PC. A schematic diagram of the PC-based system is shown in the figure 12.

An electronic hand, called the Barrett Hand from Barrett Technologies is mounted on the PUMA end effector and interfaced to the same PC to which the Omni master is interfaced via a serial port. The commands to control the Barrett Hand are embedded in the same program that is written for the Omni.

3.3.1. Laser Data Acquisition and Processing

The laser used in the telerobotic system, SICK DT60, was highly inaccurate and had poor reproducibility. This led to erroneous computations of the co-ordinates of the various points in the workspace. This in turn impacted the correctness of the generated trajectory and virtual geometries like equation of a flat surface. Ultimately, this all impacted the task execution. Thus, there was a need for recalibrating the laser to determine its characteristics. For better reproducibility there was a need for filtering the data from the laser sensor.

3.3.1.1. Calibration Procedure

The DT60 laser outputs voltages. To determine the distance to an object that the laser is pointing to a relation between the distance and voltage, known as calibration model or sensor characteristics, has to be determined for the range of the laser sensor. For determining the calibration model a set of distances and their corresponding voltages has to be determined. As the laser is connected to the serial port of the PC, the voltage values can be obtained by reading from the serial port. The schematic diagram of the interface between the laser and the PC serial port is shown in the figure 7. The target surface for measuring the distances was taken as a white flat poster board mounted on a stand so that it is always upright. The board is the same board that is used as the table top surface in the pick-and-place task. The laser characteristics may change to some extent depending on the lighting conditions or surface properties of the surface that the laser is pointing to. The reflectivity and opaqueness of a surface affect the laser readings. But since the surface we used for calibration is the surface that is used in task execution, we achieved reasonably good results. For measuring the distances from the target surface another laser distance measurement sensor, DME2000, that was calibrated and had a known accuracy was used. The DME2000 was placed next to the DT60 laser and laser beams from both the sensors were projected onto the target surface. The distance from the DME2000 was a direct read out on the sensor itself which prevented the need to interface it to a computer. In order to ensure that the DT60 was also the same distance away from the target as the DME2000, we made sure that the laser beams emerging from the two sensors were perfectly parallel to each other. The arrangement is shown in the figure 8 below.

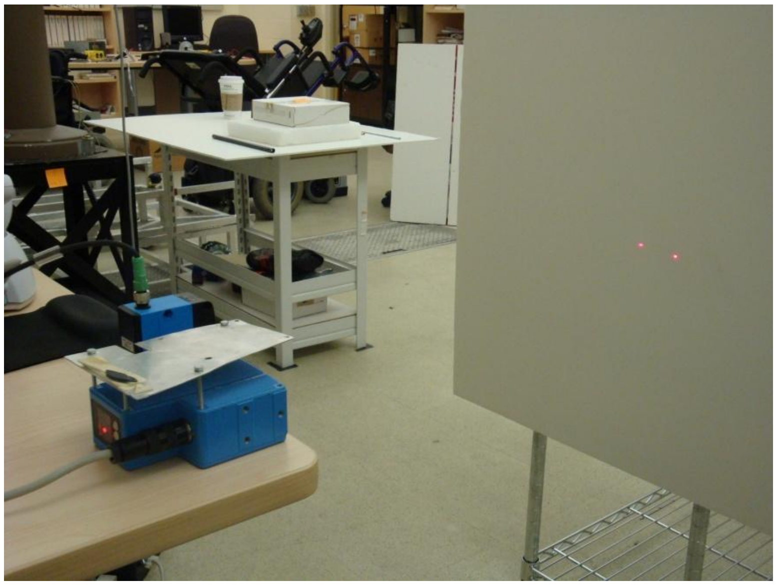


Figure 8: Laser calibration set-up

The distances were varied by moving the target surface backwards, away from the laser sensors. They were moved in steps of 5 cm, approximately. Graduation marks were made on the floor so as to move the stand by the amount. This is depicted in the figure below.

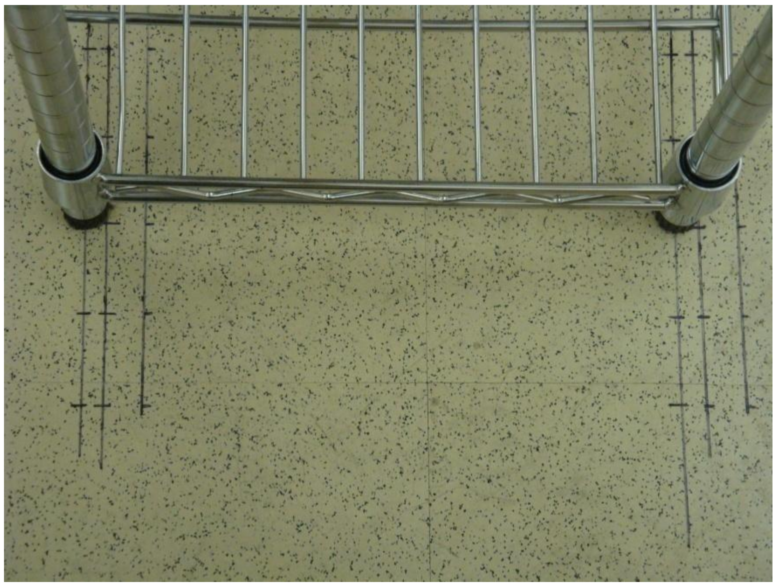


Figure 9: Graduation marks for varying distances during laser calibration

Thus for each distance or for each graduation mark, voltage values were recorded in a file on the PC as they were read from the serial port. The distance being recorded from the direct read out on the DME2000 reference laser. For each distance measured, the voltage value given out as output by the DT60 laser was not constant but varies due to the noise in the system. The variation was around 0.02 volts for each distance measured. Thus in order to get an accurate calibration model, a thousand voltage values were measured for each distance value and the average of the voltages that had a more than one hundred instances per the thousand readings was taken as the voltage for that particular voltage. This was the mean in many cases and very close to the mean in the other cases. Also a pattern similar to normal distribution was observed for the frequency distribution of the voltages. The frequency distribution is shown in the following figure 10 and the final voltage readings for each distance are shown in table 1.

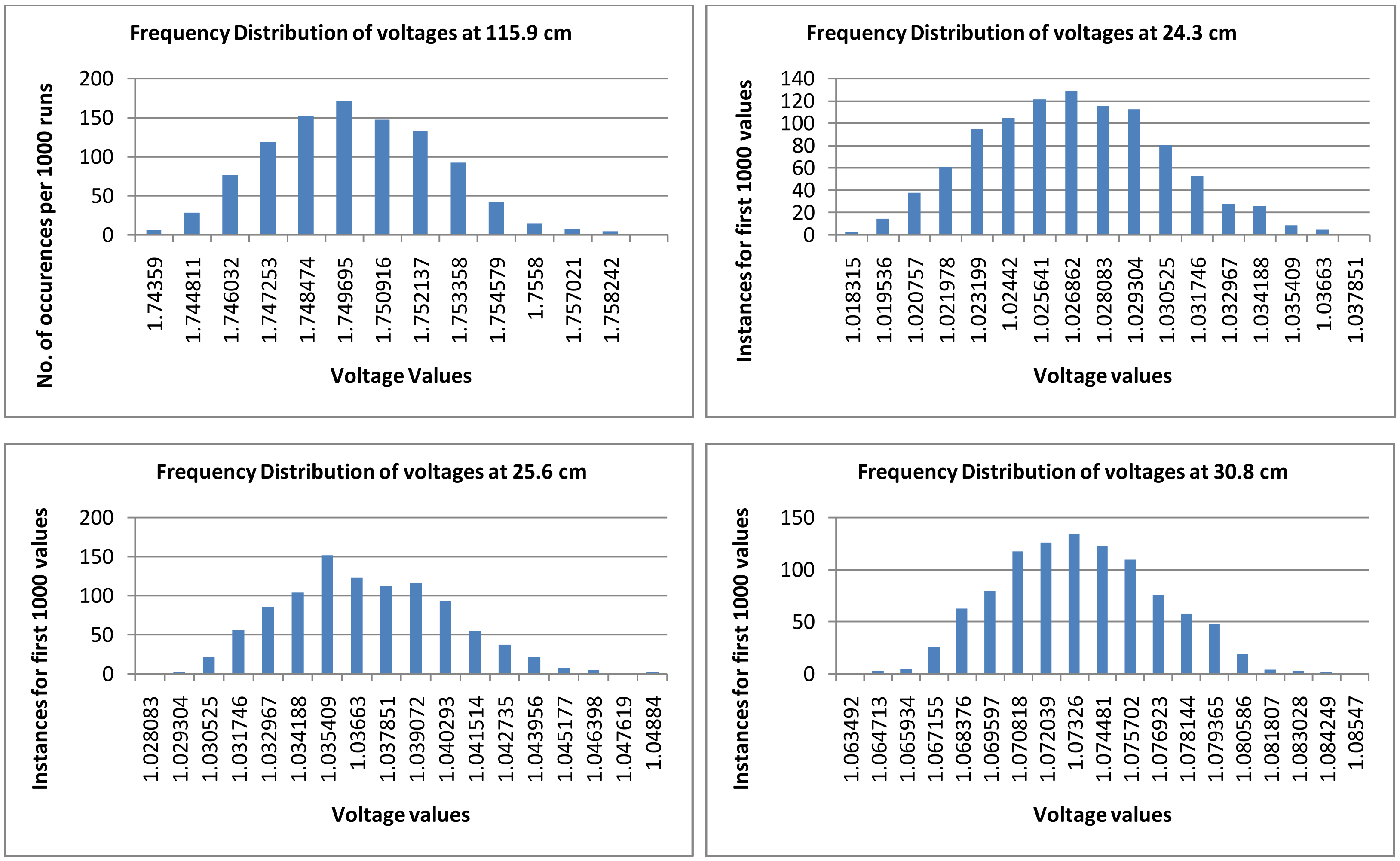


Figure 10: Frequency distribution of voltages during laser calibration

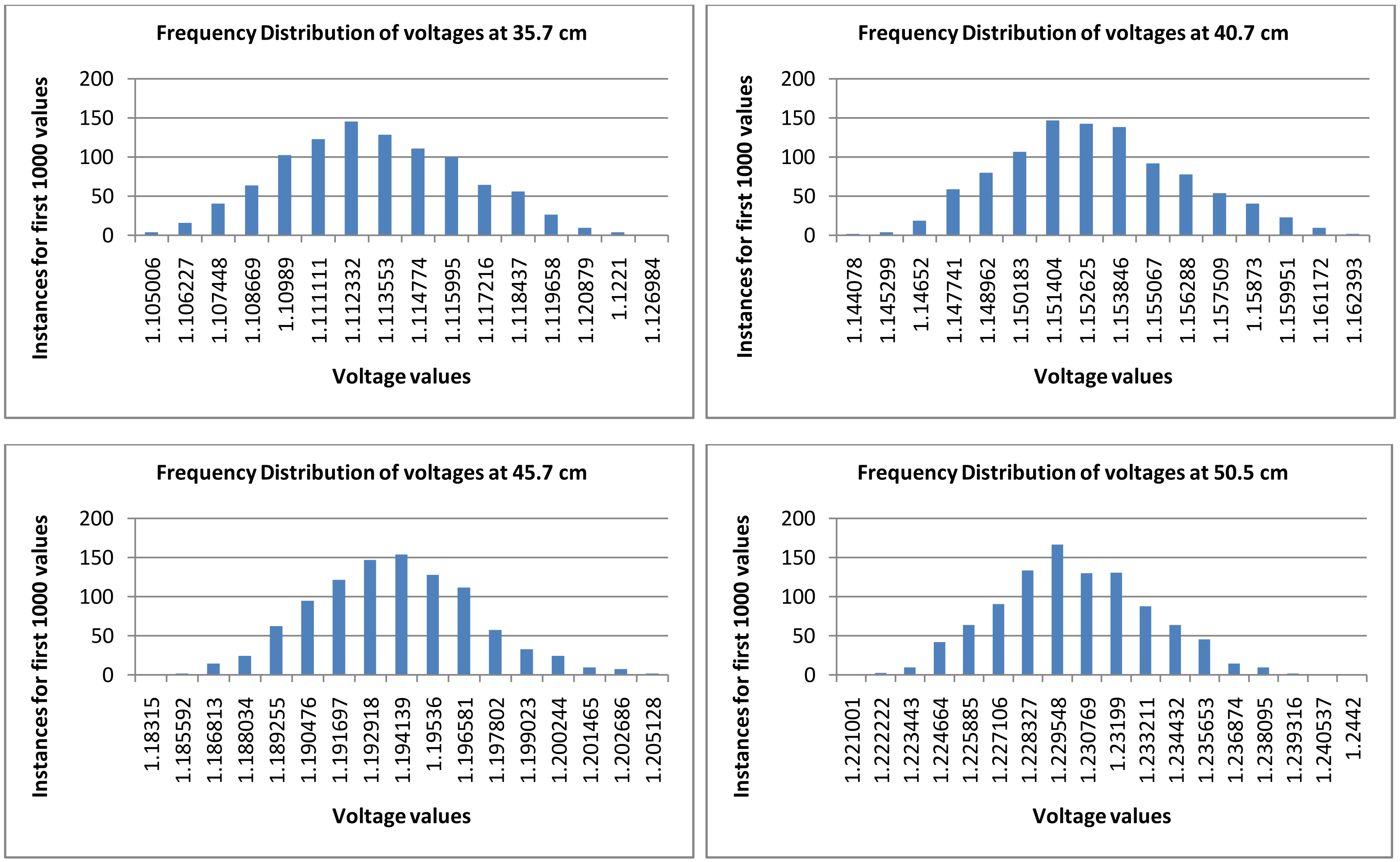
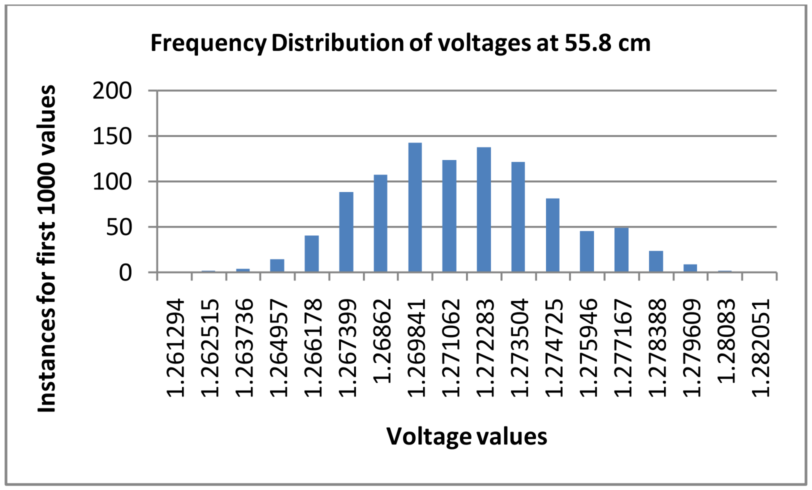
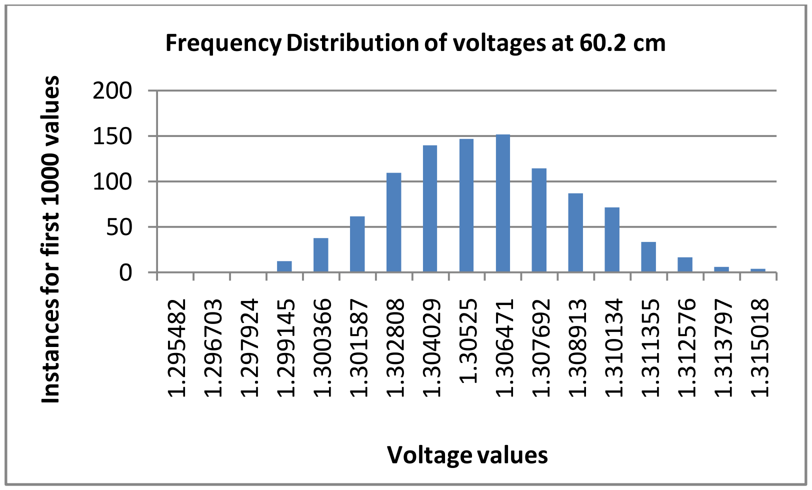
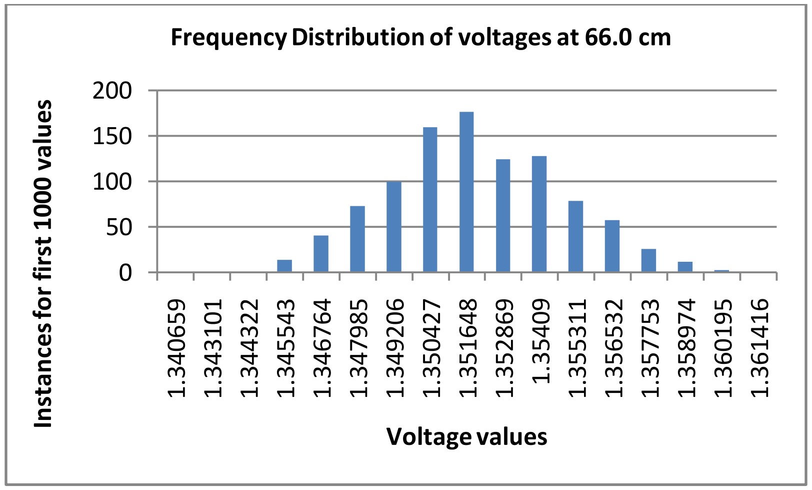


Figure 10 (Continued)







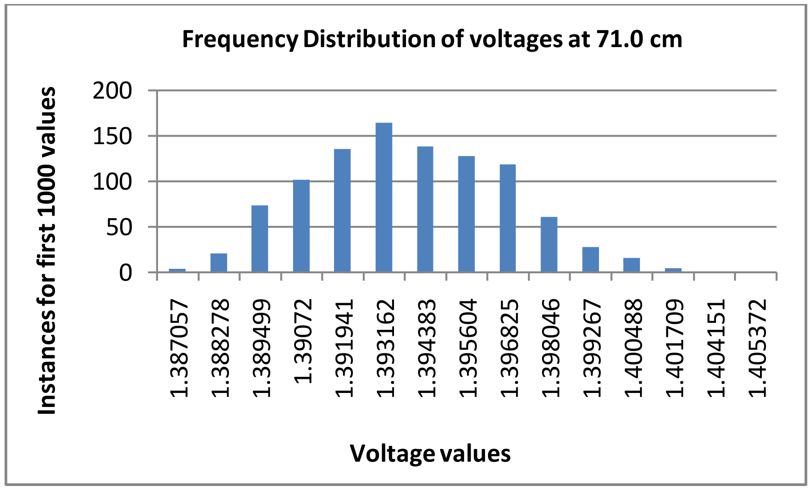


Figure 10 (Continued)

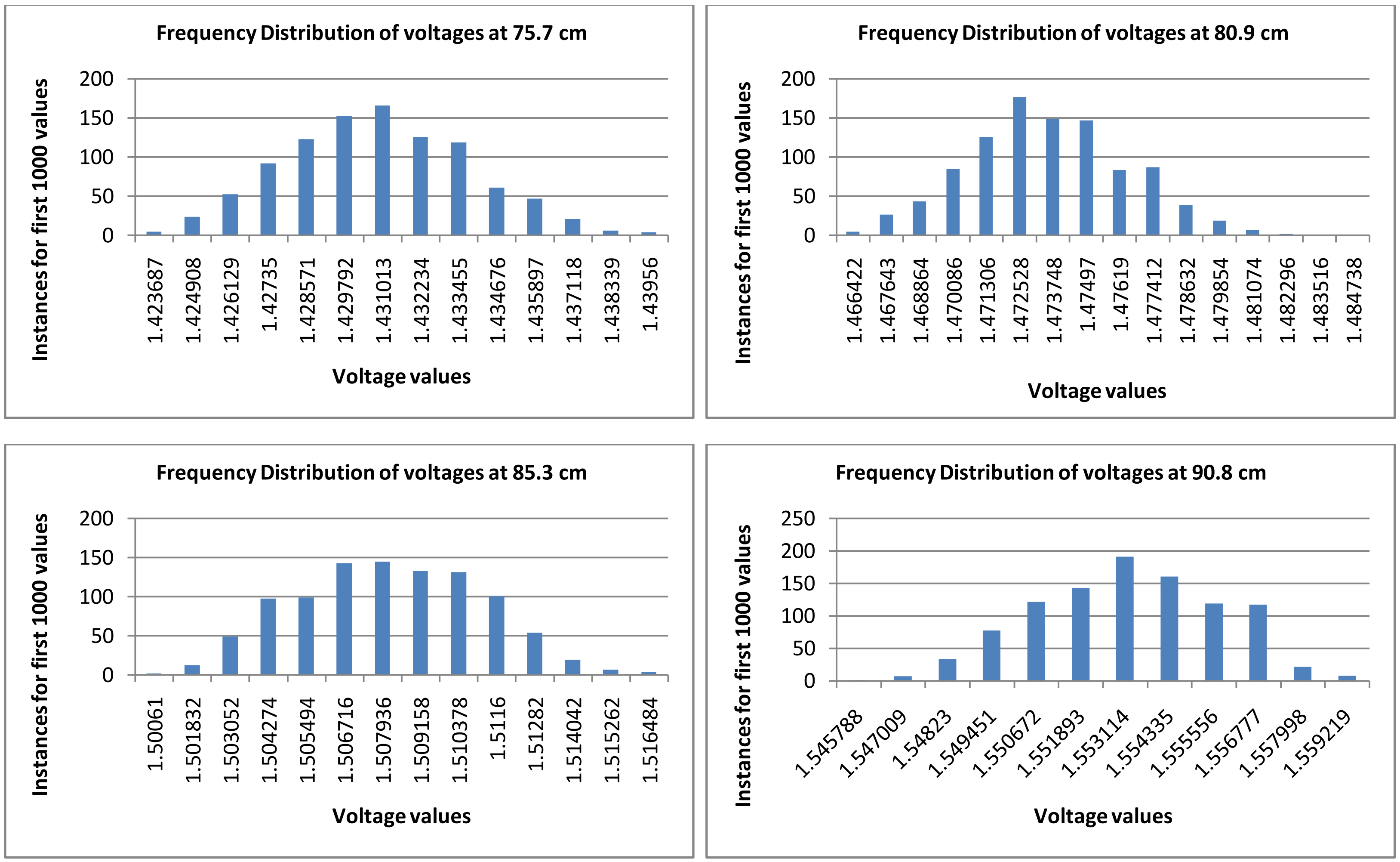


Figure 10 (Continued)

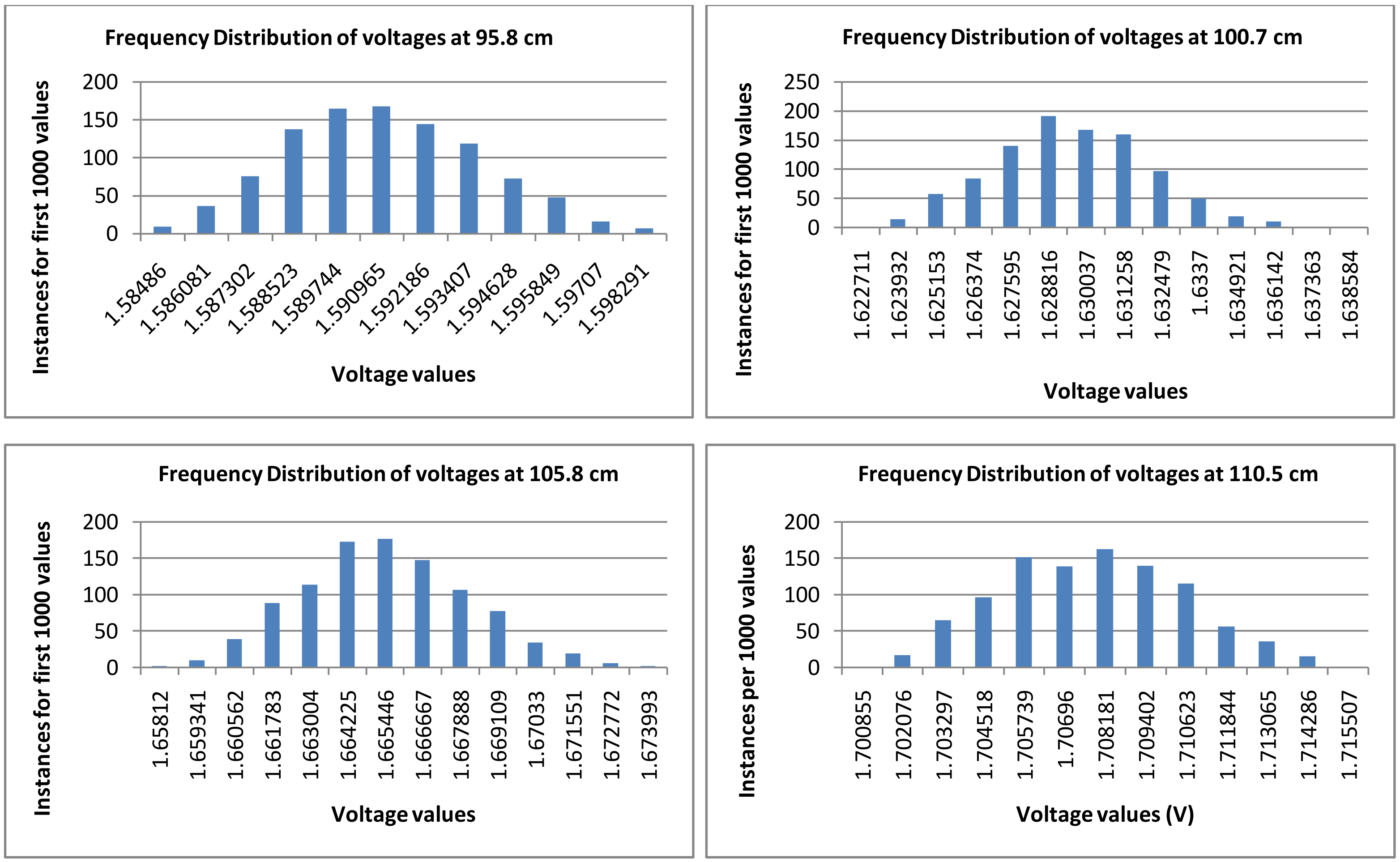


Figure 10 (Continued)

Table 1: Distance and corresponding voltage values for calibration model

Distance (cm)	Voltage (V)
21.3	1.004884
24.3	1.0262515
25.6	1.03663
30.8	1.07326
35.7	1.1129425
40.7	1.152625
45.8	1.194139
50.5	1.229548
55.8	1.271062
60.2	1.30525
66	1.351648
71	1.3937725
75.7	1.431013
80.9	1.473138
85.3	1.508547
90.7	1.5537245
95.8	1.590965
100.7	1.630037
105.2	1.665446
110.6	1.708181
115.9	1.749695

3.3.1.2. Calibration Model

After obtaining the distances and the corresponding voltages, regression analysis was used to generate the calibration model. The manufacturer's specification specified that the calibration model is linear. Hence using Matlab and the function 'regress', a Line of Best Fit' also known as the Least Squares line was generated. The model was in the form of an equation:

$$\text{distance} = (126.1774)^{*}\text{voltage} - 104.8980 \quad (1)$$

The graph of the actual distance and voltage values and the Least Squares line are shown in the figure 11.

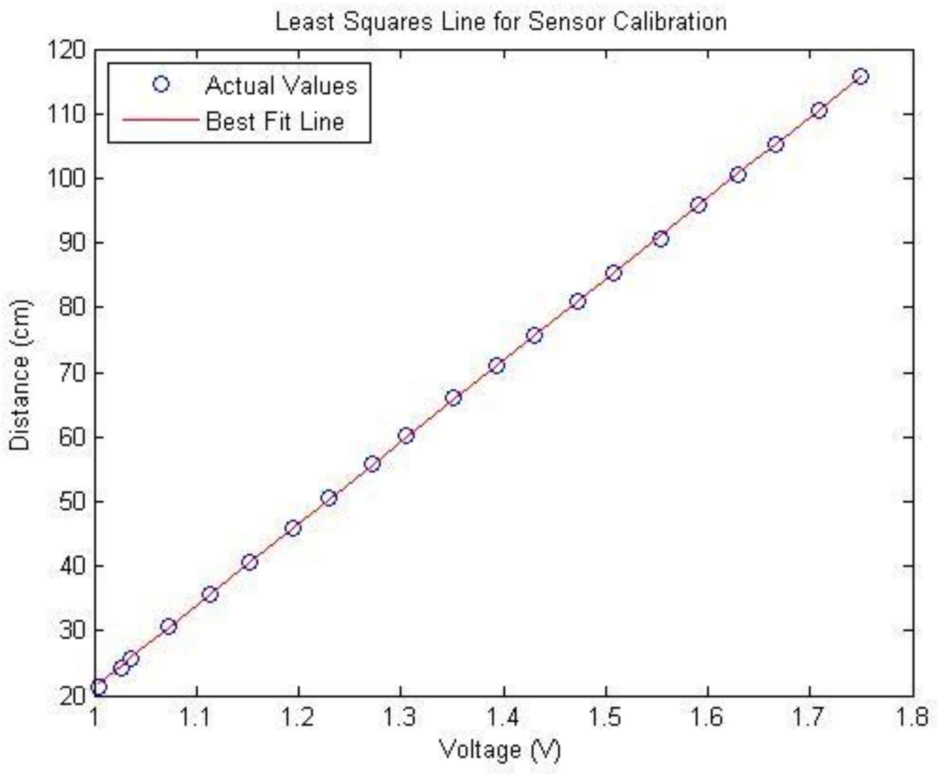


Figure 11: Least Squares line for laser calibration

3.3.1.3. Laser Data Filtering and Interfacing

The variation in voltage values (around 0.2 volts) due to noise in the system lead to the distance values vary by 2.4 cm (min) to 4.2 cm (max) i.e. if the true distance was 47.3 cm then the values obtained ranged from 46.4 cm to 48.8 cm (difference is 2.4 cm) and if the true value of the distance was 24.9 cm then the values obtained ranged from 23.7 cm to 27.9 cm (difference is 4.2 cm). Thus the reproducibility obtained was 4.2 cm (worst case scenario). Even poor reproducibility was obtained with the former calibration model. Thus there was a need for data filtering so that noise could be removed and a better reproducibility could be obtained.

Filtering was achieved by a moving average of 50 points at a rate of around 90 Hz. Initially it was possible to sample only 10 points as the serial port data read frequency was limited to 10 Hz. But with a high speed serial, it was possible to sample at 90 Hz as the read frequency of the serial port was at 90 Hz. This puts a limitation on the user to record co-ordinates of consecutive points at an interval of 0.5 to 0.55 seconds. However, the interval between the time the user ceases to teleoperate and click the Omni stylus button to record points varying from a little less than a second to more than that does not really slow the task execution process. The laser had to be interfaced to the Omni PC so that its data could be processed using C++ language due to the authors' familiarity with C++.

A few problems were encountered with the high speed processing of the laser data. The main problem was the fact that the Omni PC had a number of devices and other PCs interfaced to it and had a single processor. These devices and PCs were interfaced via different ports which had different speeds of reading data. In some cases the devices themselves had a requirement for processing at a certain speed. Moreover the software running on the Omni PC had a different update rate too. Specifically, the PC had Omni master device interfaced through the FireWire port, laser sensor interfaced through a high speed serial port, Barrett Hand interfaced through a low speed serial port and the QNX machine interfaced through the Ethernet port. OpenGL software runs on the same PC. The graphics of the OpenGL has a refresh rate of around 30 Hz whereas that of the haptic device is around 1000 Hz. The high speed serial port reads at a rate of around 90 Hz whereas the low speed serial port reads at a rate of around 10 Hz. The graphics and the haptics thread are synchronized by the OpenHaptics software core and the programmer is at an abstracted layer from this core. So the programmer need not bother about the synchronization of these threads. But the programmer does need to take care about the synchronization for the data processing from other resources. For example the Barrett Hand interfaced to the 10 Hz serial port is able to operate satisfactorily when it runs with the OpenGL 30 Hz loop but the laser serial port is unable to work at that speed as it has a much faster update rate and its buffer gets accumulated with data which ultimately results in the system hanging itself. Thus a single processor is unable to run multiple modules at different rates. A possible solution to this problem is to make multiple threads one for each module that is not able to synchronize with the rest of the modules. So, the laser data read module from the high speed serial port was run in a separate thread that was different from the one that was meant for the graphics, haptics and Barrett Hand control. A thread for receiving data from QNX machine was a separate thread already in place by Veras [14] and that for sending the data to the QNX machine was the same as the one meant for the graphics, haptics and Barrett Hand control. The threads were Windows threads. This enabled the OS to schedule and to synchronize the various modules on the processor so that they run in a stable manner. Each thread gets access to the processor based on the scheduling policy of the OS.

The new improved calibration model and filtering method resulted in a better accuracy for the laser and better a better reproducibility. The reproducibility improved from 4.2 cm (worst case scenario) to 0.5 cm (worst case scenario). This further led to more accurate task execution and improved repeatability of the robotic system. The improvement of the laser accuracy and reproducibility over the previous scheme and its impact on the robot task execution are explained in the chapter 7.

3.4. Hardware Integration

Figure 12 shows the schematic block diagram of the hardware integration of the telerobotic system. It also shows the flow of information between the different modules of the system.

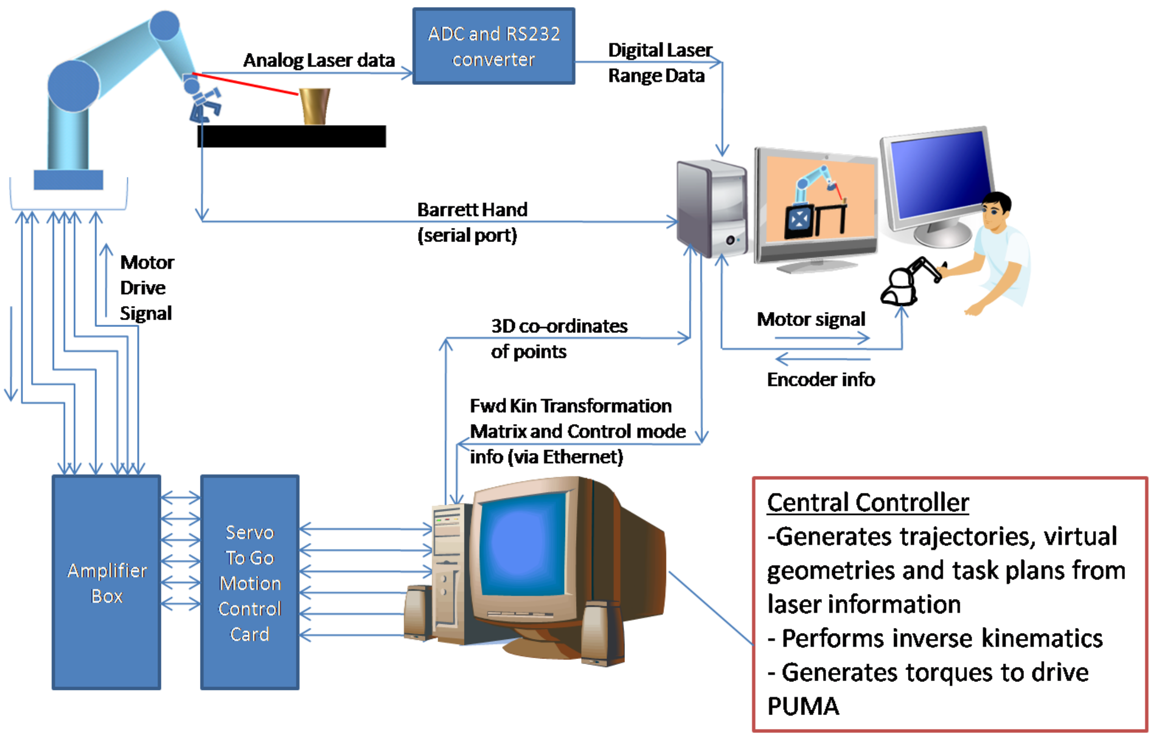

Analog Laser data
ADC and RS232 converter
Digital Laser Range Data
Barrett Hand (serial port)
Motor Drive Signal
3D co-ordinates of points
Motor signal
Encoder info
Fwd Kin Transformation Matrix and Control mode info (via Ethernet)
Amplifier Box
Servo To Go Motion Control Card
Central Controller
-Generates trajectories, virtual geometries and task plans from laser information
- Performs inverse kinematics
- Generates torques to drive PUMA

Figure 12: Integration and information flow in the telerobotic system

Figure 13 shows the various PCs involved in the telerobotic system and their configurations.

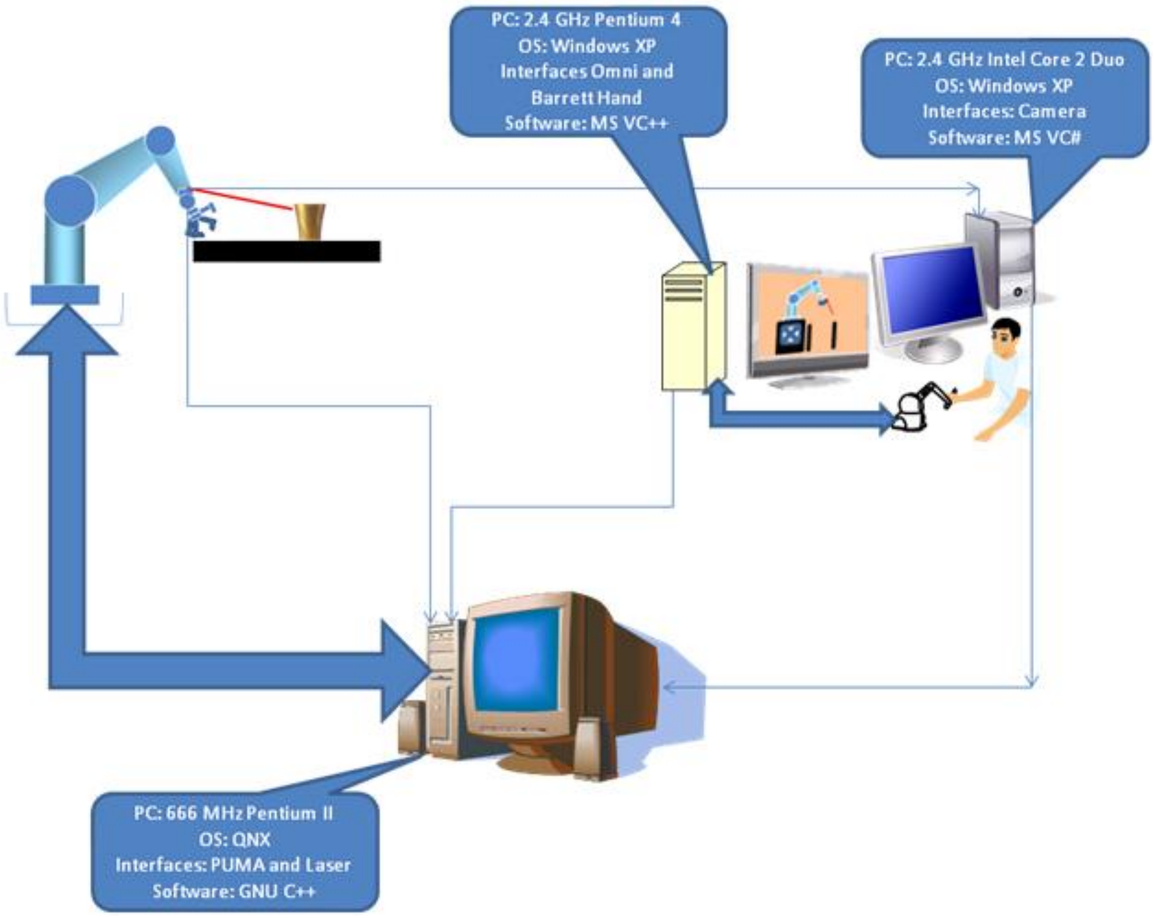


Figure 13: PC configuration in the telerobotic system

3.5. Software Architecture

In this sub-section we describe the software architecture of the telerobotic system.

3.5.1. A Real-Time System

Robotics is a real world application that requires a real-time control. There are several definitions of real-time systems some of which are given here. One defines real-time operating systems as, “A real-time operating system (RTOS) is able to execute all of its tasks without violating *specified* timing constraints." Another definition says, "Times at which tasks will execute can be *predicted deterministically* on the basis of knowledge about the system's hardware and software." The second definition means, if the hardware *can* do the job, the RTOS software *will* do the job deterministically.

In a real-time system not only the correctness of the result is critical but the time at which the result is generated is also critical. It does not necessarily mean that the response should be *fast* but it should follow the timing strictly, that is, the result should be generated or the system should respond in a certain manner and the timing has to be deterministic and accurate. Real-time systems are classified as 'hard' and 'soft' real-time systems. Hard real-time systems are those in which if the timing requirements are not met then it could result in a catastrophic failure of the system. In hard real-time systems the deadline must be met regardless of the system loads. Soft real-time systems are those in which if the timing requirements are not met then it causes inconvenience and possibly an increased cost over time but the failure is not catastrophic. An example of a soft real-time system is that of opening an audio or video file which comes at a delay of about a few milliseconds or may be a few seconds and this does not impact the system or the user adversely although this may cause a degree of inconvenience to the user. However, steering a space probe that is moving at a speed of several kilometers per second or controlling a surgical robot is strictly hard real-time as delay in actuating the space probe steering system can cause deviations in its orbit and this can lead to erroneous atmosphere entry situations that can be disastrous or a delay in the movement of a surgical robot can cause undesired movement that can cause serious injury to the patient and even a fatal injury in certain cases.

An RTOS is necessary for creating a real-time application as it manages the various functions like task scheduling, synchronization, IPC (inter process communication), interrupt servicing and so on which are explained in the following section. QNX, a real-time operating system that provides the framework for building real-time applications and also manages functions listed above, is the RTOS used in our system.

Real-time OS are similar to general purpose OS as their functions are similar to the general purpose OS. However, they execute these functions differently as they have different goals. Following are some the main functions of an OS and how they differ for different OS. Other than these RTOS are leaner in that they do not deal with functions that a general purpose OS has to deal with like file handling, graphical user interface provision, disk I/O and so on.

1. Task management and scheduling: At a time, there are several tasks running which need CPU time and memory allocation for getting executed. The OS manages the resource allocation to tasks. A general purpose OS would target maximum average throughput whereas an RTOS would target deterministic behavior for which it may have a scheduling algorithm based on a criteria like highest priority first, earliest deadline first and so on.
2. Interrupt servicing: Interrupts are asynchronous requests by hardware (hardware interrupt) or software (software interrupt) to the OS to gain access to CPU for servicing the interrupts i.e. perform the service that the device wants. When the OS receives an interrupt request it jumps to a specific address in the memory where the ISR (interrupt service routine) is. The ISR is executed by the CPU however the job of the OS is to associate memory addresses with specific interrupts and to determine as to when the interrupt needs to be serviced. In RTOS the interrupts generally have to be serviced immediately and have to be deterministic in that sense.
3. Communication and synchronization: IPC (inter process communication) is communication between different processes or tasks and it gets tricky when these tasks exchange information that can be modified in any of these tasks. It is vital to synchronize this communication in these cases or else it can lead to data corruption and eventually undesired or unpredictable results. RTOS has to make sure that the synchronization happens in a more deterministic way and uses mutex (mutual exclusion locks) and semaphore signals to do the job.

a. In an application like robotics there can be communication not only within the processes running on the same computer but also between external processes on different computers or peripheral hardware (via analog I/O cards). Here again communication and synchronization is important as there is sharing of information that is manipulated in each of these processes.

4. Memory Management: This function of the OS is important for memory mapped I/O in which different tasks would like to have access to the available memory.

APIs or application programming interfaces are the functions that call various OS routines. Application programs are written over these APIs or rather APIs are embedded in application programs (as in our case). We have made use of the POSIX threads API to implement multi threading in our control algorithm. The POSIX functions interact with the QNX kernel to create threads, delete threads and to manage various functions like IPC among threads, thread synchronization and so on. POSIX are the IEEE standard for APIs for UNIX-like OS, QNX being one of them. However, POSIX is not compliant with all the OS. The following figure shows the interfacing of the RTOS with the hardware and the application software.

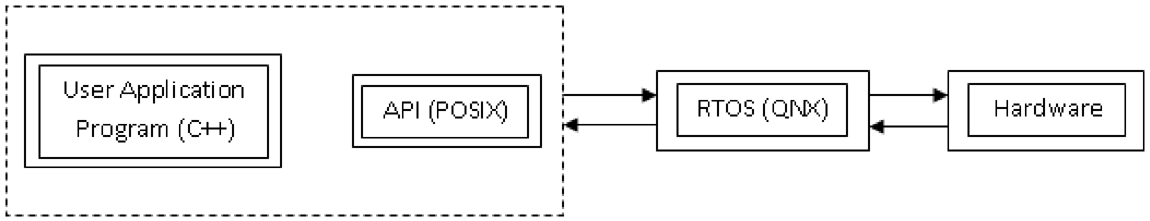


Figure 14: Interfacing of RTOS with hardware and user application software

3.5.2. Multithreaded Architecture

As mentioned POSIX standards for threads (extension 1c for threads programming) is used to create multiple threads that QNX would manage. Multiple threads are important as there are many tasks that have to run simultaneously. For example, the motors of the PUMA have to be actuated at regular intervals (servo rate which is 1000 Hz) and the sensor data (encoder angles) have to be read at the same time to update the trajectory otherwise PUMA will start deviating from the desired trajectory. There are threads implemented on the QNX machine and are listed below. The ‘main application routine’ calls the function ‘start thread’ that creates and starts these other threads.

1. Semaphore thread: The thread for synchronous communication (priority = highest)
2. Torque generation thread: It is called the ‘PositionsThread’ in the control code. This thread computes the torques using the computed joint angles (from the ‘trajectory generation thread’), joint velocities and appropriate gains. It also implements the PD+G (proportional, derivative plus gravity compensation) controller. The computed torque values are then sent to the PUMA amplifiers. (priority = highest - 1)
3. Trajectory generation thread: It is known as ‘STGReaderThread’ in the control code. This thread computes the trajectories and joint angles for automatic mode and joint angles for teleoperation and other modes based on the laser data. It implements the Resolved Rate inverse kinematics and Equivalent Angle-Axis method for trajectory generation. It sends the computed joint angles to the torque generation thread to drive the PUMA. (priority = highest - 1)
4. Sensor data acquisition thread: It is known as ‘GetSensorDataThread’ in the control code. This thread receives the vision (which is co-ordinates of the object centeroid that is in the field of view of the camera in the form of two floating point numbers) and laser data via sockets. (priority = highest - 10). Currently this thread is not used as the information from the camera PC is not used. Also there is no laser data coming in as the laser is now interfaced to the Omni PC.
5. Omni data acquisition thread: It is known as ‘GetHapticDataThread’ in the control code. It receives the nine members of the transformation matrix of the Omni tip via sockets, the keyboard key pressed from the Omni keyboard that decides the mode,

the Omni stylus button information and the Barrett Hand status. (priority = highest - 10)

6. PUMA data transmission thread: It is known as ‘SendPumaDataThread’ in the control code. It sends the (x, y, z) PUMA co-ordinates of the initial and final point in the trajectory to the Omni PC controller via sockets. (priority = highest - 10)

7. Keyboard data acquisition thread: It is known as ‘KeyBDThread’ in the control code. It monitors the key pressed on the PUMA PC keyboard. (priority = highest - 4)

Although the exchange of data between different threads does not create race conditions (as data written to a variable in one thread is not written in any other thread and is only read in other threads) the multithreaded framework using POSIX helps to scale the system in future when more sensors and/or robots are to be integrated into the system or new types of control require data to be exchanged with race conditions arising. Synchronization mechanisms like mutexes and condition variables can be introduced under such circumstances. Once the need arises and is implemented, the framework will enable threads to run concurrently without data corruption and conflict in accessing variables. Also scheduling algorithms (like highest priority first) can be implemented as higher level robotic tasks are integrated (coded) into the system. Thus the framework for real-time control created with QNX and PIOSIX API is completely scalable with the ability to accommodate various devices as well as complex control algorithms in the future.

3.5.3. Omni Software Architecture

The Omni is a haptic interface device that is primarily developed for human interaction with a virtual environment and the device administers feedback forces to the user as the user interacts with the virtual environment. As the device provides a feedback to the user it is based on closed loop control system and has servo loops running at its software core. As it has constantly changing graphics (as the user interacts with the virtual environment) and constantly changing forces (again from the user's interaction with the virtual environment) and since the graphics and the forces loops need different refresh rates (the graphics loop needs a refresh rate of at least 30 Hz for the user to feel a continuous image and the haptics loop needs to be refreshed at, at least 1000 Hz for the user to feel the forces with reasonably good fidelity), it has a multithreaded architecture in which multiple threads of code run concurrently with synchronized communication and some sort of scheduling. For synchronized communication while sharing common data, data snapshots instead of mutexes are used. This is because if one thread, say the graphics thread locks a variable then the haptics thread, a higher priority thread cannot wait for the lock to be released as it needs to be refreshed at every 1000 Hz for the user not to feel any loss of forces and to feel the forces continuously. Therefore a snapshot of the common data is taken by the SDK core and is available to both the threads simultaneously. How the data is updated and by which threads, the graphics or the haptics, and if an updated data is to be made available to the other thread when the other thread is still in the middle of processing is something not answered by the OpenHaptics user manual [23]. This software core is abstracted from the user is a layer between the application program and the OS. The user can modify certain parameters to modify her interaction with the virtual environment using APIs that are part of the OpenHaptics toolkit. However the user has no access to the software core to the level so as to change the priorities of threads or communication and synchronization of threads. This was primarily undertaken by SensAble technologies to make the device usable even with users having minimal experience with programming and to keep the device safe for users (as it administers forces and improper usage of device can be harmful to user. However, this also makes programming with the device a challenging task as the syntax of APIs is complicated and structuring the code with APIs is difficult as well. A better job with documentation from SensAble technologies would have made life easier for programmers. Also, though the API toolkit is known as OpenHaptics, it is completely proprietary and is not open source at all. Other than these shortcomings Omni from SensAble Technologies is the first of its kind device and is excellent for applications that involve manipulation with virtual environment with or without force feedback. It can also be used for manipulating actual environments which is part of this telerobotics project jointly accomplished by Eduardo Veras and the author under supervision of Dr. Rajiv Dubey.

OpenHaptics toolkit is thus the set of APIs that users can use to control the Omni by interfacing with the software core. The software core further calls the OS functions (in this case Microsoft Windows XP) that are necessary to execute the APIs. Figure 15 shows the interfacing of Omni software core with OpenHaptics API, Omni controller and the operating system.

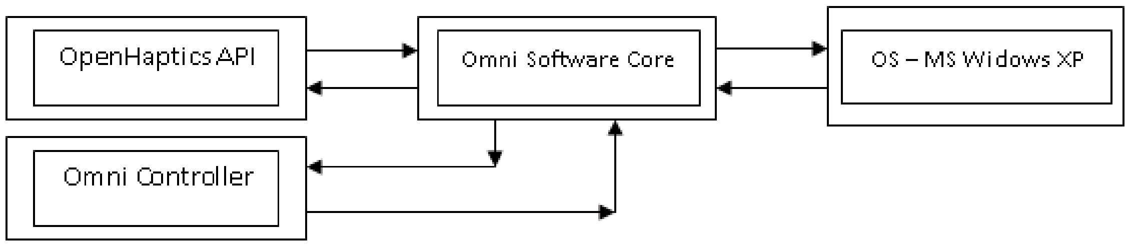


Figure 15: Interfacing of Omni software core with OpenHaptics

The APIs are of two major types. One is the HLAPI (haptic library application programming interface) and HDAPI (haptic device application programming interface). HLAPI is built over HDAPI in the sense that when a programmer calls an HLAPI functions, she is internally invoking an HDAPI routine or set of routines which then further invoke OpenHaptics core or the OS depending on the function call. Thus HLAPI is a higher level programming interface and is suitable for programmers who are not expert in programming but are familiar with basic programming and know how to use OpenGL APIs. Programmers can simply define shapes for the virtual environment and assign materials to the shapes and the software core generates the forces automatically based on material properties. With HDAPI the programmers can interact directly with the device, send forces to the device, configure runtime behavior of drivers and have access to parameters as low as motor DAC and encoder values. But they may have to deal with issues like thread safety and using efficient force rendering/collision detection algorithms and data structures etc. HLAPI is more suitable for developers who want to quickly integrate haptics into their existing applications whereas HDAPI is more suitable for those who want to directly interact with the device and use haptics in the areas of telepresence, remote manipulation etc.

3.5.3.1. HDAPI Program Structure

The HDAPI based program consists of initializing the device, initializing the scheduler callback functions (which are functions repeatedly called at every servo loop execution), enable device forces, start the scheduler and perform clean up routines when the application is terminated. The force effects that the user should feel at the Omni are defined in the scheduler callbacks. For a simple application of repelling the Omni tip from a plane when the Omni tries to penetrate the plane, following steps are performed:

1. Initialize the device.
2. Create a scheduler callback that queries the device position and commands a force away from the plane if the device penetrates the plane (commands that set the state of the device should be within haptic frames and only one frame per scheduler tick is allowed).
3. Enable device forces.
4. Start the scheduler.
5. Clean up the device and the scheduler when the application is terminated.

The scheduler is a high priority thread that runs at the servo loop refresh rate of 1000 Hz and makes possible the scheduling and communication of the scheduler callbacks with the servo loop for setting the state of the device i.e. administering forces.

The scheduler callback is an interface to communicate with the servo loop in a thread safe manner. Operations like sending forces that are to be performed in the servo loop are entered through callbacks. The scheduler callback runs only once at the first servo loop tick or runs at every servo loop tick depending on the return value of the callback i.e. HD_CALLBACK_DONE or HD_CALLBACK_CONTINUE.

The main API function calls that are entered in the callbacks are related to getting the state of the device (information about position, velocities, endpoint transformation matrices, buttons press status etc.) and setting the state of the device (setting forces). Commands for setting the state should be entered strictly inside HD frames within the scheduler callbacks. Commands for querying the state could be outside the haptic frames but they will return the device state of the previous frame, which is the most current frame. A frame thus defines the scope within which the state of the device is guaranteed to be consistent and the modifications to the device state are thread-safe. It is bracketed by hdBeginFrame and hdEndFrame statements. At the beginning of the frame the device state is updated and stored so that all queries to the device state within the frame reflect a snapshot of the data. At the end of the frame new state is written to the device. The prototype for callbacks is shown in figure 16:

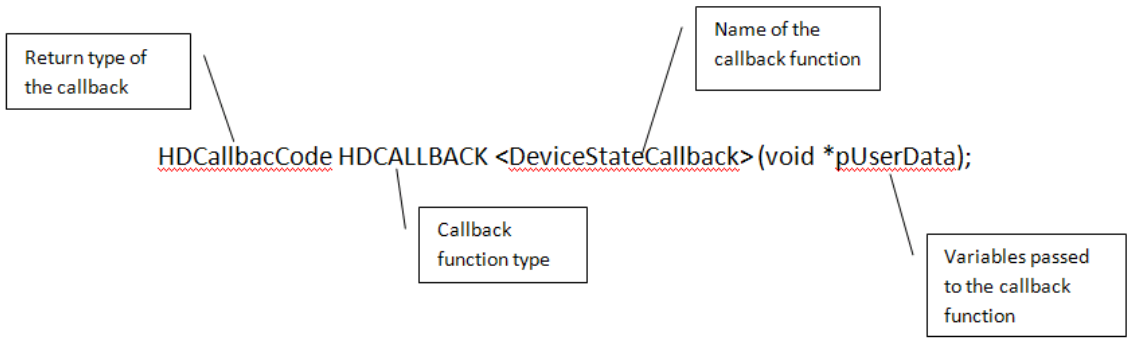

```
HDCallbacCode HDCALLBACK <DeviceStateCallback> (void *pUserData);
```
Return type of the callback
Name of the callback function
Callback function type
Variables passed to the callback function

Figure 16: HDAPI callback prototype

The return types are HD_CALLBACK_DONE and HD_CALLBACK_CONTINUE. Scheduler calls are of two types, (i) synchronous calls and (ii) asynchronous calls. Synchronous calls return to the main application program only after the callback function is complete. Asynchronous calls return immediately after being scheduled. Synchronous calls are primarily used to get a snapshot of the state of the device for the application whereas asynchronous calls are used to set the device state. The syntax of the two calls is shown in the following figures:

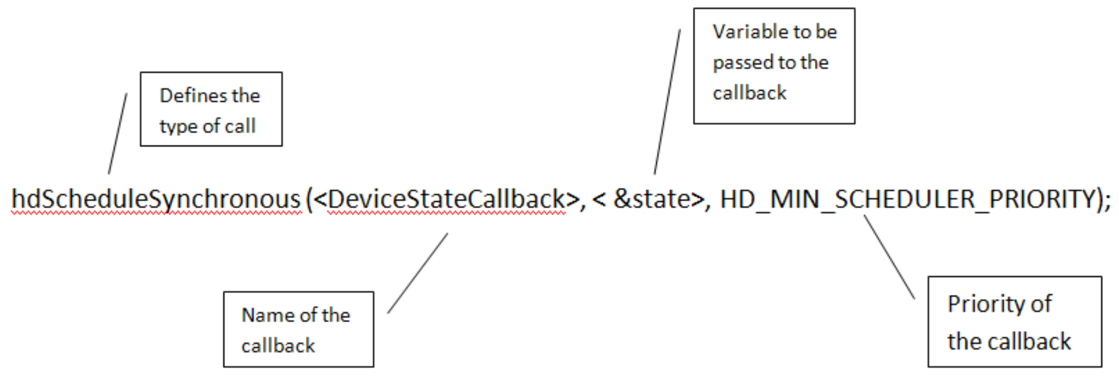

Defines the type of call
Variable to be passed to the callback
hdScheduleSynchronous (<DeviceStateCallback>, < &state>, HD_MIN_SCHEDULER_PRIORITY);
Name of the callback
Priority of the callback

Figure 17: Syntax of synchronous call

```
hdScheduleAsynchronous (<AForceSettingCallback>, <(void*)0>, HD_DEFAULT_SCHEDULER_PRIORITY);
```

Figure 18: Syntax of asynchronous call

There can be many callbacks scheduled and the order of run of the callbacks depends on the priority of the callbacks, the highest priority callback running first and those with same priority run in any order.

3.5.3.2. HLAPI Program Structure

Is similar to OpenGL APIs and does haptic rendering to the haptic device similar to how OpenGL renders graphics to a monitor. The user specifies the shape and material properties of the virtual objects and the engine automatically computes the forces and sends it to the haptic device. Unlike HDAPI the user need not bother about force computations and managing scheduler calls priorities. The engine does it for the user.

There are three ways to generate forces using the haptic device. One is using the shapes drawn using OpenGL and rendering the shapes haptically using HLAPI commands, the second is to generate standard force effects like friction and viscosity as well as custom forces and the third is to define a POSE and the haptic device generates the necessary forces to reach that POSE.

Just like in HDAPI, in HLAPI also all the commands for haptic rendering must be inside a haptics frame bracketed by hlBeginFrame and hlEndFrame. This allows the engine to properly synchronize changes to device state between multiple threads. hlBeginFrame creates a snapshot of the state and hlEndFrame flushes the changes to the monitor and the device. Also the graphics and the haptics refresh rates in HLAPI are the same unlike in HDAPI.

3.6. User Interfaces

Various user interfaces that were developed by Veras [14] were used in this project. Veras had developed a virtual environment interface, a PUMA 3D solid model interface, a keyboard interface and had integrated the Omni master as a haptic interface. Veras was successful in using all the interfaces in his work except the PUMA 3D model interface due to communication issues of the Matlab software, in which the 3D model was running using the Virtual Reality toolbox of Matlab, with the serial port, though which the Omni tip transform information and commands from the Omni PC were coming.

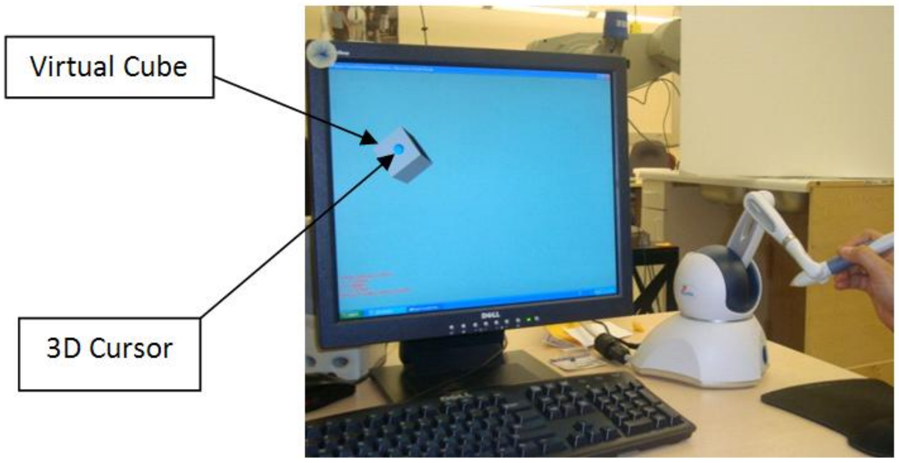


Figure 19: 3D virtual environment interface using OpenGL in MSVC++

The virtual environment interface was developed by Veras using OpenGL APIs in Microsoft Visual C++ program, the same program that controls the Omni and the communication between Omni PC and QNX machine. It consists of a cube and a sphere, the sphere representing the Omni tip and known as 3D cursor. The virtual environment is sized to the Omni workspace. Figure 19 shows the virtual environment. The cube and the 3D cursor are used for engaging the Omni and the PUMA and for indexing. To engage the PUMA, the user moves the 3D cursor and hence the Omni so that the 3D cursor touches the cube. The user knows this has happened when the user feels a force as the 3D cursor touches the cube. Then the user presses the blue button on the Omni stylus while maintaining contact with the cube (sort of a click-and-drag method) and the PUMA can be teleoperated. If either the contact of the Omni tip (or the 3D cursor) is lost or the blue Omni stylus button is not pressed the PUMA will not be engaged and will not be teleoperated. The workspace of the Omni being smaller than that of the PUMA, the Omni has to be brought to the center of its workspace every time its limit is reached. This method is called Indexing. When the 3D cursor along with the cube reach the edge of the Omni workspace the white button on the Omni stylus can be pressed while maintaining the contact of the 3D cursor with the cube to bring the cube to the screen center. The PUMA is disengaged at this instant and the 3D cursor has to be taken to the origin and the process of engaging the PUMA has to be repeated.

The keyboard interface is basically used for sending the commands to the QNX machine. Specific keys are assigned specific functions like ‘A’ key for autonomous trajectory generation and execution, ‘T’ key for autonomous hand alignment with the flat surface and so on. The function of each key is explained in the Chapter 6 where the specific key is used to execute a specific laser based feature. The keyboard information is read in the same C++ program that controls the Omni and is sent to the QNX machine through the data structure ‘gOmni’ and the C-socket ‘pQNXSock’. The basic data structure and communication was set up by Veras [14] however for the laser based features implemented in this thesis work, the data structure was extended to include the new keyboard keys.

Another interface that the user interacts with is the master device itself which provides a haptic interface to the user. The haptic interface is used while engaging the Omni with the PUMA as explained in the 3D Virtual Interface above and also for generating force feedback along a trajectory which is explained in detail in chapter 6, section 6.2.3.

Chapter 4: Kinematics of Manipulators

This chapter forms the basic building blocks of the concepts that will be used in the chapter 6 to implement the kinematic algorithms for laser based robotic control. We introduce the reader to the concepts of manipulator forward kinematics and inverse kinematics. The Resolved Rate algorithm that is used for inverse kinematics is presented here.

4.1. Introduction

Kinematics is the science of the motion of a manipulator without regard to the forces that cause this motion. It is the study of position, velocity and acceleration (all higher order derivatives of position with respect to time or some other variable). The basic aim of the study of the kinematics of a manipulator is that the position and orientation, also known as POSE, of the end effector (and that of the other joints from the base of the manipulator to its end effector) with respect to its base should be defined at all times as the manipulator moves in its workspace. The process of determining the POSE of the end effector with respect to the base is known as forward manipulator kinematics or simply forward kinematics. The knowledge of the POSE of the end effector of the manipulator with respect to the base serves two purposes. First, it determines where the manipulator (and its various links and joints) is at a given time which further helps to determine whether the manipulator is within its workspace or is out of its limits. Secondly, it helps to determine the position and orientation of other objects (targets and obstacles) in the environment. This is useful in applications like autonomous control of the manipulator which involves automatic trajectory generation and obstacle avoidance. The position and orientation of the other objects in the environment can be determined with the help of sensors mounted on the end effector of the manipulator and provided the POSE of the end effector is known. The study of kinematics also serves another purpose. It helps to determine the joint angles (or joint rates) at a point in time when the end effector POSE is known. The process is known as inverse manipulator kinematics or simply inverse kinematics. Joint angles could be determined by querying the encoders at that point in time however for motion planning we need to determine a set of joint angles for a trajectory specified in terms of end effector co-ordinates which is where inverse kinematics kicks in.

The forward kinematics is carried out by the following procedure. Cartesian co-ordinate frames are affixed to each joint of the manipulator including the base. Then a relationship of each frame (joint) with respect to its previous frame (previous joint) is determined in terms of joint angles. This relationship determines the POSE of each frame with respect to its previous frame. This way as we go from the end effector to the base of the manipulator joint by joint, we establish a relationship of the end effector with respect to the base of the manipulator in terms of joint angles. Thus as the manipulator articulates the POSE of each joint and that of the end effector with respect to any of its previous joints or the base is known at any point in time by querying the joint angles from the encoders at that instant. The inverse kinematics is carried out by solving the relationship between the end effector and the base developed in the formulation of forward kinematics for joint angles.

There are two manipulators, also known as robotic arms used in the telerobotic set up. One is the master device, the Phantom Omni from SensAble Technologies. This master device has 6 degrees of position feedback to the system by means of encoders installed on six of its joints and it also has three degrees of force feedback to the user by means of motors installed on its first three joints.

The second robotic arm is the PUMA 560 manipulator from Unimation. This is also a six DOF manipulator and is generally used for industrial applications like welding spray painting and such other jobs related to factory automation including assembly operations. The PUMA has the ability to send feedback via its encoders, one on each joint.

The kinematics of the PUMA and the Omni are similar to each other as they are both six DOF arms and have a similar configuration. However their base Cartesian co-ordinate axis have different orientation and hence they have different DH parameters.

Robotic manipulators are open ended mechanisms i.e. they differ from closed mechanisms in that the last link is not physically connected to the base link. Moreover most of the general purpose manipulators have revolute joints that connect the neighboring rigid links. Each joint is a one DOF joint. This way the number of joints in a manipulator is equal to its DOF. A manipulator with n joints (and hence n DOF) will have n links. The links are numbered starting from the base which is called link 0. The moving member between the joints 1 and 2 is called link 1 and so on out to the free end of the arm which is link n. In a particular manipulator may not have all n links of finite length and some links may be of zero length. In order to define the POSE of an object completely in the 3D Cartesian space six variables are necessary and sufficient. Thus manipulators with six joints (or six DOF) oriented in a specific configuration are sufficient to reach any point in its workspace. If the end effector of the manipulator is not allowed to move then the manipulator is locked in that configuration. It appears like a rigid closed chained mechanism with the base and the end effector POSE as fixed ground points. Manipulators that have the ability to move one of their joints and not get locked at any POSE of the end effector when the end effector is not allowed to move are called redundant manipulators. Such manipulators will have more than six DOFs.

The joints of the manipulators may be revolute or prismatic. However, most of the manipulators have revolute joints and so do the two that we have used in our project i.e. the PUMA560 and the Omni. From now on whenever we mention joints we mean revolute joints only.

To describe the mechanisms similar to the manipulators described above i.e. those with revolute joints (lower pairs) a set of conventions called Denavit-Hartenberg notation [24] are used. There are four parameters that completely define a link with respect to its neighboring links.

A set of four parameters for all links are known as Denavit-Hartenberg parameters or DH parameters. The next section describes the DH parameters.

4.2. Link Frame Assignments and DH Parameters

First we introduce the reader to DH parameters. Consider the figure 20. This figure shows the connection of two links, link $i-1$ and link i via revolute joint i. The links here have two joints that connect with the neighboring links. It is found that two parameters are necessary and sufficient to completely define two lines in 3D space and hence to completely define a link. They are the link length, a and the link twist, $\propto$.

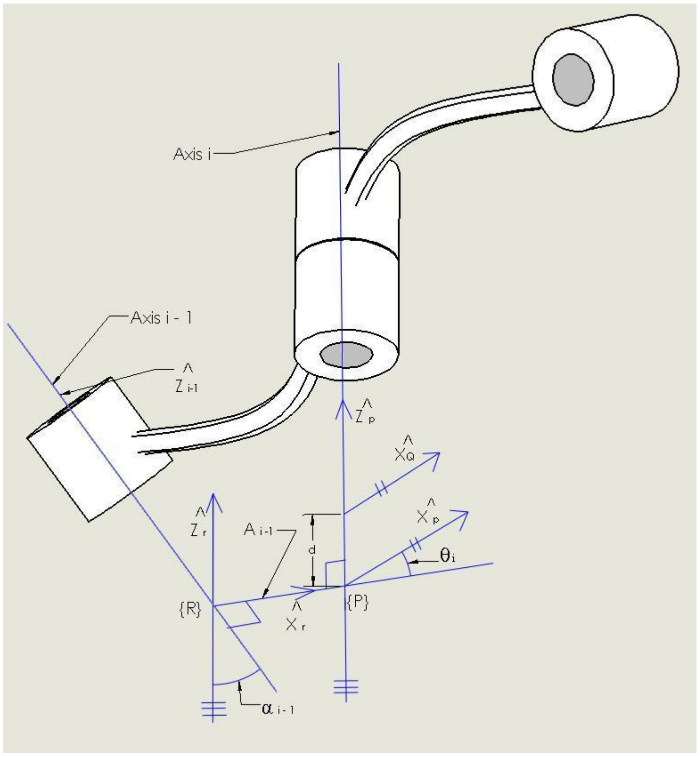


Figure 20: Robot link frames and DH parameters

If the joints are represented by lines that pass through the axis of the joints, then the distance measured along the mutual perpendicular to the two axes is the link length parameter.

The mutual perpendicular is unique except in the case when the two axes are parallel to each other in which case any mutual perpendicular defines the link length. If the axes are intersecting then the link length is zero. In the figure 20, a_{i-1} is the link length for link $i-1$.

The second parameter that defines the two axes is the link twist. If we imagine a plane perpendicular to the mutual perpendicular just constructed and we project the two axes, i and $i-1$, onto this plane then the angle between the $i-1$ and i, measured from $i-1$ to i in the right hand sense about $a_{i-1}$with the direction of a_{i-1} being from axis $i-1$ to i, then this angle is called the link twist $\propto_{i-1}$. When the two axes are intersecting each other then link twist $\propto_{i-1}$ is defined arbitrarily. In figure 20, the link twist is shown.

After describing the link completely only two parameters are necessary and sufficient to describe the link connections. These are the link offset d and joint angle θ.

Link offset defines the distance of one link from the other along the common axis of the links. In figure 20 the link offset of link i is d_i which is the signed distance from the point of intersection of link length a_{i-1} with axis i to the point of intersection of link length a_i with the axis i.

The second parameter known as the joint angle is the angle $\theta_i$measured from the line a_{i-1} to a_i in the right hand sense about axis i with the direction of i being such that the thumb is pointing outwards the manipulator structure.

The first and the last links in the chain of the manipulator are treated as special cases since link 0 does not exist physically and since the link n does not have all its parameters defined. For link 0 and link n, $a_0 = a_n = \propto_0 = \propto_n = d_1 = d_n = 0$. Moreover $\theta_1 = \theta_n = 0$.

Thus, a set of the four parameters for each joint define a manipulator completely and are known as the DH parameters of the manipulator.

Next we give the conventions used for assigning frames to links. In order to formalize the procedure generating DH parameters and hence to standardize it, Cartesian Co-ordinate frames are attached to each link with the origin of these frames at the joints. This also helps to define the DH parameters uniquely (although there can be more than one ways to define the DH parameters, which we will discuss later), generate the values of DH parameters so that they are zero for the base and last links which makes the calculations easier and also helps to determine manipulator kinematics relations between links that ultimately contribute to the determination of forward and inverse kinematics.

As mentioned above we affix Cartesian Co-ordinate frames to each link. These frames are stationary with respect to the link i.e. are rigidly attached to the links. However they move as their links move. The frames are numbered as per the link they are associated with. So, the frame associated with link i is called frame $\{i\}$.

The convention used for affixing frames to links is as follows. The z axis of the frame,$\hat{Z}$, is defined so that it points in the direction of joint axis $\{i\}$. Here there are actually two possibilities for defining the $\hat{Z}$ axis. It could be pointing in any of the two directions of the ray of axis $\{i\}$. Usually the direction of the $\hat{Z}$ axis is chosen in such a manner that it is pointing outward the structure of the manipulator. The origin is defined as on the axis $\{i\}$ and at the point where the mutual perpendicular a_i intersects the axis$\{i\}$. This makes it easier to determine the values of the link offset parametersd_i, d_{i+1} and so on. The $\hat{X}$ axis is defined to be pointing on the direction of a_i. For the case in which a_i is zero or the axis $\{i\}$ and $\{i+1\}$ intersect each other, the $\hat{X}$ axis is defined to be perpendicular to the plane defined by the axis $\{i\}$ and$\{i+1\}$. The $\hat{Y}$ axis is then determined by the right hand rule. Figure 20 shows the assignment of frames to the links $i-1$ and i.

As always the first and the last links of the chain are considered as special cases. The base frame, frame $\{0\}$, is attached to the non-moving part of the robot or to the base of the robot (link $\{0\}$) of the robot. Frame $\{0\}$ is assigned arbitrarily and so it simplifies the determination of DH parameters when frame $\{0\}$ is affixed so that the $\widehat{Z_0}$ axis points in the direction of that of frame $\{1\}$ and its origin coincides with that of frame $\{1\}$. Also, to fix the frame $\{0\}$, the position of $\hat{X}_0$ is chosen such that joint angle of joint 1, θ_1 is zero. As a result of this assignment, a_0 and $\propto_0$ become zero and d_1 becomes zero too. This makes the calculations of manipulator kinematics easier. For the joint *n*, frame $\{n\}$ is chosen so that $\hat{X}_n$ points in the direction of $\hat{X}_{n-1}$ when θ_{n-1} is zero and d_n is zero.

Here we describe a step by step procedure of assignment of frames to links and the derivation of DH parameters from these assignments. This also demonstrates that the DH parameters are derived from the frame assignments and once the frames are assigned. Thus it demonstrates the interrelationship of the frame assignments and the DH parameters and the dependence of DH parameters on the frame assignments. It basically is a two step process to define a manipulator for kinematic analysis. The first step involves affixing of frames to links and the second step involves determining the DH parameters from this. Later on we will introduce modified DH parameters' for the PUMA560 robot which have the same derivation convention for DH parameters but differ in the assignment of frames to the PUMA links i.e. the location of frames on links. This difference in the assignment of frames to PUMA links modifies the DH parameter values.

The procedure for link frame assignment is given below:

1. Identify all the joints in the manipulator and assign a ray passing thorough the center of these joints. Let these rays be the joint axes.
2. Assign numbers to these joints and axes. Let joint 0 be coincident with joint 1 but consider joint 0 immobile whereas joint 1 will rotate. The joint preceding a link gets that number associated with it. For example, in figure 20, the joint that precedes link $i-1$ is the joint $i-1$.
3. Let $\hat{Z}_i$ lie along the axis i.
4. Identify mutual perpendiculars between two consecutive axes. Let the origin of frame $\{i\}$ be at the point of intersection of the perpendicular that passes through link i with the axis $\{i\}$. Consider following two special cases:
   a. If the axes are parallel and have infinitely many mutual perpendiculars, then choose that mutual perpendicular that will result in d_i becoming zero.

b. If the axes intersect each other then the origin of frame $\{i\}$ is chosen at the point of intersection of axes i and $i-1$ or so that a_{i-1} and/or d_i is zero.

5. Let $\hat{X}_i$ lie along the mutual perpendicular pointing from axis i to axis $i+1$. Consider the following special case:

    a. If the mutual perpendicular is zero or the neighboring axis intersect then $\hat{X}_i$ is perpendicular to the plane defined by $\hat{Z}_i$ and $\hat{Z}_{i+1}$

6. Define $\hat{Y}_i$ by the right-hand rule.

7. Frame $\{0\}$ is chosen so that it coincides with frame $\{1\}$ when θ_1 is zero.

8. Frame $\{n\}$ is chosen so that $\hat{X}_n$ coincides with $\hat{X}_{n-1}$.

After the link frames have been assigned to the manipulator, the DH parameters are then determined in the following manner:

1. Link Length, a_i, is the distance measured along $\hat{X}_i$ from $\hat{Z}_i$ to $\hat{Z}_{i+1}$. It is always positive.

2. Link twist, $\propto_i$, is the angle measured in the right hand sense about $\hat{X}_i$ from $\hat{Z}_i$ to $\hat{Z}_{i+1}$.

3. Joint offset, d_i, is the distance measured along $\hat{Z}_i$ from $\hat{X}_{i-1}$ to $\hat{X}_i$. It can be positive or negative.

4. Joint angle, θ_i, is the angle measured in the right hand sense about $\hat{Z}_i$ from $\hat{X}_{i-1}$ to $\hat{X}_i$.

For a particular manipulator, there are more than one ways to affix frames and hence more than one value of DH parameters i.e. the DH parameters generated is not unique. As we had seen earlier, there are two ways of assigning the $\hat{Z}$ axis as the joint axis extends in both directions like a ray, there are two ways of assigning the $\hat{X}$ axis also when the neighboring joint axis intersect i.e. on any of the two sides of the plane that is formed by the intersection of the two joint axes. However, the frames are chosen in such a way so that the values of the Link parameters or DH parameters are zero or as simplified as possible.

Another note that needs to be mentioned here is that while the link length and link twist parameters for a particular link are determined from the joint associated with that link and the one associated with the next link, the link offset and joint angle parameters for a link/joint are determined from the joint associated with that link and the one associated with the previous link.

4.3. Manipulator Forward Kinematics

As mentioned earlier manipulator forward kinematics or simply forward kinematics is the operation that determines the position and orientation of link n with respect to link 0. Actually we can determine the POSE of any link with respect to the base link (or any other link) once the equations for forward kinematics are developed. However, most of the times we are interested only in determining the POSE of link n with respect to link 0. The operation involves using the DH parameters to form a transformation matrix that defines a frame associated with a link with respect to that associated with the previous link. The transformation matrices obtained in this way are a representation of the POSE of one joint (or link) with respect to the previous. Then all such transformation matrices are concatenated and we obtain the transformation matrix that defines the POSE of link n with respect to link 0.

The general form of a transformation matrix that defines the POSE of a link (or joint) with respect to a previous link (or joint) in terms of the DH parameters is given by the following equation:

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ -s\theta_i \,.\, c\propto_{i-1} & c\theta_i \,.\, c\propto_{i-1} & -s\propto_{i-1} & -s\propto_{i-1}.d_i \\ s\theta_i \,.\, s\propto_{i-1} & c\theta_i \,.\, s\propto_{i-1} & c\propto_{i-1} & c\propto_{i-1}.d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \ldots\ldots\ldots\ldots(2) $$

For derivation of this equation the reader is advised to refer to [24] pg. 74-75. After obtaining the individual link transformation matrices, ${}^{i-1}_{i}T$ for all links, they can be multiplied together to generate the transformation matrix that relates frame $\{N\}$ to frame $\{0\}$.

$$ {}^{0}_{N}T = {}^{0}_{1}T\ {}^{1}_{2}T\ {}^{2}_{3}T\ \ldots.\ {}^{N-1}_{N}T \ldots\ldots\ldots\ldots(3) $$

The link parameters (or the DH parameters) are all constant for a particular manipulator except for the joint angle θ. Thus by knowing the values of all the link parameters we can develop equations, called kinematic equations, which define the POSE of a link with respect to the previous link in terms of joint variables. This way kinematic equations are developed that define the POSE of link n with respect to link 0 in terms of all joint angles (six in the case of PUMA560). By knowing the joint angles from the encoders on the manipulator, we can determine the POSE of link n with respect to link 0, ${}^{0}_{N}T$. Thus the transformation ${}^{0}_{N}T$ is a function of all n joint variables.

Here we summarize the procedure for computing the forward kinematics of any manipulator with n joints or n DOF in the four steps below:

1. Affix frames to the links at the joints and number them as per the links starting from the base $\{0\}$, $\{1\}$, $\{2\}$ …. $\{n\}$ and so on.
2. Determine the DH parameters for all the links and tabulate the values $\propto_{i-1}$, $a_{\boldsymbol{i-1}}$, $d_{\boldsymbol{i}}$, θ_i (only joint angles would be variables).
3. Develop individual link transformation matrices that define the POSE of a particular link with respect to the previous link, ${}^{i-1}_{i}T$.
4. Multiply the individual link transformation matrices to determine the POSE of link n with respect to the base (also known as the transformation matrix of the end effector with respect to the base), ${}^{0}_{N}T$ in terms of joint variables.

4.3.1. PUMA Forward Kinematics

In this section, we determine the kinematic equations for solving the forward kinematics of the PUMA560 manipulator. For this we will assign co-ordinate frames to the links and the base of the PUMA, determine the DH parameters, develop the individual link transformation matrices and then finally develop the transformation matrix that gives the POSE of the end effector with respect to the base in terms of the joint variables $\theta_{i=1\ to\ 6}$.

The PUMA560 manipulator is a widely used industrial robot. It has six revolute joints. The last three of its joints intersect at a point which is a common arrangement for industrial robots. It makes the solution of inverse kinematics, which is solving for the joint angles when the Cartesian POSE of the end effector is known, easier. For more information on the PUMA560 manipulator the reader is suggested to browse through chapter 3 which explains the manipulator in greater detail. For further information the reader is suggested to browse through [25].

First we assign Cartesian co-ordinate frames to the PUMA560 manipulator.

Figures 21 and 22 show the assignment of frames to the six links of the PUMA560 manipulator and to the base of the manipulator.

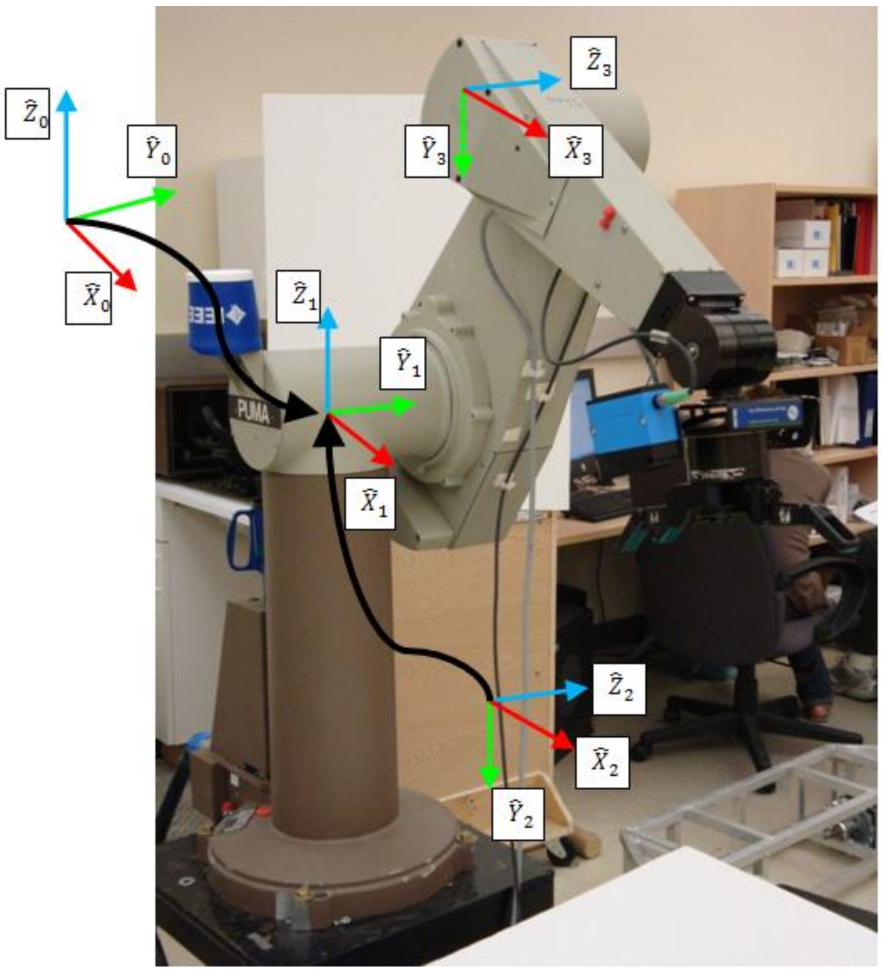


Figure 21: Frame assignments to joints 0, 1, 2 and 3 of PUMA

The conventions described in the previous section are used in the frame assignment. As expected the frame $\{0\}$ coincides with $\{1\}$ when $\theta_1 = 0$. Also, as shown figure 22 that as joints 4, 5 and 6 coincide the origin of the frames $\{4\}$, $\{5\}$ and $\{6\}$ are the same. We must note that there are more than one ways of representing the frames as, as discussed, there are more than one ways of choosing the $\hat{Z}$ axis for all joints and more than one ways of choosing the $\hat{X}$ axis when link length for the two links is zero.

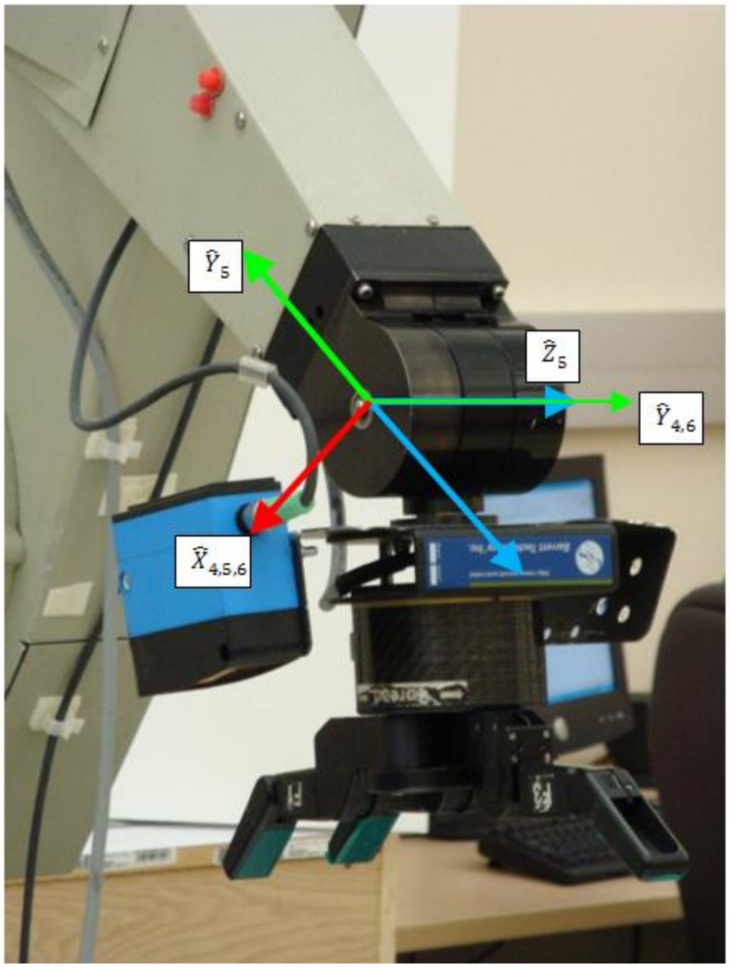


Figure 22: Frame assignment for joint 4, 5 and 6

Next we determine the PUMA DH parameters. The DH parameters are based on the convention followed in John Craig [24] and thus are modified DH parameters. The convention for deriving the modified DH parameters is the same as that for the standard DH parameters and they differ in the assignment of frames. The modified DH parameters are tabulated below:

Table 2: Modified DH parameters of PUMA560 manipulator

Link i	$\propto_{i-1}$	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	-90^0	0	0	θ_2
3	0	0.4318	0.15	θ_3
4	-90^0	0.0203	0.4331	θ_4
5	90^0	0	0	θ_5
6	-90^0	0	0	θ_6

Now we determine the individual link transformation matrices. Using Equation 1 we determine the transformation matrix that defines the POSE of link i with respect to $i-1$ in terms of the joint angles $\theta_{i=1\ to\ 6}$.

$$ {}^0_1T = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots(4)$$

$$ {}^1_2T = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s\theta_2 & -c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots(5)$$

$$ {}^2_3T = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & 0.4318 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0.15 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots(6)$$

$$ {}^3_4T = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & -0.0203 \\ 0 & 0 & 1 & 0.4331 \\ -s\theta_4 & -c\theta_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots(7)$$

$${}^{4}_{5}T = \begin{bmatrix} c\theta_5 & -s\theta_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_5 & c\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots(8)$$

$${}^{5}_{6}T = \begin{bmatrix} c\theta_6 & -s\theta_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s\theta_6 & -c\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots(9)$$

Now, we determine the transformation matrix that determines the POSE of end effector with respect to the base, ${}^{0}_{6}T$,

$${}^{0}_{6}T = \begin{bmatrix} noap0 & noap3 & noap6 & noap9 \\ noap1 & noap4 & noap7 & \text{noap10} \\ noap2 & noap5 & noap8 & noap11 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots\dots(10)$$

where,

noap0 = (-c1 * c23 *s4 - s1 * c4) * s6 + ((c1 * c23 * c4 - s1 * s4) * c5 - c1 * s23 * s5) * c6 .. (11)

noap1 = (c1 * c4 - s1 * c23 * s4) * s6 + ((c1 * s4 + s1 * c23 * c4) * c5 - s1 * s23 * s5) * c6. (12)

noap2 = s23 * s4 * s6 + (-c23 * s5 - s23 * c4 * c5) * c6.. (13)

noap3 = (c1 * s23 * s5 + (s1 * s4 - c1 * c23 * c4) * c5) * s6 + (-c1 * c23 * s4 - s1 * c4) * c6 (14)

noap4 = (s1 * s23 * s5 + (-c1 * s4 - s1 * c23 * c4) * c5) * s6 + (c1 * c4 - s1 * c23 * s4) * c6 (15)

noap5 = (c23 * s5 + s23 * c4 * c5) * s6 + s23 * s4 * c6 .... (16)

noap6 = (c1 * c23 * c4 - s1 * s4) * s5 + c1 * s23 * c5.. (17)

noap7 = (c1 * s4 + s1 * c23 * c4) * s5 + s1 * s23 * c5.. (18)

noap8 = c23 * c5 - s23 * c4 * s5 .... (19)

noap9 = (noap6) * d6 + c1 * s23 * d4 - s1 * d3 + c1 * c23 * a3 + c1 * c2 * a2 .... (20)

noap10 = (noap7) * d6 + s1 * s23 * d4 + c1 * d3 + s1 * c23 * a3 + s1 * c2 * a2..................... (21)

noap11 = (noap8) * d6 + c23 * d4 - s23 * a3 - s2 * a2.. (22)

Equation (10) completely defines the kinematics of the PUMA560 manipulator. It helps in the determination of the POSE of the frame $\{6\}$ with respect to the frame $\{0\}$ when all the joint angles are known.

4.4. Inverse Manipulator Kinematics

Inverse manipulator kinematics is, as the name implies, the inverse of the forward kinematics. In Inverse kinematics we solve the equations developed in the forward kinematics solution formulation for joint angles given the POSE of the end effector with respect to the base ${}_{N}^{0}T$ (or ${}_{6}^{0}T$ in the case of the PUMA560 manipulator). Thus we solve for $\theta_{i=1\ to\ 6}$ given ${}_{6}^{0}T$.

Inverse kinematics is an important computational step in the process of trajectory generation of the arm. As mentioned, inverse kinematics on a manipulator calculates its joint rates given the Cartesian rates of its end effector. With inverse kinematics the user (or the programmer) need not specify the joint rates for each joint along the trajectory that the arm is intended to follow. This way of specifying a trajectory is non intuitive and the user basically has to carry out inverse kinematic computations for each and every point on the trajectory which is not possible. The user would prefer any easier way of specifying a trajectory in terms of the end effector Cartesian velocities (or Cartesian positions). After this the inverse kinematics takes care of computing the joint rates (or joint positions) that each joint must assume through the intended trajectory.

The inverse kinematics not only makes trajectory specification easier it is also necessary for the following reason. The motion controller that controls the rotation of each joint of the manipulator requires voltage values that it needs to send to each joint of the manipulator. For computing the voltage values for each joint, torque that each joint needs to be subjected to move to the next point in the trajectory needs to be computed. To calculate this torque value, joint rates (or velocities) need to be computed for each joint which is done by inverse kinematics.

PUMA560 manipulator has 6 joints and hence the solution of the inverse kinematics problem should yield six joint angles, $\theta_{i=1\ to\ 6}$. As we saw in the previous section, there are 12 equations (in equation 10) that are required to be solved. These twelve equations come out of the twelve non-zero values of the transformation matrix ${}^{0}_{6}T$. However out of these twelve matrix members, there are only six independent members. Out of the nine members of the rotation part of ${}^{0}_{6}T$, only three are independent and the other three are the translation components. So in all there are six variables and six unknown joint angles.

Also, PUMA560 manipulator has a very common construction of its joints in that its last three joints intersect at a common point and have a common joint frame origin viz., joints 4, 5 and 6. This common construction makes solving for inverse kinematics much easier and a closed form method called Piper's solution when three axes intersect makes this possible. This method also applies to manipulators which have three of their axis parallel (as these three axes intersect at a point at infinity). Piper's solution can solve for the inverse kinematics on majority of industrial robots [24]. The importance of this type of design in industrial robots will be discussed in the following sections.

An important note to be mentioned here is that the solution for inverse kinematics does not always exist. There is a certain volume of space in which the manipulator can move or rather its end effector can reach. This volume is known as the workspace of the manipulator. At points outside this workspace where the manipulator cannot reach, the solution does not exist and these points are known as singular points. The points that are outward of the workspace with respect to the robot are external singularities and those that are inward i.e. between the manipulator's structure and the workspace are internal singularities. As the manipulator approaches singular points its joint velocity goes on increasing and so do joint torques and at singular points the velocity is ideally infinity and unpredictable behavior should be expected out of the robot which is dangerous for people around it in terms of injury and can even cause death. To avoid singular points from being reached and to enable the manipulator to still reach the target points, singularity avoidance method of singularity-robust inverse of the Jacobian has been implemented. In this method the inverse kinematics computes approximate solutions and not exact solutions that enable the manipulator to go around the singular points and come back to its original trajectory when it has passed the singular point. This way the manipulator does not go over the singular points or near enough so as to cause high velocities and unpredictable behavior. This is similar to obstacle avoidance if the singular point is considered as an obstacle. This method will be discussed in the next chapter.

Another important note to be mentioned here is that at a time more than one solutions may exist i.e., the end effector of the manipulator would be able to locate the same target with more than one configuration. This can create problems when the computations are performed by a system because the system has to choose one solution out of the many possibilities. There are different criteria that are used in choosing a solution. The criteria are least distance travelled by the manipulator so that it will choose the solution that is closest to its current location. Another criteria could be lesser motion of the first three joints than that of the last three (in case of a manipulator with six joints and “good” link lengths – good link length means that first link is the longest and they progressively become short as we move towards link 6) as this will result in lesser overall movement of the manipulator.

There are two major ways by which the conversion of Cartesian rates to joint rates can be achieved. One is through closed form solution and the other is by using the Resolved Rate implementation. Each of these will be discussed in detail in the following sections. There are other ways of computing the inverse kinematics solutions too like the geometric method which is a form of closed form solution method (and differs only in approach) and the numerical solutions iterative method. But as these methods are not widely applied and we have not implemented those we will not be discussing them. The reader is suggested to refer to the literature [24] for more information on these methods and for information on the other methods.

4.4.1. Closed Form Method of Computing Joint Rates

The closed form method of computing joint rates from Cartesian rates uses an exact solution to the set of equations that relates the joint velocities to the Cartesian velocities. An exact solution is a solution that can be obtained by algebraically solving an equation (generally polynomials of degree four or less can be solved algebraically) and not using any iterative method to solve the equations. Thus, the equations that were derived for solving for forward kinematics viz. equation 10 are solved algebraically to determine the joint angles provided ${}^{0}_{6}T$ is known.

${}^{0}_{6}T$ are the various trajectory points determined from the trajectory generator and they are solved for the joint angles at each point using equation 10 algebraically. Because the equations are non-linear and transcendental in nature there is no general algebraic solution available. Moreover, since the equations that relates joint rates to Cartesian rates viz. equation 10 in the case of PUMA560 manipulator are different for different manipulators. So there is no general solution to these equations. The equations have to be solved algebraically and when they are in the form of n expressions such that each expression expresses a joint angle in terms of Cartesian rates, for an n joint manipulator, then these expressions can be coded into an algorithm to get joint angles on the fly as the manipulator moves. So the computations that are done when the manipulator moves are basically evaluation of the algebraic expressions and not solving equations. Equations are solved offline.

Another point to be noted is that as the equations are non-linear as well as transcendental they are difficult to solve algebraically and require complex algebraic manipulations. However for most of the manipulators, some of the algebraic manipulations are common in solving the kinematic equations which makes closed form inverse kinematics easy to solve. For examples on how the inverse kinematics is solved using the closed form method by using algebraic manipulations that make the non-linear and transcendental equations easy to solve, the reader is suggested to refer to Craig [24].

A recent result mentioned in Craig [24] is that manipulators with six joints (and possibly up to six joints) are solvable. The above statement is true for an iterative method and is not always true for analytic or closed form method. An analytic solution for a manipulator with six joints will exist only if certain conditions are satisfied like certain number of joint's axis should be intersecting at a point or several joint twists $\propto_i$ should be either 0 or (+/-)90^0. Therefore to be able to solve a manipulator analytically or using closed form solution, it is important to design a manipulator in a particular way. Most industrial manipulators are designed in this way after the designers realized this. A sufficient condition for a manipulator with six joints to have an analytic solution available is that at least its three neighboring joints intersect at a point. This is called Pipers solution as is given in detail in Craig [24]. Almost every industrial manipulator is built with this condition in mind including PUMA560 which has its last three joints, joints 4, 5 and 6 intersecting at a point.

Here we list the expressions that we have used in our algorithm to determine the six joint variables of the PUMA560 manipulator on the fly. These expressions are basically the solutions of the kinematic equations. For the complete analytic solution the reader is suggested to refer Craig [24] or Lee, Fu, Gonzalez [26]. To resolve for the ambiguity of multiple solutions, the solution closest to the last configuration is chosen.

Input to the closed form inverse kinematics algorithm:

1. Transformation of end effector with respect to base, ${}_6^0T = \begin{bmatrix} n_x & o_x & a_x & P_x \\ n_y & o_y & a_y & P_y \\ n_z & o_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$
2. Current configuration of PUMA. The four possibilities are RIGHTANDUP, RIGHTANDDOWN, LEFTANDUP and LEFTANDDOWN. Based on the sign of the arm and elbow values, which are based on DH parameters and joint angles, one out of the four configurations is chosen (Refer Gonzalez, Lee)
3. Current joint angles, $\theta_{c_{i=1\ to\ 6}}$
4. PUMA560 DH parameters which are non-zero: *a2, a3, d3, d4*

5. Define joint limits for all the joints J1LO, J1HI, J2LO, J2HI, J3LO, J3HI, J4LO, J4HI, J5LO, J5HI, J6LO, J6HI, where J1LO is the lower limit of joint 1 and J1HI is the higher limit of joint 1.

The steps to calculate the joint angles by using closed form inverse kinematics are listed below:

1. Calculate joint angle θ_1 and check for joint limits:
   a. Evaluate $rkin$ = p_x^2 + p_y^2 + d_3^2
      i. If $rkin$ < 0, then "Solution does not exist", otherwise, evaluate $rkin$ = $\sqrt{rkin}$
      ii. If configuration = RIGHTANDUP OR configuration RIGHTANDDOWN, then $rkin$ = - $rkin$
   b. θ_1 = arctan $(\frac{P_y}{P_x})$ - arctan $(\frac{\mathrm{d3}}{rkin})$.
      i. If θ_1< J1LO, then θ_1 = θ_1 + (2*π), otherwise if θ_1 > J1HI, then θ_1 = θ_1 - (2*π)
      ii. If still $\theta_1$is out of its limits, then "Joint 1 is out of its range"
2. Evaluate joint angle θ_3 and check for joint limits:
   a. $f11$ = c1. P_x + s1. P_y
   b. d = $f11^2$ + p_z^2 - d_4^2 - a_3^2 - a_2^2
   c. Evaluate $w1$ = [4 * a_2^2 * (a_3^2 + d_4^2)] – d^2
      i. If $w1$< 0, then "Solution does not exist"
      ii. $rkin$ = $\sqrt{w1}$
      iii. If configuration = LEFTANDUP OR configuration = RIGHTANDDOWN, then $rkin$ = - $rkin$
   d. θ_1 = arctan $(\frac{d}{rkin})$ - arctan $(\frac{\mathrm{a3}}{\mathrm{d4}})$
      i. If θ_3< J3LO, then θ_3 = θ_3 + (2*π)
      ii. If still θ_3 is still out of its limits, then "Joint 3 is out of its range"

3. Evaluate joint angle θ_2 and check for joint limits:

   a. $w1$ = a_2* c3 + a_3

   b. $w2$ = d_4 + a_2 * s3

   c. θ_{23} = arctan $(\frac{w2*f11-w1*P_Z}{w1*f11+w2*P_Z})$, where θ_{23} = θ_2 + θ_3

   d. θ_2 = θ_{23} - θ_3

      i. If θ_2> J2HI, then θ_2 = θ_2 - (2*π)

      ii. If still θ_2 is still out of its limits, then "Joint 2 is out of its range"

4. Evaluate joint angle θ_4:

   a. θ_4 = θ_5 = θ_6= 0

   b. $noap_{i=1\ to\ 12}$ = FKINE($\theta_{i=1\ to\ 6}$)

   c. $\bar{A}$ = $noap[6].\hat{\imath}$ + $noap[7].\hat{\jmath}$ + $noap[8].\hat{k}$

   d. $\bar{B}$ = $a_x.\hat{\imath}$ +$a_y.\hat{\jmath}$ +$a_z.\hat{k}$

   e. $\bar{C}$ = $\bar{A}$ X $\bar{B}$

   f. $\bar{D}$ = $o_x.\hat{\imath}$ +$o_y.\hat{\jmath}$ +$o_z.\hat{k}$

   g. e = $\bar{C}$. $\bar{D}$

      i. If e=0.0, then, "Invalid Solution"

      ii. If e < 0.0, then, θ_4 = arctan $(\frac{s1*a_x-c1*a_y}{-c23*(c1*a_x+s1*a_y)+s23*a_Z})$, otherwise θ_4 =

      arctan $(\frac{-s1*a_x+c1*a_y}{c23*(c1*a_x+s1*a_y)-s23*a_Z})$

5. Evaluate joint angle θ_5:

   a. θ_5 = arctan $(\frac{c4*(c23*(c1*a_x+s1*a_y)-s23*a_Z)+s4*(-s1*a_x+c1*a_y)}{s23*(c1*a_x+s1*a_y)+c23*a_Z})$

6. Evaluate joint angle θ_6:

   a. d = c1 * o_x + s1 * o_y

   b. e = -s1 * o_x + c1 * o_y

   c. θ_6 = arctan $(\frac{-c5*(c4*(c23*d-s23*o_Z)+s4*e)+s5*(s23*d+c23*o_Z)}{-s4*(c23*d-s23*o_Z)+c4*e})$

7. Determine the solution closest to the current arm configuration:

   a. If $|\theta_4 - \theta_{c_4}|$ > $|-\theta_4 - \theta_{c_4}|$, then θ_4 = θ_4 + (π) and θ_5 = - θ_5 and θ_6 = θ_6 + (π)

8. Check for joint angle wrap around

   a. if $(\theta_4 - \theta_{c_4})$ > π, then θ_4 = θ_4 - (2*π), otherwise

      i. if $(\theta_4 - \theta_{c_4})$ < -π then θ_4 = θ_4 + (2*π)

      ii. if $(\theta_5 - \theta_{c_5})$ > π then θ_5 = θ_5 - (2*π), otherwise if $(\theta_5 - \theta_{c_5})$ < -π then θ_5 = θ_5 + (2*π)

9. Check limits for joints 4, 5 and 6.

   a. If θ_4< J4LO and θ_4 > J4HI, then "Joint 4 is out of its range"

   b. If θ_5< J5LO and θ_5 > J5HI, then "Joint 5 is out of its range"

   c. If θ_6< J6LO and θ_6 > J6HI, then "Joint 6 is out of its range"

Even though the provision of closed form inverse kinematics solution is there in the algorithm, we have used Resolved Rate method, which we will be discussing next, to solve for joint angles. This is because closed form inverse kinematic solutions are specific for each arm and the same equations that we use for PUMA560 manipulator cannot be used for other manipulators. The reason for this is that because the DH parameters and the joint co-ordinate frames of each arm are different, the kinematic equations for each arm are different and so is the inverse kinematics solution. But since we have included closed form inverse kinematics provision in our code and it can be included if desired, we have listed the expressions here.

4.4.2. Resolved Rate Method of Computing Joint Rates

As mentioned previously the closed form solution of inverse kinematics is not a general method of inverse kinematics for all manipulators because the equations of manipulator kinematics are different for different manipulators. Therefore a more general method called the Resolved Rate algorithm [27] is used for many applications and we have also made use of the Resolved Rate algorithm for solving the inverse kinematics. The Resolved Rate algorithm has been used in both teleoperation as well as autonomous control of the manipulator. The Resolved

Rate method of solving for inverse kinematics is also not a completely general method because a matrix called Jacobian, which we will be explaining next, is different for different manipulators and changes also for the same manipulator depending on which frame Cartesian rates are taken as input for computing joint rates. However, once an expression for the Jacobian is determined, which is in terms of DH parameters including joint variables, the rest of the procedure remains the same.

We first introduce the readers to Jacobian. A Jacobian is a multidimensional form of a derivative and relates a number of variables dependent on a certain fixed number of same independent variables in the form of a linear matrix equation.

Consider the following functions, f_1, f_2, ……, $f_6$that relate dependent variables y_1, y_2, ……, y_6 to independent variables x_1, x_2, ……, x_6.

$y_1 = f_1(x_1, x_2, x_3, x_4, x_5, x_6)$

$y_2 = f_2(x_1, x_2, x_3, x_4, x_5, x_6)$

.

.

.

$y_6 = f_6(x_1, x_2, x_3, x_4, x_5, x_6)$... (23)

In the matrix equation form this can be written as, Y = F(X). Now if we take the partial derivatives of each dependent variable with respect to all independent variables, we get,

$\delta y_1 = \frac{\delta f_1}{\delta x_1}\delta x_1 + \frac{\delta f_1}{\delta x_2}\delta x_2 + \ldots\ldots + \frac{\delta f_1}{\delta x_6}\delta x_6$

$\delta y_2 = \frac{\delta f_2}{\delta x_1}\delta x_1 + \frac{\delta f_2}{\delta x_2}\delta x_2 + \ldots\ldots + \frac{\delta f_2}{\delta x_6}\delta x_6$

.

.

.

$\delta y_1 = \frac{\delta f_6}{\delta x_1}\delta x_1 + \frac{\delta f_6}{\delta x_2}\delta x_2 + \ldots\ldots + \frac{\delta f_6}{\delta x_6}\delta x_6$ .... (24)

In the matrix equation form the above equation can be written as follows:

$\delta Y = \frac{\delta F}{\delta X}\delta X$ .... (25)

The 6X6 matrix of partial derivatives is what is known as the Jacobian, J. If the functions f_1 through f_6 are non-linear functions of x_1 through x_6, then the Jacobian is a function of $x_{i=1\ to\ 6}$. Hence we can write the following:

$\delta Y = J(X).\delta X$ .... (26)

Taking time derivatives on both sides we can write as follows:

$\dot{Y} = J(X).\dot{X}$ .... (27)

If the independent variables $x_{i=1\ to\ 6}$ have certain values then $J(X)$ is constant and the above expression becomes a linear matrix equation and $J(X)$ is a time varying linear transform. It is time varying because at each instant as the value of X changes the value of $J(X)$ also changes.

In manipulator kinematics, the Jacobians are used to relate the joint rates to the Cartesian rates in the following way:

${}^{0}v = {}^{0}J(\theta)\,.\,\dot{\theta}$ .... (28)

where θ is a vector of joint angles and v is a vector of Cartesian velocities. The superscript describes the frame with respect to which the Cartesian velocities are expressed which is also the frame for the Jacobian. If the Cartesian velocities are expressed with respect to some other frame, then the Jacobian is also expressed in that frame and it changes. The number of rows of the Jacobian equals the degrees of freedom in the Cartesian space and the number of columns is equal to the number of joints of the manipulator. So for a planar robot with four joints the Jacobian is a 3X4 matrix. The Jacobian need not be a square matrix and can have any no. of rows and columns depending on the DOF in Cartesian space and the number of joints. For a 6 joint manipulator like the PUMA560, the Jacobian is a 6X6 matrix, $\dot{\theta}$ is a 6X1 joint velocity vector and v is a 6X1 Cartesian velocity vector with a 3X1 linear velocity vector and a 3X1 rotational velocity vector stacked together.

That is, ${}^{0}v$ = $\begin{bmatrix} {}^{0}v \\ {}^{0}\omega \end{bmatrix}$, where ${}^{0}v$ is a 3X1 linear velocity vector of linear velocities along the three Cartesian co-ordinate axis x, y and z and ${}^{0}\omega$ is a 3X1 rotational velocity vector with angular velocity of the end effector about the three Cartesian co-ordinate axis. The superscript that indicates the reference frame can be omitted when it is obvious which frame is being referenced.

We must remember that the Jacobian is a function of joint angles. When the Resolved Rate algorithm computes the joint angles using equation (16) and the manipulator moves as per the computed joint angles, the Jacobian which is a function of joint angles changes and must be evaluated again for the next step or the next point in the trajectory.

One of the ways to evaluate the expression for the Jacobian of a manipulator is by the successive application of the equations of link linear and angular velocities given below.

These equations relate the linear and angular velocities of a particular link with joint velocities of all joints up to that link. On applying these equations to each link, starting from the first link that is attached to the base and moving towards the nth link, we can ultimately develop a matrix equation that relates the linear and angular velocity of the last link to joint velocities of all joints. The Cartesian velocities in this equation are with respect to the nth link and so to describe it with respect to the base we can pre multiply by the rotation matrix ${}^{0}_{n}R$. On decomposing this matrix we get the equation of the form of equation (16) and hence the Jacobian. For examples the reader is suggested to refer to Craig [24] pg. 148-150.

Now we discuss the importance of the inverse of the Jacobian of the manipulator, define the important concept of singularity and also describe the relation between the two.

When generating a trajectory, we specify the Cartesian velocities that the end effector will follow and perform inverse kinematics to compute joint angles which are then sent to the motion controller to move the manipulator along the trajectory. The equation (16) would be useful only when we express joint velocities in terms of the Jacobian and Cartesian velocities. For this we have to compute the inverse of the Jacobian and the equation becomes:

$$\dot{\theta} = J(\theta)^{-1} \, . \, v \quad \text{.......} \quad (29)$$

From matrix algebra we know that the inverses of only square matrices exist. Hence the inverse of the Jacobian for manipulators which have less than six or more than six joints (redundant manipulators) cannot be computed by general matrix algebra. For these manipulators pseudo inverses or singularity-robust inverses are computed. These will be discussed in the next chapter where we deal with the subject of singularity in greater detail. The pseudo and SR inverses also serve a dual purpose that of preventing the manipulator from entering singular configurations while following trajectories and thus preventing unstable behavior of the manipulator which could be unsafe for the occupants around the manipulator. This is also dealt with in the next chapter. Here we introduce singularity.

The Jacobian is also not invertible at certain configurations of the arm known as singular configurations. This is because at these singular configurations the determinant of the Jacobian becomes zero and the inverse of the Jacobian tends to infinity. Hence the joint rates tend to be very high. This results in an unstable behavior of the manipulator. These points where the manipulator is in a singular configuration are known as singular points and the manipulator is said to have singularity. The manipulator looses at least one of its DOF when it reaches a singularity. Following are the two types of singularities:

1. Workspace-boundary singularities: These occur at or very near the boundary of the workspace of the manipulator. Either the manipulator is completely stretched or is

folded inside and so is unable to move any further. The former happens at the outer boundary and the latter happens in a region engulfed by the workspace.

2. Workspace-interior singularities: These occur when the manipulator is well within its workspace but two or more of its joints are in such a position that the manipulator has lost one of its DOFs e.g., lining up of two or more joints.

It is very dangerous for the manipulator to have its end effector at or very near singular points as at these points joint rates tend to infinity, the manipulator behavior is unstable and unsafe for occupants nearby. Singularity avoidance techniques are discussed in the next chapter.

Now we give details of the Resolved Rate algorithm. Having defined the Jacobian and mentioned the importance of the computing inverse of the Jacobian to compute joint rates, now we describe the Resolved Rate method for inverse kinematics. We have used the Resolved Rate method for controlling the arm in both the teleoperation and autonomous modes.

The schematic block diagram for the implementation of the Resolved Rate algorithm is shown in the figure 23.

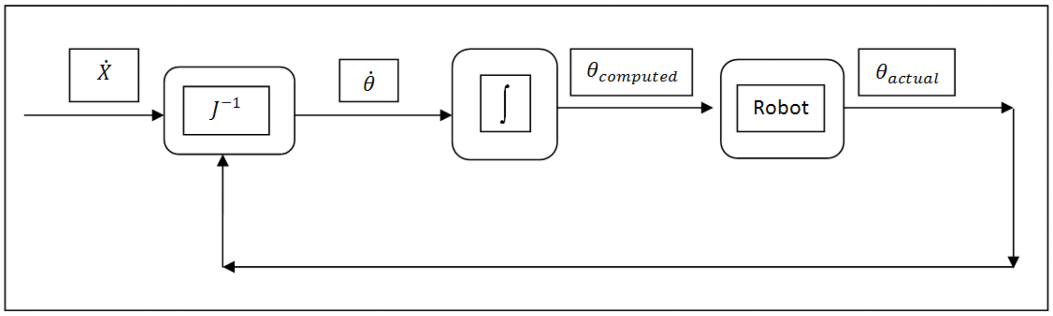


Figure 23: Schematic block diagram of the Resolved Rate algorithm

In the Resolved Rate algorithm, the input is the end effector velocity vector $\dot{X}$, which is the same as v above in Equation (5) above. It consists of end effector linear velocity components,$\dot{v}_x$ $\dot{v}_y$ and $\dot{v}_z$ as well as rotational velocity components $\dot{w}_x$, $\dot{w}_y$ and $\dot{w}_z$. Each of the two is a 3X1 vector that makes the end effector velocity vector$\dot{X}$ a 6X1 vector.

$$\dot{X} = \begin{bmatrix} \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{w_x} \\ \dot{w_y} \\ \dot{w_z} \end{bmatrix} \quad (30)$$

Another input to the algorithm is the Jacobian. Then the following equation is used to determine the joint angles.

$$\dot{X} = \mathrm{J}(\theta) * \dot{\theta} \quad (31)$$

Here, *J* is the Jacobian and is a function of the end joint angles. As the manipulator moves in the workspace, its joint angles change and so does the Jacobian. Hence Jacobian is time varying and needs to be updated at every time step. $\dot{\theta}$ is the 6X1 joint rate vector that gives the joint velocities at each of the joints.

In the case of the PUMA, the Jacobian is a 6X6 matrix. As it is a square matrix, the Jacobian inverse, J^{-1} exists, except at singular points (a case which we will deal in the chapter 5 on arm dexterity improvement), and hence it is possible to compute joint rates from end effector velocity components directly by using the following equation:

$$\dot{\theta} = J^{-1} \times \dot{X} \quad (32)$$

A new input velocity vector $\dot{X}$ is constantly supplied to the algorithm and it may change as the arm traverses along the trajectory. Also, as the manipulator traverses along the trajectory, its joint angles constantly change and so does the Jacobian as it is a function of joint angles. The Jacobian is constantly updated. Next the equation (20) is used to compute the joint rates vector, $\dot{\theta}$. The integration of the joint rates gives joint angles, θ, a 6X1 vector. This is then supplied to the manipulator motion controller that moves the manipulator to the computed joint angles positions. In the next cycle, the Jacobian is updated with the new joint angles and the cycle repeats. This way the manipulator completes the trajectory.

A point to be mentioned here is that due to the inherent non-linearities of the manipulator, it does not end up at the computed joint angle values. Instead there is an error and the actual joint angle positions that the manipulator ends up at are different from the computed ones. This error needs to be corrected otherwise a wrong Jacobian is computed at the next time step (due to wrong joint angles) and the error in trajectory generated keeps on accumulating. A way to correct this error has been explained in detail in chapter 6 on laser based telerobotic control.

Chapter 5: Arm Dexterity Improvement

In this chapter we introduce the concept of singularity to the reader and why singularity avoidance is necessary. We discuss the types of singularity avoidance techniques and give a detailed discussion on the singularity-robust (SR) inverse method to avoid singularity. Then we present the heuristic approach adopted to evaluate the parameters necessary in the implementation of the SR inverse of the Jacobian.

5.1. Introduction to Arm Dexterity

Arm dexterity is the kinematic ability of a robotic arm to be able to move in 3D space in smooth and stable manner and to be able to achieve different configurations in 3D space. The main parameters that define the dexterity of the arm are its singularity, joint limits and number of joints of the arm [13]. In this chapter we deal with the implementation of singularity avoidance techniques. Specifically we describe the theory behind singularity-robust inverse of the Jacobian and its implementation. This was undertaken to achieve smooth and stable motion of the PUMA when it approached singular configurations or reached workspace limits. As the PUMA is a six DOF arm, it is non-redundant and thus is unable to move any of its joints when the end effector is locked to a point in 3D space.

5.2. Singularity

Singularity is a kinematic property of a robotic arm. The arm is said to be at a singular configuration when very large joint velocities are required to move the end effector by small amounts in 3D Cartesian space. These configurations are called singular configurations. When the arm is at or near singular configurations, the arm movement is dangerous as the joint velocities are very high. The arm movement is thus unpredictable and unstable. The singular configurations should be thus avoided.

One way to determine whether the arm has reached a singular configuration is to check the value of the determinant of the Jacobian of the arm. At or near singular configurations, the determinant of the Jacobian of the arm is zero or very close to zero.

Another way to determine whether the arm has reached singularity is from evaluating the manipulability measure [28]. The manipulability measure is zero at singular points and increases as the arm moves away from the singularity. The manipulability measure is given by,

$$\omega = \sqrt{\det[J(\theta).J(\theta)^T]} \quad \text{.......} (33)$$

where *J(θ)* is the Jacobian matrix at joint angles *(θ)*. The manipulability measure method of determining singularity is especially useful for redundant arms for which the computation of the determinant of the Jacobian is not possible as they are non-square matrices and $[J(\theta).J(\theta)^T]$ always gives a square matrix. Physically the manipulability measure is a representation of the volume of an ellipsoid in the six dimensional Euclidean space as shown in the figure below [28]. The larger the manipulability measure or the larger the volume of the ellipsoid at a particular configuration of the arm, the farther the arm is away from singularity.

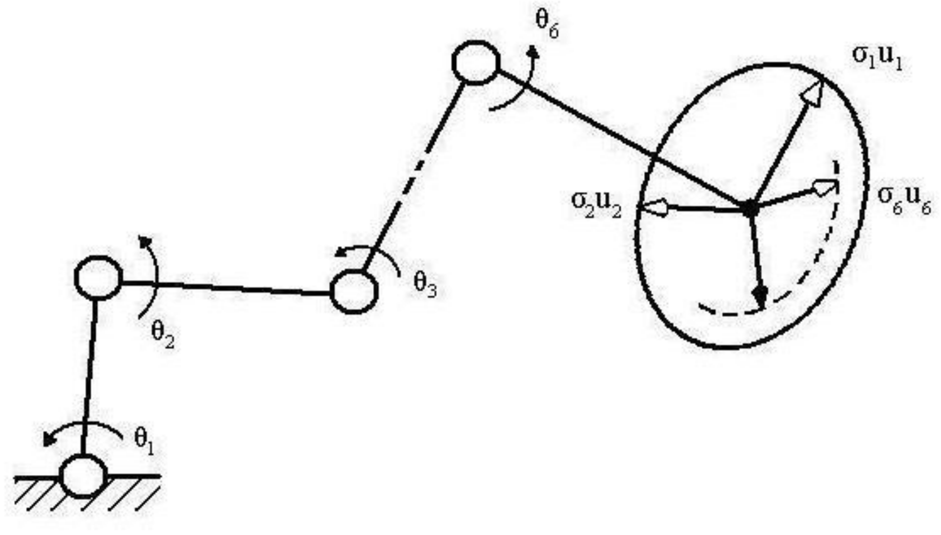


Figure 24: Manipulability ellipsoid [28]

One technique of singularity avoidance is to use pseudo inverse of the Jacobian [29]instead of simply the inverse of the Jacobian. The pseudo inverse of the Jacobian is given by the following formula:

$$J^* = J^T . (J.J^T)^{-1} \qquad (34)$$

At singular configurations the determinant of the pseudo inverse of the Jacobian is not zero and hence its inverse is defined. So a solution to inverse kinematics exists and the arm does not experience any high joint velocities. The joint rates can then be computed using:

$$\dot{\theta} = J^* . \dot{X} \qquad (35)$$

This is a least squares solution with a minimum norm for equation (20), viz. $\dot{\theta}$ satisfies (min || $\dot{\theta}$ ||) among all $\dot{\theta}$ that fulfill (min || $\dot{X} - J(\theta). \dot{\theta}$ ||), where ||*M*|| denotes Euclidean norm of *M* [29]. The difference between J^* and J^{-1} is that J^* is defined at singular points whereas J^{-1} is not. However, the shift from J^{-1} to J^* is discontinuous and this may result in an unstable robot motion when approaching singular configurations. Also, the exactness of the solution is more important here than the feasibility of the solution. This is because (min || $\dot{X} - J(\theta).\,\dot{\theta}$ ||) has a higher hierarchy than (min || $\dot{\theta}$ ||). As a result around singular points, the pseudo inverse J^* and the inverse J^{-1} are both forced to find the exact solution. This gives large joint rates around singular points [29]. Therefore this method is not preferred.

The other technique that is used for singularity avoidance is the singularity-robust Inverse or SR inverse of the Jacobian. Unlike the pseudo inverse of the Jacobian, the SR inverse of the Jacobian is a robust method and prevents the arm from reaching high joint rates at and around singular points. Like pseudo inverse, it also weighs between the exactness of the solution and feasibility of the solution. However, unlike pseudo inverse it computes the solution by simultaneously evaluating the exactness and feasibility. Thus the SR inverse is an approximate method as it gives approximate solutions to desired trajectories at and near singularities. The joint position error is increased at the expense of reducing joint velocities at and near singularities.

5.3. Singularity-Robust Inverse of the Jacobian

As stated above, the SR inverse of the Jacobian compromises the accuracy of the trajectory generated while keeping the joint velocities from reaching high values. Thus if singularity is approached while following a trajectory, the arm will minimally deviate from the commanded trajectory instead of attaining a singular configuration. It gets back on the trajectory once the point of singularity is passed. This is in case of an internal singularity i.e. a singular point that is inside the reachable workspace of the robotic arm. In case of external singularities i.e. when the arm is extending out of its workspace while following a trajectory that goes out of the robot workspace, the arm continues to move in its extended configuration along the boundary of the workspace with reasonable joint velocities minimizing the end effector velocity. Thus in both cases the arm generates erroneous motion to avoid singular points. In fact in the case of an internal singularity, the algorithm treats the singular point as an obstacle and goes around the obstacle as though it is avoiding the obstacle.

The general way of computing the joint angles from Cartesian trajectory is by using equation (20. However, at and around singular configurations when the determinant of the Jacobian is zero or near zero, J^{-1} tends to infinity and this results in very high joint velocities. To counter this problem we use, the SR inverse of the Jacobian instead of J^{-1}. The SR inverse of the Jacobian is given by [29],

$$J^{*} = J^{T}.(J.J^{T} + k.I_{6})^{-1} \quad (36)$$

where J is the Jacobian, I_6 is the 6X6 Identity matrix and k is the scale factor. k is the factor that weighs between the exactness and feasibility. The value of k should not be too high and should not be too low. If it is too high then the joint error will be too high and if it is too low then the joint rate will be too high. Also when k is zero the SR inverse of the Jacobian is equal to the general inverse of the Jacobian. The scale factor should have a zero value when it is far from singular positions because it is desired not to have inaccuracies in the trajectory execution at points far away from singular positions. For this an adaptive value of k should be chosen. When the end effector is far from singular points the value of k should be zero and once the end effector enters a neighborhood of points resulting in singularity, the value of k should steadily increase. Thus a parameter that changes as the arm moves away from singularity should be used. Manipulability measure introduced in the previous section is such a quantity. It is given by equation (33):

The manipulability measure [28] as the name suggests evaluates the kinematic ability of robot manipulators. It also gives an indication of how far the arm is from singular points. The manipulability measure is zero at singular points and its value increases as the arm moves away from singular points. They are non-negative. As a result the manipulability measure can be considered as a kind of distance from singular points. Hence it can be used for determining the value of scale factor k adaptively.

The adaptive value of k is chosen according to the following equation:

$k = k_0 * (1 - \frac{w}{w_0})^2$ for w < w_0

$k = 0$ for w >= w_0

(37)

Here w_0 is the manipulability measure at the boundary of the neighborhood of the singular point and k_0 is the scale factor at the singular point. Thus w_0 defines the boundary of the neighborhood of the singular point. As the values of w_0 and k_0 determine the value of k, they have to be chosen such that they do not induce high joint rates and at the same time do not induce high errors. We used heuristics to determine the values of w_0 and k_0 this is demonstrated in the next section. The values of w_0 and k_0 obtained were 0.014 and 0.025 respectively. The following figure summarizes the above discussion.

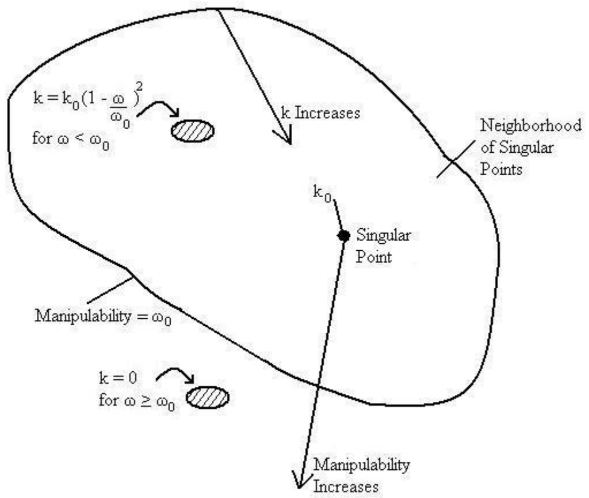

$k = k_0(1 - \frac{\omega}{\omega_0})^2$
for $\omega < \omega_0$
k Increases
Neighborhood of Singular Points
k_0
Singular Point
Manipulability = ω_0
k = 0
for $\omega \geq \omega_0$
Manipulability Increases

Figure 25: Relation between points of singular configurations, manipulability and scale factor

5.4. Implementation of the SR Inverse of Jacobian

As mentioned above, the values of k_0 and w_0 were determined from heuristics ensuring that we get an optimal value of k which would ultimately result in an optimal match between feasibility and accuracy. Matlab was used for determining the values of k_0 and w_0. A trajectory with an external singularity was executed in simulation and one of the two, k_0 and w_0 was varied keeping the other constant. A graph of manipulability v/s time was generated and from the observations in the graph, the values of k_0 and w_0 were finalized. The steps are explained below:

1. A random value of w_0 was chosen, w_0 = 0.034 and k_0 was varied from 0.0014 to 0.014 in the steps of 0.001. We must remember from equation (26) that since k is directly proportional to k_0, too high a value of k_0 will result in highly inaccurate result whereas too low a value of k_0 will result in high joint rates. The manipulability v/s time graph for the first step is shown in the following figure.

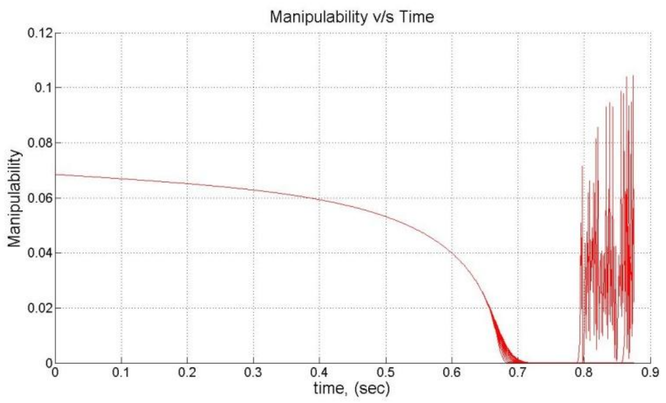


Figure 26: *k0* varies from 0.0014 to 0.014 in steps of 0.001, *w0* = 0.034

From this graph we can see that the arm shows unstable behavior at around 0.8 secs. This indicates that the value of k_0 is very low and must be increased.

2. So we increase the value of k_0 and make it vary from 0.0054 to 0.014 in the steps of 0.001 while keeping w_0 = 0.034.

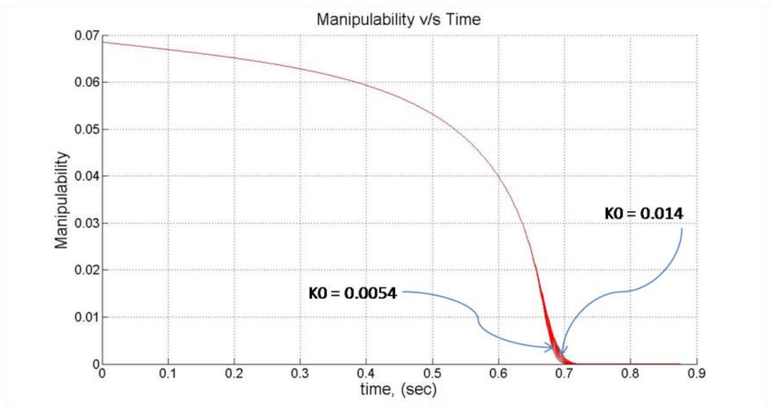


Figure 27: *k0* varies from 0.0054 to 0.014 in steps of 0.001, *w0* = 0.034

From this graph we observe that the value of k_0 is reasonably good as the graph is smooth. However, we fix the value of k_0 at 0.054 as 0.0054 is too low and it did cause high velocities in the joints. Next we keep k_0 constant at 0.054 and vary w_0 until we arrive at a reasonably good value for it. Though, w_0 at 0.034 looks good we must try to reduce it as much as possible. This is because as w_0 represents the boundary of the neighborhood of the singular point a large value of w_0 would mean large neighborhood and this would imply more errors in the followed trajectory. This is also evident by observing equation (25).

3. So we start with varying w_0 from 0.000034 to 0.00034 with steps of 0.00001 while keeping k_0 at 0.054.

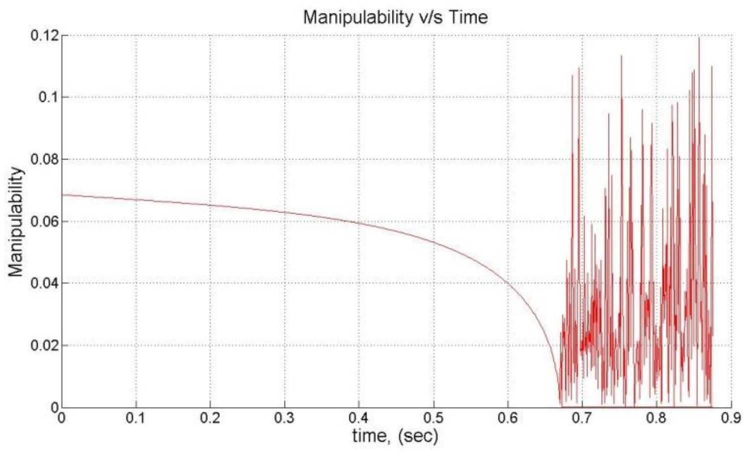


Figure 28: *w0* varies from 0.000034 to 0.00034 in steps of 0.00001, *k0* = 0.054

As the value of w_0 was too low, high joint velocities were observed in the arm near singular points. So we increased the value of w_0.

4. We varied w_0 from 0.00034 to 0.0034 with steps of 0.0001 while keeping k_0 at 0.054.

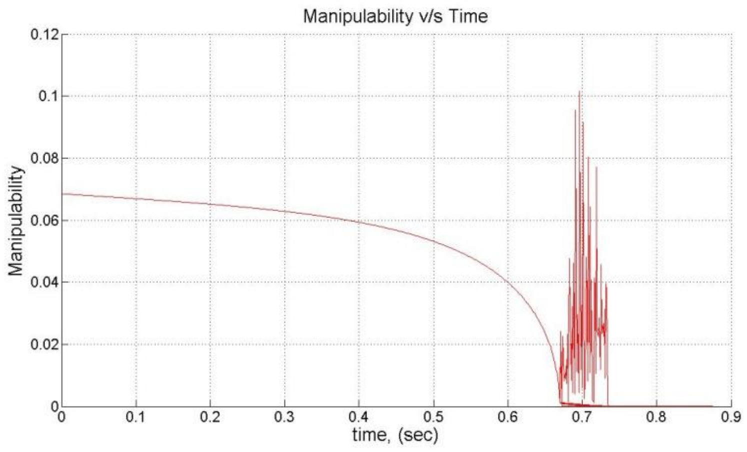


Figure 29: *w0* varies from 0.00034 to 0.0034 in steps of 0.0001, *k0* = 0.054

We observed that the joints were reaching high joint rates and so the value of w_0 needed to be increased further.

5. Next we increased the value of w_0 from 0.0034 to 0.034 with steps of 0.001 while keeping k_0 at 0.054.

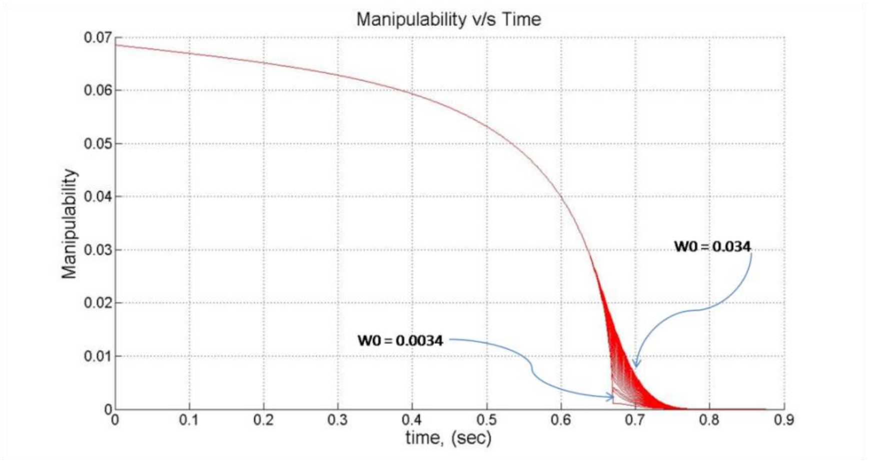


Figure 30: *w0* varies from 0.0034 to 0.034 in steps of 0.001, *k0* = 0.054

It was observed that w_0 = 0.0034 was too low not to induce high joint velocities near singular points. This is evident from the sharp change in the slope of the line when w_0 = 0.0034. Also, 0.034 is too high a value for w_0. So we choose a range with maximum value lower than 0.034 and minimum value higher than 0.0034 but near 0.034.

6. We vary w_0 from 0.01 to 0.02 with steps of 0.001 while keeping k_0 at 0.054.

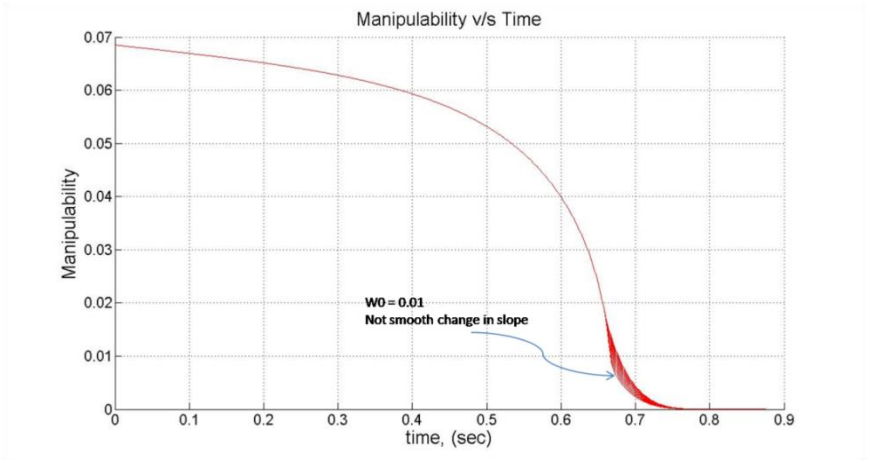


Figure 31: *w0* varies from 0.01 to 0.02 in steps of 0.001, *k0* = 0.054

We finalize the value of w_0 as 0.014 as this results in a smooth curve. Next we refine the value of k_0 so that it is compatible with w_0 = 0.014 and we get an optimized value. Though our value of k_0 was finalized at 0.054 we reduce it so we get more accurate result.

7. We vary k_0 from 0.01 to 0.02 in steps of 0.001 while keeping w_0 constant at 0.014.

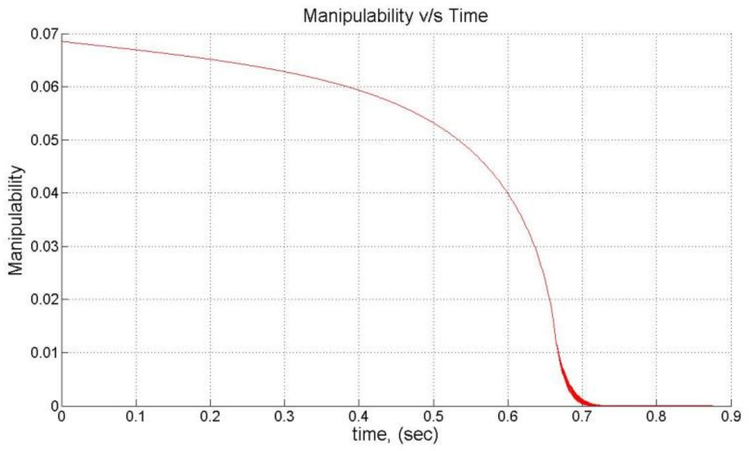


Figure 32: *k0* varies from 0.01 to 0.02 in steps of 0.001, *w0* = 0.014

As we obtain a smooth curve, we finalize the value of k_0 at 0.025. Thus by continuous refinement we are able to finalize the values of k_0 and w_0 at 0.025 and 0.014 respectively. These values are optimum as they provide reasonably accurate solution as well as prevent the joint velocities from reaching excessively high values. Thus they provide a good tradeoff between accuracy and feasibility of the solution. The following figure summarizes the relationship between manipulability, accuracy and feasibility. It is a magnified view of the previous figure in which k_0 was varied from 0.01 to 0.02 in steps of 0.001 and w_0 was held constant at 0.014. From the figure we see that, for lower values k_0 and w_0 which ultimately result in lower value of scale factor k (from equation 37). This results in the high joint velocities. This is observed from the sharp changes in slope (figure 30) and unstable behavior (figure 28 and 29). With high values of these parameters we get inaccurate results. This is observed from the continuous separation of the curve from the time axis as the values of the parameters increase (figure 30, 31 and 32).

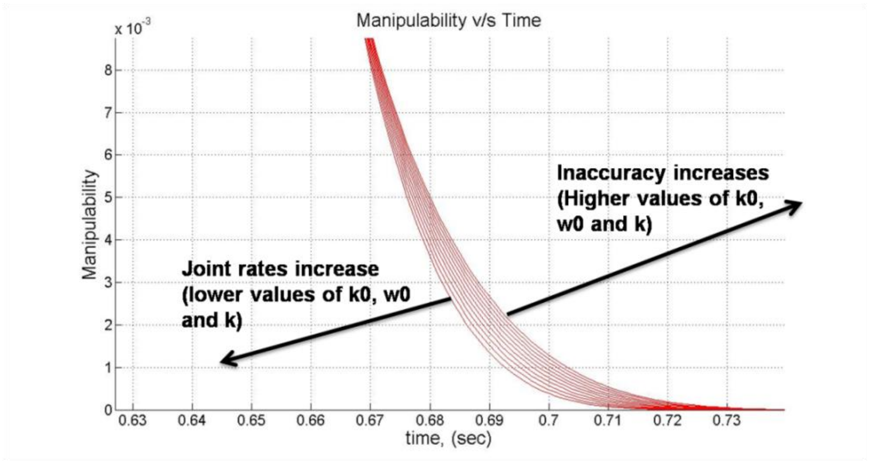


Figure 33: Graphical relationship between feasibility, accuracy and manipulability

Thus we use the values of k_0 and w_0 obtained from the optimization process above to determine the adaptive value of scale factor k. Using this value and equation 23, we determine the value of the SR inverse of the Jacobian which is then used to determine the joint rates. The result is that singular configurations are avoided by temporarily deviating the end effector away from the desired trajectory. The simulation results of the application are given in the results chapter.

Chapter 6: Laser based Telerobotic Control

This chapter is the central chapter of the thesis. Here we describe the various laser based capabilities that the reader was introduced to in the chapter 1. These laser based capabilities along with machine intelligence generate trajectories, virtual constraints and virtual geometries that assist the user either by executing trajectories autonomously or assisting the user along certain trajectories as the user teleoperates. The information from laser along with machine intelligence also generates 3D environment information and task plans which assist the user in executing the task. The assistance is mostly in the form of Traded Control where the human is involved in making high level task decisions and task planning and the computer executes tasks autonomously for the most part while relinquishing control to the human when needed. There is also some assistance for executing tasks in teleoperation based on virtual constraints and motion scaling again based on information from the laser.

In this chapter we describe the concepts related to the various laser based capabilities, the kinematic and mathematical formulations involved in the implementation of these capabilities, the specific user interaction that takes place while interacting with the system to execute a task based on assistance from laser and the application of the laser based features in the execution of a task. We will also describe how the user gets assistance in the specific laser based capabilities.

The conventions for representing the kinematics are adopted from John Craig [24]. The 4X4 transformation matrices are written in a split form comprising of a rotation component and a translation component as $[R\ |P]$ where 'R' represents the 3X3 rotation component and 'P' represents 3X1 translation vector. The last row of the 4X4 transformation matrix is dropped because it is the same for every configuration of the arm and is [0 0 0 1]. The subscript 'O' is used to represent Omni and ‘P’ represents PUMA. The second level subscript ‘1’ is used to represent current (previous) time step and ‘2’ represents next (current) time step.

The reader is suggested to go through the chapter 1 as it summarizes the details mentioned in this chapter without going into the technical details. Also the kinematic and mathematical formulations are an extension of the robot kinematic concepts described in chapter 4.

6.1. Unassisted Telerobotic Control

Unassisted telerobotic control is the direct control of the remote manipulator, PUMA by the human user using the master device, Omni. As there is no computer intermediation involved, it is also called teleoperation control. The user does not get any assistance from a sensor or from machine intelligence while teleoperating the arm in this fashion. The movements from the Omni are directly transferred to the PUMA.

In order to operate the system in this mode, the user needs to engage the Omni with the PUMA. By default the Omni and the PUMA are disengaged. This is for the safety of the PUMA and elements in its workspace. Teleoperation is also the default mode of control when the system starts as here the PUMA is completely under human control and there is no computer control. This is also from the safety point of view. To engage the PUMA with the Omni, the user makes use of the buttons on the Omni stylus and the virtual environment user interface shown in the figure 19. Initially the Omni is disengaged and moves in free space as the user moves it. The movements of the 3D cursor (in the form of a sphere) correspond to these movements. To engage the PUMA the user touches the 3D cursor with the cube, receives force feedback and then presses the white Omni stylus button. Once this is done the cube sticks to the 3D cursor and the PUMA moves with the Omni. The PUMA remains engaged as long as the white button is pressed with the cube engaged with 3D cursor. This is similar to the click-and-drag feature of graphical user interfaces except that the PUMA will move as long as the Omni white button is kept pressed and the 3D cursor and cube move. To disengage the user simply releases the white Omni stylus button. Then when the user indexes (Indexing is explained in chapter 3, section 3.6) the Omni with its blue stylus button, and reengages the PUMA, the user is able to teleoperate the PUMA again. The user need not index the PUMA while indexing the Omni because the transfer of motion from the Omni to the PUMA is relative. This means that instead of transferring the absolute values of motion co-ordinates from the Omni to the PUMA, the difference in the Omni co-ordinates between current and previous positions are transferred to the PUMA and the PUMA motion is incremented by this difference [14].

The main problem with the current implementation [14] was that improper information was passed to the PUMA from the Omni and the mapping of Omni co-ordinates from Omni to PUMA was improper which resulted in unrealistic teleoperation. To improve teleoperation consecutive transformation matrices of the Omni tip were computed and were sent to the QNX controller of PUMA where they were mapped to the PUMA base co-ordinates. These were sent via the Ethernet port. The difference in position between the two consecutive incremental transformation matrices and difference in orientation was added to the PUMA current position and orientation which resulted in PUMA motion in teleoperation. This implementation in algorithmic form is delineated below:

1. Determine consecutive Omni tip transformation matrices as the Omni is being manipulated by user using forward kinematics on the Omni. $[R_{o_1}\ |P_{o_1}]$ and $[R_{o_2}\ |P_{o_2}]$
2. Send the transformation matrices computed from step 1 to the QNX controller.
3. Transform the Omni tip transformation matrices computed from Omni base co-ordinates to PUMA base co-ordinates by the following mapping formula:

$$(X_P, Y_P, Z_P) = (x_o, y_o, z_o) * M \quad (38)$$

where $M = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$. Diagrammatic representation of the mapping is shown in the figures 34 and 35.

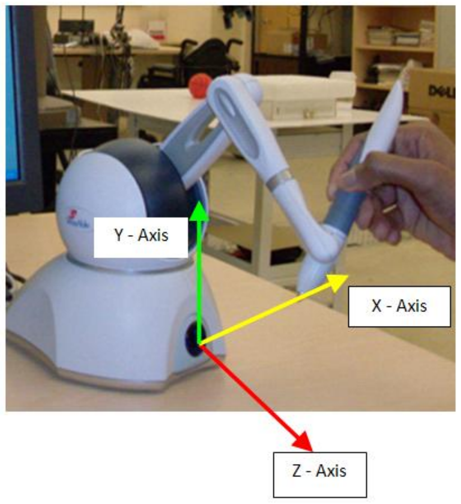


Figure 34: Omni base Cartesian co-ordinate system

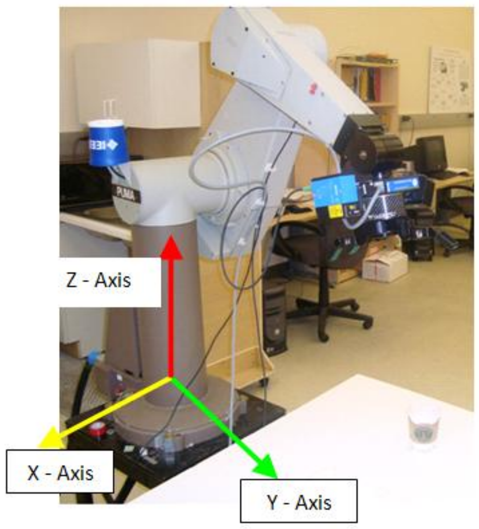


Figure 35: PUMA base Cartesian co-ordinate system

4. Determine the current end effector transformation matrix of the PUMA by forward kinematics $[R_{P_1}\,|P_{P_1}]$.

5. Determine the change in the rotation, *dR*, based on the two consecutive Omni transformation matrices mapped to the PUMA:

   $R_2 = R_1 * dR$

   $dR = R_1^{-1} * R_2$

   $dR = R_1^{T} * R_2$ (since R_1 is an orthogonal matrix)

6. Determine the change in position, *dP*, again based on the two consecutive Omni transformation matrices mapped to the PUMA:

   $dP = P_2 - P_1$

7. Form the differential transformation matrix, *dT* from the differential rotation, *dR* and differential translation, *dP* components.

   dT = [dR |dP]

8. Determine the transformation matrix for the PUMA end effector for the next time step

   $[R_{P_2}\,|P_{P_2}] = [R_{P_1}\,|P_{P_1}]$ * dT

9. Perform inverse kinematics using Resolved Rate algorithm to get the new joints angles for the PUMA, $[\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6]$.

The teleoperation concept is shown in the following figure.

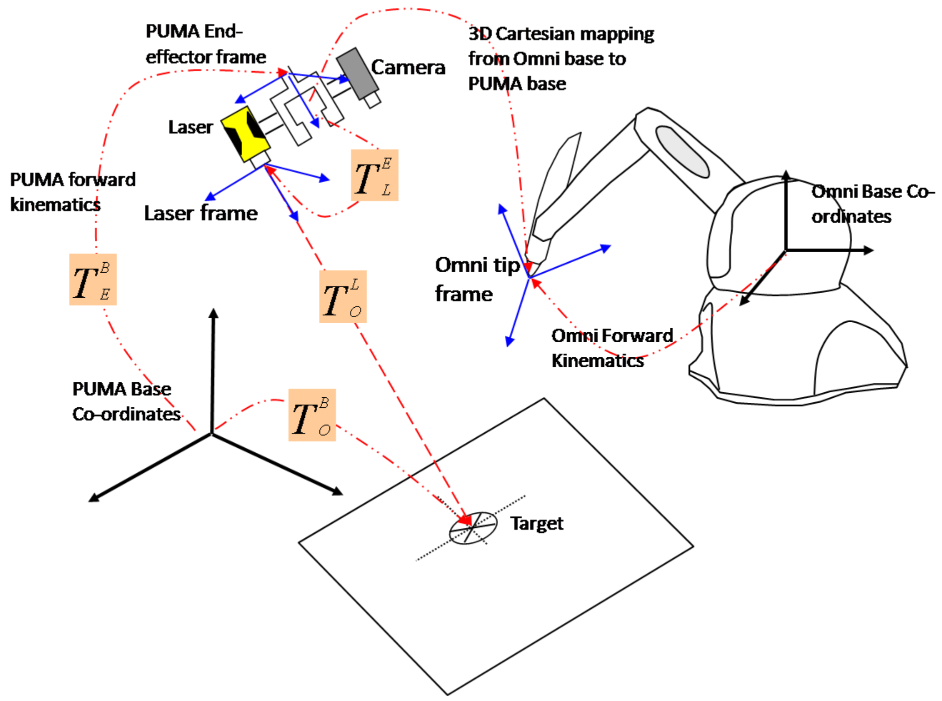


Figure 36: Teleoperation control and autonomous control concept

Another improvement that was added to teleoperation was removal of the jerks in teleoperation which were experienced whenever the PUMA was reengaged after indexing the Omni. The primary reason for the occurrence of these jerks was that the Omni transformation matrices were computed and transmitted to QNX even when the PUMA was disengaged. This lead to accumulation of transformation matrices in the buffer and when they were suddenly applied to the PUMA it lead to a jerky motion.

Moreover, the closed form method of computing joint rates was eliminated and Resolved Rate method was implemented. This was done as Resolved Rate method is applicable to any arm once its Jacobian is known whereas closed form method is only applicable for a specific arm with its specific equations. The closed form method cannot be used for arms that do not have closed form solutions like redundant arms.

The unassisted telerobotic control can be used by itself to execute a task. The user teleoperates the PUMA with the Omni based on visual feedback and can complete tasks such as pick-and-place. This control is also used in Traded Control mode for the human component of task. The human teleoperates the PUMA with the Omni and this way is able to point the laser pointer to various objects or surfaces of interest and record the co-ordinates of these points for the autonomous component of Traded Control. The ease with which the user is able to set the arm up for autonomous task execution by pointing with laser is due to the fact that the laser is mounted to the end effector in such a way that its rotation is same as the joint 6 but it is offset from it along the Cartesian co-ordinate axes. Thus the laser beam is parallel to the z-axis of the joint 6. This is shown in the figure 37. In this way the teleoperation mode is also a means to the high level decision making in task execution as the user selects targets and destination points in teleoperation and the robot then does the low level task autonomously. The user also uses teleoperation to make fine adjustments to the arm while grasping objects and dropping objects in say pick-and-place tasks so that the hand is in proper orientation for the activity.

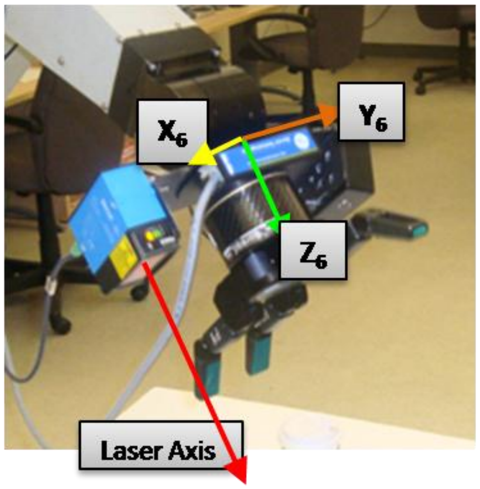


Figure 37: Mounting of laser sensor on PUMA end effector

6.2. Laser based Trajectory Generation Capability

Using information from the laser and PUMA forward kinematics we generate a linear trajectory from the current location of the PUMA end effector to the point the laser is pointing to. This linear trajectory can be used in a number of ways to assist the user in doing a task. The generated trajectory can be executed by the robot when the user commands it to. This autonomous execution of the trajectory can be used with Traded Control mode for the computer controlled portion of the task. It can provide assistance to the user in scaled teleoperation where the component of the users' motion in the direction of the trajectory is scaled up whereas the other components are scaled down thus giving the user a kinesthetic sense of the desired trajectory. The user will see less motion of the PUMA when the user is deviating away from the trajectory. It can also be used in creating a force feedback based virtual constraint along the trajectory. Force feedback capabilities of the Omni are utilized for the purpose. This will keep the user on the trajectory by attractive forces and not allow him to deviate. Such forms of assistance can be used in moving towards a target in teleoperation or while opening a drawer along a linear trajectory. The autonomous trajectory generation and execution is also used in laser based path planning and execution where small segments of the path are linear trajectories. The manner in which the linear trajectory is generated from the laser range information and provides assistance to the user either in autonomous execution, scaled teleoperation or virtual constraint based teleoperation is the novelty of this work.

6.2.1. Autonomous Trajectory Generation

As mentioned earlier the system generates a linear trajectory using the laser data and PUMA forward kinematics from the current location of the PUMA to the point determined by the laser pointer. Initially the user teleoperates the PUMA via the Omni and this way moves the laser pointer of the beam emerging from the laser sensor mounted on the PUMA end effector from one object to another. Once the user decides on a point of interest as the target point, the user ceases to teleoperate and the laser pointer remains stationary on the surface of the object. Then the user presses the ‘A’ key on the keyboard of the PC interfaced to the Omni. Once this is done the PUMA end effector traverses along the trajectory autonomously. It does however stop at a certain threshold distance of 20 cm from the laser pointer position along the trajectory to avoid crashing into the target. This demonstrates Traded Control where the human relinquishes the control to the computer after determining the target point and the computer takes over the control when asked to do so and retains control until the task is completed. It also demonstrates the merging of human decision making and high level task execution with robot computational and precision task execution capability. The human is involved in making high level decisions like choosing target objects whereas the machine does the low level, activities like generating and executing the trajectory. Generating and executing a trajectory also involves computations from the machine and precision in task execution which is done by the machine. Moreover, the trajectory can be generated from PUMA current location to any target by simply teleoperating and pointing the laser to a target. This demonstrates the simplicity of interface, ease of use and the ability to maneuver in unstructured environments. The conceptual model of the autonomous trajectory generation is shown in the figure 38 and 39. In this 3D model and all the following 3D models, the robotic arm shown is not PUMA but another arm. The alternative arm was acceptable even though the algorithms were applied on PUMA as these 3D models were for the demonstration of the concept only.

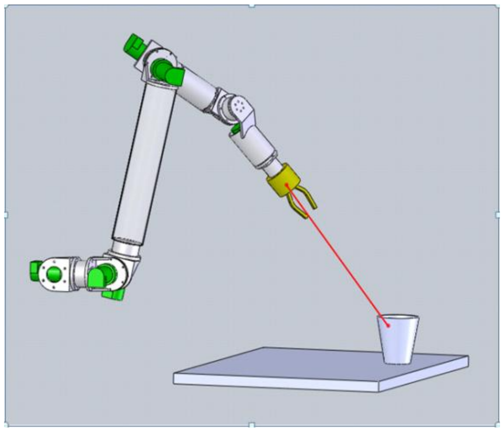


Figure 38: Autonomous trajectory generation - initial point

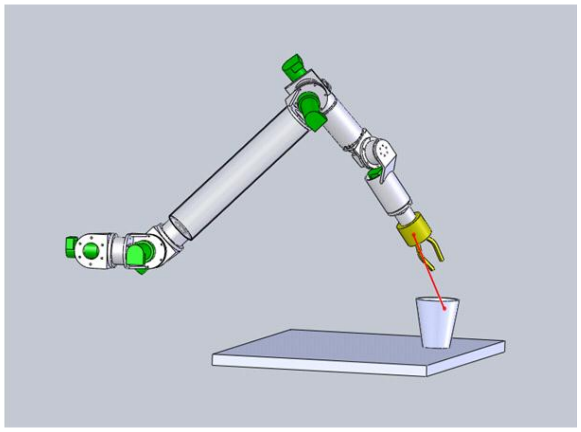


Figure 39: Autonomous trajectory generation - final point

When the user presses the ‘A’ key on the Omni PC keyboard this information is sent to the QNX machine via Ethernet where the trajectory generation algorithm and other motion control algorithms reside. Once this information is received by the QNX machine it triggers that part of the code that generates and executes trajectory based on laser information. For trajectory generation linear interpolation and Equivalent Angle-Axis method have been used along with Resolved Rate algorithm. When the initial and final points of a trajectory are determined via points, which are equally spaced intermediate points along the trajectory, need to be computed. These via points are in the form of transformation matrices that the end effector has to attain at each time step. It is these successive points that will form incremental velocity vectors that the Resolved Rate algorithm will use to compute joint rates at each succession. After via points are computed and stored in an array, they are read at a rate of 200 Hz and joint angles are determined which are fed to the torque generator at the same rate to cause the robot joints and hence the robot to move.

The positions for via points can be very simply computed by linear interpolation of the initial and final trajectory points. However, the rotation vectors cannot be simply computed by linear interpolation as not all of them will be necessarily orthogonal. Rotation matrices of the robot end effector have to be orthogonal. Therefore Equivalent Angle-Axis method is used to compute incremental rotations. The algorithm to generate a trajectory is given next. The inputs to the algorithm are the initial and final point transformation matrices and velocity of the robot. The output is joint angles computed at each time step (200 Hz) that are fed to the torque generator to execute the trajectory. The following algorithm is a general algorithm and will compute a trajectory for any source and destination points. The manner in which the laser information has been used to generate a trajectory will be explained later.

1. The inputs to the algorithm are the transformation matrix of the initial point (generally determined by forward kinematics) and that of the final point (determined in our case using range information from the laser). Both of these are with respect to the PUMA base frame.

$$T_i = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad T_f = \begin{bmatrix} n_{x_f} & o_{x_f} & a_{x_f} & P_{x_f} \\ n_{y_f} & o_{y_f} & a_{y_f} & P_{y_f} \\ n_{z_f} & o_{z_f} & a_{z_f} & P_{z_f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Assume V_r as the linear velocity of the end effector and find the distance, D between the initial and the final points.

   $D_x = P_{x_f} - P_{x_i}$

   $D_y = P_{y_f} - P_{y_i}$ and $D = \sqrt{(D_x^2 + D_y^2 + D_z^2)}$

   $D_z = P_{z_f} - P_{z_i}$

3. Determine the total time t_f, which the end-effector would take for travelling between the initial and the final points.

   $t_f = V_r / D$

4. Assume the sampling time to be $dt = 0.001$ sec

5. Determine the no. of via points, $N = t_f / dt$

6. Generate the via points transformation matrices, $T_1, T_2, T_3, \ldots\ldots.., T_{n+1}$, where $T_i = T_1$ and $T_f = T_{n+1}$ by calling the sub-routine 'Via Points Transformation Matrices'. They are stored in an array.

7. Initialize the old-joint-angle vector, q_{old} and current-joint-angle-vector, q.

$q = q_{old} = q_i$ *or* $\begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\\ \theta_4\\ \theta_5\\ \theta_6\end{bmatrix} = \begin{bmatrix}\theta_{old_1}\\ \theta_{old_2}\\ \theta_{old_3}\\ \theta_{old_4}\\ \theta_{old_5}\\ \theta_{old_6}\end{bmatrix} = \begin{bmatrix}\theta_{1_i}\\ \theta_{2_i}\\ \theta_{3_i}\\ \theta_{4_i}\\ \theta_{5_i}\\ \theta_{6_i}\end{bmatrix}$

8. Take the first transformation matrix which is the initial point of the trajectory.

9. Take the next transformation matrix from the array.

10. Compute *dv* = $\begin{bmatrix}dv_1\\ dv_2\\ dv_3\end{bmatrix} = \begin{bmatrix}P_{x_{k+1}} - P_{x_k}\\ P_{y_{k+1}} - P_{y_k}\\ P_{z_{k+1}} - P_{z_k}\end{bmatrix}$

11. Compute *dw* = $\begin{bmatrix}dw_1\\ dw_2\\ dw_3\end{bmatrix}$

$$= \begin{bmatrix}0.5 * (o_{y_k} * o_{z_{k+1}} - o_{z_k} * o_{y_{k+1}} + n_{y_k} * n_{z_{k+1}} - n_{z_k} * n_{y_{k+1}} + a_{y_k} * a_{z_{k+1}} - a_{z_k} * a_{y_{k+1}})\\ 0.5 * (o_{z_k} * o_{x_{k+1}} - o_{x_k} * o_{z_{k+1}} + n_{z_k} * n_{x_{k+1}} - n_{x_k} * n_{z_{k+1}} + a_{z_k} * a_{x_{k+1}} - a_{x_k} * a_{z_{k+1}})\\ 0.5 * (o_{x_k} * o_{y_{k+1}} - o_{y_k} * o_{x_{k+1}} + n_{x_k} * n_{y_{k+1}} - n_y * n_{x_{k+1}} + a_{x_k} * a_{y_{k+1}} - a_{y_k} * a_{x_{k+1}})\end{bmatrix}$$

12. The screw velocity becomes *dV* = $\begin{bmatrix}dv_1\\ dv_2\\ dv_3\\ dw_1\\ dw_2\\ dw_3\end{bmatrix}$ (These are actually positions and not velocities).

13. Compute the Jacobian with respect to the base, *Jac0* for the current joint angle vector, *q*.

14. Compute the Inverse of the Jacobian, *iJac0*

15. The change in the angular position of joints or joint angle difference vector is calculated.

q_{rate} = *iJac0 * dV* *or* $\begin{bmatrix}\theta_{rate_1}\\ \theta_{rate_2}\\ \theta_{rate_3}\\ \theta_{rate_4}\\ \theta_{rate_5}\\ \theta_{rate_6}\end{bmatrix}$ = *iJac0* * $\begin{bmatrix}dv_1\\ dv_2\\ dv_3\\ dw_1\\ dw_2\\ dw_3\end{bmatrix}$

16. New joint angle vector is computed from the current joint angle vector, q and the joint angle difference vector, q_{rate}.

$$q = q_{\,old} + q_{rate} \qquad \text{or} \qquad \begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\\ \theta_4\\ \theta_5\\ \theta_6\end{bmatrix} = \begin{bmatrix}\theta_{old_1}\\ \theta_{old_2}\\ \theta_{old_3}\\ \theta_{old_4}\\ \theta_{old_5}\\ \theta_{old_6}\end{bmatrix} + \begin{bmatrix}\theta_{rate_1}\\ \theta_{rate_2}\\ \theta_{rate_3}\\ \theta_{rate}\\ \theta_{rate_5}\\ \theta_{rate_6}\end{bmatrix}$$

17. Update the old-joint-angle vector, $q_{\,old}$ and the loop counter,

$$q_{\,old} = q \qquad or \qquad \begin{bmatrix}\theta_{old_1}\\ \theta_{old_2}\\ \theta_{old_3}\\ \theta_{old_4}\\ \theta_{old_5}\\ \theta_{old_6}\end{bmatrix} = \begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\\ \theta_4\\ \theta_5\\ \theta_6\end{bmatrix}$$

$k = k + 1$

18. Compute the current transformation matrix from forward kinematics. This now becomes the initial transformation matrix for the next iteration.

19. Go to step X and repeat the steps (loop between steps 9 and 18) until $k = N$, where N is the number of via points.

The sub-routine that computes the transformation matrices at via points is given next. The inputs to the sub-routine are the transformation matrices at the initial and the final points on the trajectory, T_i and T_f, and the no. of via points, N.

$$\text{Let, } T_i = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i}\\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i}\\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i}\\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } T_f = \begin{bmatrix} n_{x_f} & o_{x_f} & a_{x_f} & P_{x_f}\\ n_{y_f} & o_{y_f} & a_{y_f} & P_{y_f}\\ n_{z_f} & o_{z_f} & a_{z_f} & P_{z_f}\\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1. Compute the rotation matrix operator, R, between the initial and the final points.

$$R = \left(\begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i}\\ n_{y_i} & o_{y_i} & a_{y_i}\\ n_{z_i} & o_{z_i} & a_{z_i} \end{bmatrix}\right)^T * \begin{bmatrix} n_{x_f} & o_{x_f} & a_{x_f}\\ n_{y_f} & o_{y_f} & a_{y_f}\\ n_{z_f} & o_{z_f} & a_{z_f} \end{bmatrix}$$

$$= \begin{bmatrix} n_{x_r} & o_{x_r} & a_{x_r} \\ n_{y_r} & o_{y_r} & a_{y_r} \\ n_{z_r} & o_{z_r} & a_{z_r} \end{bmatrix}$$

2. Initialize the initial rotation matrix, R_i

$$R_i = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} \end{bmatrix}$$

3. Compute the angle for the end-effector rotation (by Equivalent Angle-Axis method).

$$\phi = \text{atan2}\left(\left(\sqrt{(o_{z_i} - a_{y_i})^2 + (a_{x_i} - n_{z_i})^2 + (n_{y_i} - o_{x_i})^2}\right), (n_{x_i} + o_{y_i} + a_{z_r} - 1)\right)$$

4. Let $st = \sin(\phi)$, $ct = \cos(\phi)$, $vt = 1 - \cos(\phi)$

5. To determine the axis of rotation of the end effector (which is necessary for determining the incremental rotation matrix), we assume *kx*, *ky* and *kz* to be the components of the axis vector.

6. If $\phi < 0.001$, then $kx = 0$, $ky = 0$, $kz = 1$

```
a.  Else if $\phi < (90° + 0.001)$ then
        $Kx = (o_{z_r} - a_{y_r})/(2*st)$,
        $Ky = (a_{x_r} - n_{z_r})/(2*st)$,
        $Kz = (n_{y_r} - o_{x_r})/(2*st)$
b.  Else
        $Kx = \pm(o_{z_r} - a_{y_r}) * \sqrt{(n_{x_r} - ct)/vt)}$
        $Ky = \text{sign}(a_{x_r} - n_{z_r}) * \sqrt{(o_{y_r} - ct)/vt)}$
        $Kz = \text{sign}(n_{y_r} - o_{x_r}) * \sqrt{(a_{z_r} - ct)/vt)}$
    i.  If (($kx > ky$) and ($kx > kz$)), then
        $Ky = (n_{y_r} + o_{x_r})/ (2*kx*vt)$
        $Kz = (a_{x_r} + n_{z_r}) / (2*kx*vt)$
    ii. Else if (($ky > kx$) and ($ky > kz$))
        $Kx = (n_{y_r} + o_{x_r})/ (2*ky*vt)$
        $Kz = (o_{z_r} + a_{y_r}) / (2*ky*vt)$
```

iii. Else $Kx = (a_{x_r} + n_{z_r}) / (2*kz*vt)$

$Ky = (o_{z_r} + a_{y_r}) / (2*kz*vt)$

7. Generate the vector (set of angles) a_t that contains the angular positions of the axis of the Equivalent Angle-Axis representation at via points by calling the sub-routine 'Incremental Angles for Angle-Axis Representation'. This is required for determining the incremental rotation matrices of the end effector transforms such that the rotation matrices are orthogonal.

8. To generate via points transformation matrices carry out the steps IX through XIV for all via points.

9. Compute the angular position of the axis, *da,* from the vector a_t:

   $da_n = a_{t_n} - a_{t_1}$

10. Compute the values, $sa = sin(da)$, $ca = cos(da)$, $va = 1 – cos(da)$

11. Compute the differential rotation matrices at the via points (differential wrt the rotation matrix at the initial point on the trajectory)

$$dR = \begin{bmatrix} kx^2 * va + ca & kx * ky * va - kz * sa & kx * kz * va + ky * sa \\ kx * ky * va + kz * sa & ky^2 * va + ca & ky * kz * va - kx * sa \\ kx * kz * va - ky * sa & ky * kz * va + kx * sa & kz^2 * va + ca \end{bmatrix}$$

12. Determine the position vectors for the initial and final points on the trajectory, X_i and X_f, from the initial and final transformation matrices, T_i and T_f

$$X_i = \begin{bmatrix} P_{x_i} \\ P_{y_i} \\ P_{z_i} \end{bmatrix} \quad \text{and} \quad X_f = \begin{bmatrix} P_{x_f} \\ P_{y_f} \\ P_{z_f} \end{bmatrix}$$

13. Compute the displacement vector from the initial point up till the point of consideration.

   $X = X_i * \frac{(N-i)}{N} + X_f * \frac{i}{N}$

14. Compute the transformation matrix at the point.

$$TT = \begin{bmatrix} Ri * dR & \cdots & & X(1) \\ \vdots & \ddots & & X(2) \\ & & & X(3) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

15. Return the set of transformation matrices to the calling sub-routine ‘Main Routine – Trajectory Generation’.

The sub-routine that calculates the incremental angles by which the axis of the Equivalent Angle-Axis representation should rotate to give orthogonal via points rotation matrices is given next.

The inputs to the sub-routine are the initial and final angles, ψ_i = 0 and ψ_f =ϕ, and the number of via points, *N*. The sub-routine returns the vector a_t that contains the angles values at each via point that the axis of the Equivalent Angle-Axis method should attain as the PUMA traverses the trajectory.

1. Determine the angle increments, $d\Psi = (\psi_f - \psi_i) / N$
2. Set the value of the first element of the vector a_t, $a_t[0] = d\Psi$
3. Repeat the step IV for all via points as counter i varies from 1 to (N-1) for each iteration
4. $a_t(i) = a_t(i-1) + d\Psi$
5. Return the vector a_t to the sub-routine that calculates via points transformation matrices.

The important point to be considered in the above algorithm is that the two main inputs to the algorithm i.e. the initial point transformation matrix and the final point transformation matrix are both with respect to the PUMA base frame. Now we describe how the laser information is used to generate trajectory. The distance from the laser sensor is used here along with frame transformation equations to determine the final point transformation matrix, the initial point transformation matrix being determined from forward kinematics. As mentioned, both have to be referenced to the PUMA base frame. There are two kinds of linear trajectories that can be generated with the laser. One trajectory type is from the laser source to the laser point on the target. The other is from the end effector to the laser point on the target. In the first type as the PUMA moves along the laser beam, the end effector does not end up over the target object due to the laser sensor offset from the end effector (refer figure 37). A second trajectory moves the end effector laterally to align the hand over the target. In the second type the laser offset from the PUMA end effector is accounted for in the transformation matrix equations so that the end effector ends up over the target.

In the first type of trajectory as the PUMA moves along the laser beam direction, we observe that the laser point on the target remains stationary. The transformation matrix equations were developed as explained here. Refer figure 36 with the explanation. We assume that the orientation of the end effector will remain the same when it reaches the target or in other words the orientation of the target frame is the same as that of the PUMA end effector frame when it is at the beginning of the trajectory. We already know that the orientation of the laser frame is the same as that of the PUMA end effector frame. This further suggests that the z-axis of the laser frame and the end effector frame are aligned or are parallel to each other. Also the target frame origin lies on the target along the laser line or along the laser frame z axis. In other words the target frame is displaced from the laser frame only along the z-axis of the laser frame. Let that distance be 'D'. Thus if we generate a linear trajectory from the end effector, along its z axis up to a distance 'D' along it, we would see the PUMA following a trajectory such that the laser pointer remains stationary on the target. Virtually the end effector is proceeding towards a target at distance 'D' along it. This is due to the parallel nature of the z axes mentioned. Moreover because the orientations of the end effector frame, the laser frame and the target frame are the same, the only variable in the transformation equations is the distance 'D' which is easily obtained from the laser sensor. The equation to determine the final point transformation matrix with respect to the PUMA base is given below:

$${}^{B}_{O}T = {}^{B}_{E}T * {}^{E}_{O}T \text{ ....} \quad (39)$$

Where,

1. ${}^{E}_{O}T$ = transformation matrix of the virtual target with respect to the PUMA end effector $= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and D is the distance given by the laser

2. $^{B}_{E}T$ = transformation matrix of end effector with respect to the base obtained by PUMA forward kinematics.

3. $^{B}_{O}T$ = transformation matrix of the virtual target with respect to the PUMA base frame.

$^{E}_{O}T$ gives the trajectory in what is known as Hand Co-ordinates. However, it is necessary to reference all the transformation matrices with respect to the base frame of the PUMA and hence the above equations are necessary.

In the second type of the trajectory we include the transformation matrix of the laser with respect to the end effector (which has Identity rotation component and is offset only along x, y and z axis) in the transformation equation in order to consider the laser offset. The laser offset co-ordinates are determined by physical measurements. Here again the only unknown is the distance of the target from the laser source, 'D', which is determined from the laser sensor. Now using the equation below we are able to compute the transformation matrix of the target with respect to the base frame of the PUMA.

$$^{B}_{O}T = {}^{B}_{E}T * {}^{E}_{L}T * {}^{L}_{O}T \quad \text{.......} \quad (40)$$

Where,

1. $^{L}_{O}T$ = transformation of the target frame with respect to the laser = $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D \\ 0 & 0 & 0 & 1 \end{bmatrix}$

2. $^{E}_{L}T$ = transformation matrix of the laser with respect to the end effector

= $\begin{bmatrix} 1 & 0 & 0 & -0.026 \\ 0 & 1 & 0 & -0.115 \\ 0 & 0 & 1 & 0.115 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

3. $^{B}_{E}T$ = transformation matrix of the end effector with respect to the base from forward kinematics.

Thus the initial and final points of the trajectory are computed from laser range data and these are then supplied to the algorithm described above to generate the trajectory.

We now give a note on the modification made to the trajectory generator to account for the error in robot motion which is due to its inherent non-linearities. The robot does not follow the laser generated trajectory but a slightly different trajectory. This is because at each time step the motors do not move precisely by the differential joint angle computed by the Resolved Rate algorithm which is due to their non-linear characteristics but by a slightly different angle. If the joint angles at successive time steps are computed from via points velocity vectors and the end effector continues to move erroneously the manipulator would follow an altogether different trajectory. In order to correct this, a self correction method has been employed. Figure 40 shows the laser generated trajectory and via points along the trajectory. Ideally the end effector should traverse though via points. But instead it moves to a point close to via point, represented by the blue dot, at each time step due to the error. So instead of computing the velocity vector for the next time step from the current and next via point, it is computed from the current POSE determined by forward kinematics and the next via point. Again the end effector moves to another blue dot very close to via point it should have moved to. This continues and the PUMA moves very close to the desired trajectory. This way at the end of each cycle forward kinematics is done to determine the current PUMA POSE and next via point is retrieved from the array to generate the velocity vector. This way the error due to the deviation of the PUMA is accounted for and the PUMA traverses the trajectory closely.

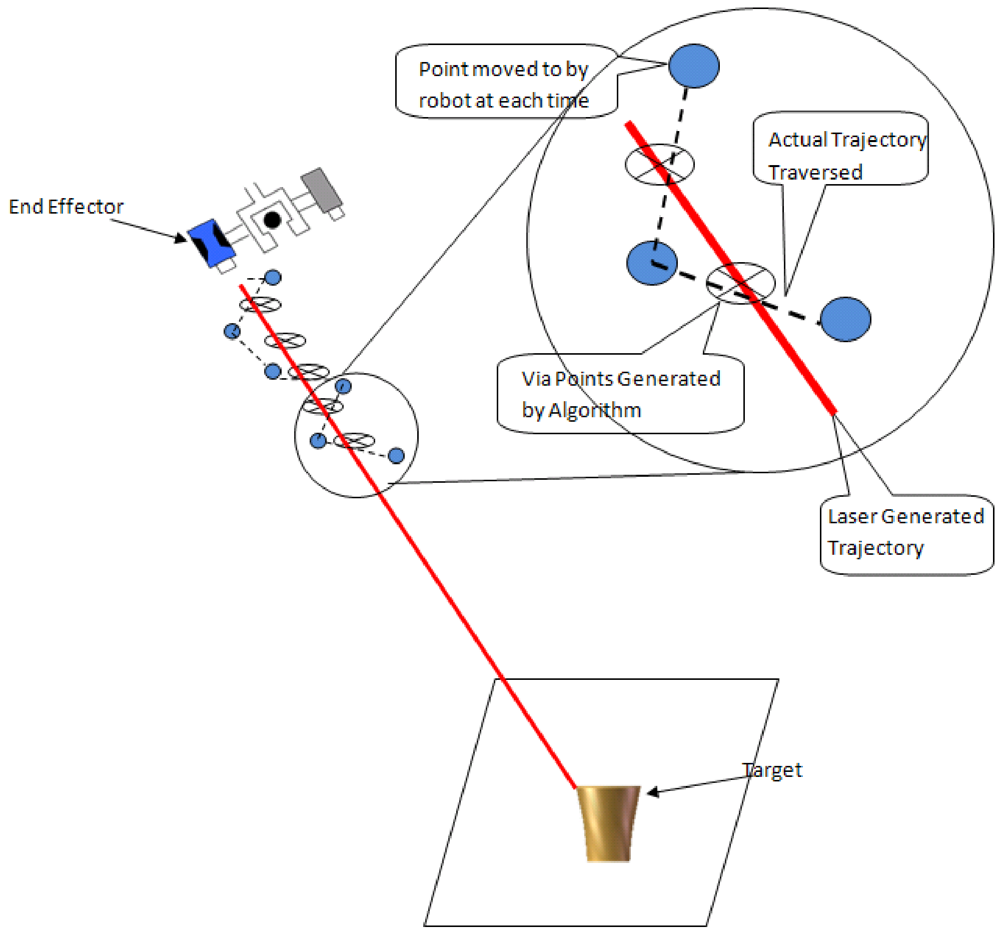

Point moved to by robot at each time
Actual Trajectory Traversed
End Effector
Via Points Generated by Algorithm
Laser Generated Trajectory
Target

Figure 40: Actual trajectory v/s ideal trajectory generated from laser data

Now we summarize the online trajectory generation using laser data and PUMA forward kinematics:

1. Initial and final points transformation matrices are computed from PUMA forward kinematics and laser data.
2. Via points are generated using linear interpolation and Equivalent Angle-Axis method.
3. At each time step (200 Hz) velocity vector is generated from current PUMA POSE to the next via point.
4. Joint angles are generated using Resolved Rate algorithm and are sent to the trajectory generator in each time step to execute the trajectory.

6.2.2. Scaled Teleoperation along the Trajectory

The trajectory generated from the laser data has been used to provide assistance to the user in scaled teleoperation. In scaled teleoperation, the component of the user's motion along the vector in the direction of the desired trajectory is scaled up while the components of the user's motion along vectors perpendicular to the desired trajectory are scaled down. The result is that the magnitude of the motion of the PUMA is proportional to magnitude of the motion of the Omni in the direction of the desired trajectory. Thus if the user is teleoperating the PUMA via the Omni in such a way that the direction of motion is along or close to the desired trajectory then the PUMA motion is amplified. However, if the Omni motion direction is in a direction perpendicular to the direction of the desired trajectory or close to this direction, the PUMA motion is attenuated. This method of teleoperation gives the user an idea of whether the user is teleoperating in the right direction towards the target or whether the user is deviating from the desired trajectory. The amplification and attenuation in the motion of the remote arm that the user observes gives the user this idea. Thus scaled teleoperation is a way of assisting the user in reaching the target faster and in a more accurate fashion. Here we use the trajectory generated by the laser as the reference trajectory and perform vector algebra on the incremental Omni motions to provide scaled teleoperation to the user.

From the user interface point of view, the user teleoperates in unassisted form and points the laser to the desired target. Then the user presses the 'S' key on the Omni PC keyboard. This information goes to the PUMA PC via Ethernet where the trajectory is generated from the laser distance and PUMA forward kinematics. The trajectory generated here is not the same as that generated for the autonomous trajectory execution as we only need the direction of the trajectory and do not need to execute the trajectory. The user will reach the target in teleoperation. Therefore, we would not need to generate via points and also would not need to execute the Resolved Rate algorithm. The determination of the unit vector that represents the direction along the laser generated trajectory is explained in the next paragraph. The reference trajectory is generated online in a matter of milliseconds. Then when the user engages the PUMA for teleoperation vector algebra is performed on the incremental Omni motion vector mapped to the PUMA base frame to generate a new motion vector that is amplified or attenuated according to the direction of the Omni motion vector. It is this vector that the PUMA is commanded to move. This cycle of the incremental motion vectors being passed to the PUMA and vector algebra being performed on them based on the laser generated reference trajectory to determine PUMA motion goes on until the 'S' key is pressed. When this happens the control becomes the direct teleoperation mode. While the 'S' key is pressed the indexing capability is still active. Also there is no scaling as far as the orientation is concerned.

Let us now give the mathematics involved in laser based scaled teleoperation. The reader can refer to figure 41 for diagrammatic explanation. In the figure the third vector $\widehat{M\_vec}$ is not shown. We already know that Omni tip transformation matrices are passed from the Omni PC to the PUMA PC while the PUMA is engaged in teleoperation and then are mapped to the PUMA base frame. Let these two be as given below.

Let, $T_1 = \begin{bmatrix} n_{x_1} & o_{x_1} & a_{x_1} & P_{x_1} \\ n_{y_1} & o_{y_1} & a_{y_1} & P_{y_1} \\ n_{z_1} & o_{z_1} & a_{z_1} & P_{z_1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $T_2 = \begin{bmatrix} n_{x_2} & o_{x_2} & a_{x_2} & P_{x_2} \\ n_{y_2} & o_{y_2} & a_{y_2} & P_{y_2} \\ n_{z_2} & o_{z_2} & a_{z_2} & P_{z_2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

The motion vector that should be added to current PUMA position to move the PUMA in teleoperation is given as

$$\vec{A} = (P_{x_2} - P_{x_1})\hat{\imath} + (P_{y_2} - P_{y_1})\hat{\jmath} + (P_{z_2} - P_{z_1})\hat{k} \quad \text{.......} (41)$$

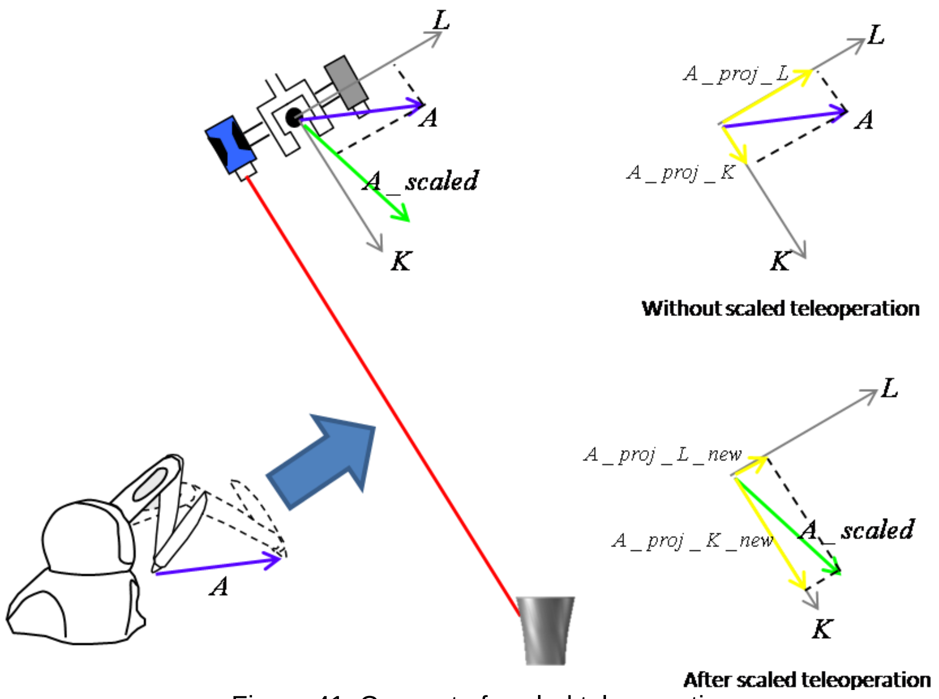


Figure 41: Concept of scaled teleoperation

However, to implement the scaled teleoperation, this vector will be amplified or attenuated based on whether its direction is along or close to the reference trajectory determined from laser data and PUMA forward kinematics or whether it is along or close to its perpendiculars. The reference vector that gives the desired trajectory is determined as follows. Let ${}^{i}_{f}T$ represent the transformation matrix of the target with respect to the end effector. Let ${}^{O}_{i}T$ represent the transformation matrix of the end effector with respect to the base. We have already seen that ${}^{O}_{f}T$ can be determined from ${}^{O}_{i}T$ from the following equation.

$${}^{O}_{f}T = {}^{O}_{i}T * {}^{i}_{f}T \quad \text{.......} (42)$$

where ${}^{i}_{f}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D \\ 0 & 0 & 0 & 1 \end{bmatrix}$. Here 'D' is the distance measured by the laser. Let ${}^{O}_{f}T$ and ${}^{O}_{i}T$ be given as follows.

$$ {}^{o}_{i}T = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad {}^{o}_{f}T = \begin{bmatrix} n_{x_f} & o_{x_f} & a_{x_f} & P_{x_f} \\ n_{y_f} & o_{y_f} & a_{y_f} & P_{y_f} \\ n_{z_f} & o_{z_f} & a_{z_f} & P_{z_f} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

As (P_{x_i}, P_{y_i}, P_{z_i}) are the co-ordinates of the initial point of the trajectory and (P_{x_f}, P_{y_f}, P_{z_f}) are the co-ordinates of the final point of the trajectory the vector along this trajectory is given by,

$$\overrightarrow{K\_vec} = (P_{x_f} - P_{x_i})\hat{\imath} + (P_{y_f} - P_{y_i})\,\hat{\jmath} + (P_{z_f} - P_{z_i})\,\hat{k} \quad \text{(43)}$$

The unit vector in the direction of the reference trajectory is given by, $\widehat{K\_vec}$. Let the unit vectors perpendicular to $\widehat{K\_vec}$ be given by $\widehat{L\_vec}$ and $\widehat{M\_vec}$ such that $\widehat{K\_vec}$, $\widehat{L\_vec}$ and $\widehat{M\_vec}$ form an x-y-z Cartesian triad. This means $\widehat{K\_vec}$, $\widehat{L\_vec}$ and $\widehat{M\_vec}$ are orthogonal (mutually perpendicular) and orthonormal ($\widehat{K\_vec}$ X $\widehat{L\_vec}$ = $\widehat{M\_vec}$, $\widehat{L\_vec}$ X $\widehat{M\_vec}$ = $\widehat{K\_vec}$ and $\widehat{M\_vec}$ X $\widehat{K\_vec}$ = $\widehat{L\_vec}$). Moreover, these are unit vectors.

Next we take the projections of $\vec{A}$ on $\widehat{K\_vec}$, $\widehat{L\_vec}$ and $\widehat{M\_vec}$. Let these be $\overrightarrow{A\_proj\_K}$, $\overrightarrow{A\_proj\_L}$ and $\overrightarrow{A\_proj\_M}$ respectively. These become the components of $\vec{A}$ along $\widehat{K\_vec}$ and along vectors perpendicular to $\widehat{K\_vec}$. We must remember that $\overrightarrow{A\_proj\_K}$, $\overrightarrow{A\_proj\_L}$ and $\overrightarrow{A\_proj\_M}$ are not unit vectors.

$$\overrightarrow{A\_proj\_K} = (|\,\vec{A}\,.\,\widehat{K\_vec}\,|)\ \widehat{K\_vec} \quad \text{(44)}$$

$$\overrightarrow{A\_proj\_L} = (|\,\vec{A}\,.\widehat{L\_vec}\,|)\ \widehat{L\_vec} \quad \text{(45)}$$

$$\overrightarrow{A\_proj\_M} = (|\,\vec{A}\,.\ \widehat{M\_vec}\,|)\ \ \widehat{M\_vec} \quad \text{(46)}$$

Where | A. B | represents the dot product of the two vectors A and B. Next we scale up $\overrightarrow{A\_proj\_K}$ and scale down $\overrightarrow{A\_proj\_L}$ and $\overrightarrow{A\_proj\_M}$. Let these new vectors be $\overrightarrow{A\_proj\_K\_new}$, $\overrightarrow{A\_proj\_L\_new}$ and $\overrightarrow{A\_proj\_M\_new}$ respectively. Also let us scale up to 300% and scale down to 20%.

If,

$\overrightarrow{A\_proj\_K} = x_K\,\hat{\imath} + y_K\,\hat{\jmath} + z_K\,\hat{k}$

$\overrightarrow{A\_proj\_L} = x_L\,\hat{\imath} + y_L\,\hat{\jmath} + z_L\,\hat{k}$

$\overrightarrow{A\_proj\_M} = x_M\,\hat{\imath} + y_M\,\hat{\jmath} + z_M\,\hat{k}$

Then,

$\overrightarrow{A\_proj\_K\_new} = 3 \text{ X } (x_K\,\hat{\imath} + y_K\,\hat{\jmath} + z_K\,\hat{k})$

$\overrightarrow{A\_proj\_L\_new} = 0.2 \text{ X } (x_L\,\hat{\imath} + y_L\,\hat{\jmath} + z_L\,\hat{k})$

$\overrightarrow{A\_proj\_M\_new} = 0.2 \text{ X } (x_M\,\hat{\imath} + y_M\,\hat{\jmath} + z_M\,\hat{k})$

Now after scaling up the component of $\vec{A}$ along $\widehat{K\_vec}$ and scaling down the component of $\vec{A}$ along directions perpendicular to $\widehat{K\_vec}$ we take the vector sum of the three components to get the $\overrightarrow{A\_scaled}$.

$$\overrightarrow{A\_scaled} = \overrightarrow{A\_proj\_K\_new} + \overrightarrow{A\_proj\_L\_new} + \overrightarrow{A\_proj\_M\_new} \quad \text{...} (47)$$

This is the vector that is then added to the current PUMA position to realize scaled teleoperation.

6.2.3. Force Feedback along the Trajectory

The Omni master device is a haptic device and provides force effects to the user. The user feels these forces when there is interaction with either a virtual environment or forces experienced by a remote manipulator can be transferred to the user via the haptic device. The latter is also termed as bilateral teleoperation. In this work we provide assistance to the user in following a trajectory while the user teleoperates in the direction of the trajectory. The assistance is in the form of an attractive force that pulls the user to the trajectory when the user is near the trajectory and makes the Omni tip stick onto this trajectory i.e. the user has to apply some force that is more than the force with which the user maneuvers within the environment to pull the Omni pen out of the trajectory. Thus there is no force feedback from the slave but a trajectory generated in the slave environment is sent to the Omni machine where it is mapped to the Omni base frame. The Omni machine generates force feedback on this trajectory and since it is mapped to the PUMA environment the PUMA follows the desired trajectory as the user teleoperates. This is a sort of reverse of transfer of kinematic parameters and application of dynamics in the sense that the trajectory is transferred from PUMA to Omni and forces are generated and applied at the Omni. Usually the trajectory or motion is transferred from the master to the slave and the forces are generated at the slave. This is also a virtual constraint type of assistance as the Omni is constrained to move over a virtual path by forces acting on the Omni tip, the trajectory actually existing in the PUMA workspace. Here we use information from the laser along with PUMA forward kinematics to determine the virtual constraint and then use the Omni force feedback capabilities to constrain the user over the constraint.

When the user wants to teleoperate the PUMA to move it to a target using force based virtual constraint, the user points to the target with the laser and presses the ‘F’ key on the Omni PC keyboard. When the user does this a line appears in the Omni virtual environment and a line is also rendered haptically. As the user follows the line with the Omni tip by making the 3D cursor follow the line in the virtual environment, the user can feel the line haptically as if there was a line in the Omni workspace. The line draws the Omni tip onto itself by attractive forces that act on any object (like Omni tip) which is around the line in a cylinder of influence. The user cannot deviate easily from the line as once on the line the user has to force the Omni tip out of the line. The direction of the line is such that if it is followed in teleoperation the PUMA moves in the direction of the laser pointer towards the target or away from it. The user gets assistance as the user is not able to deviate easily from the line due to the forces. Once the user reaches the target the user can press the ‘F’ key again. When the user does this the line vanishes from the graphics and the haptics. The user is now in teleoperation control mode.

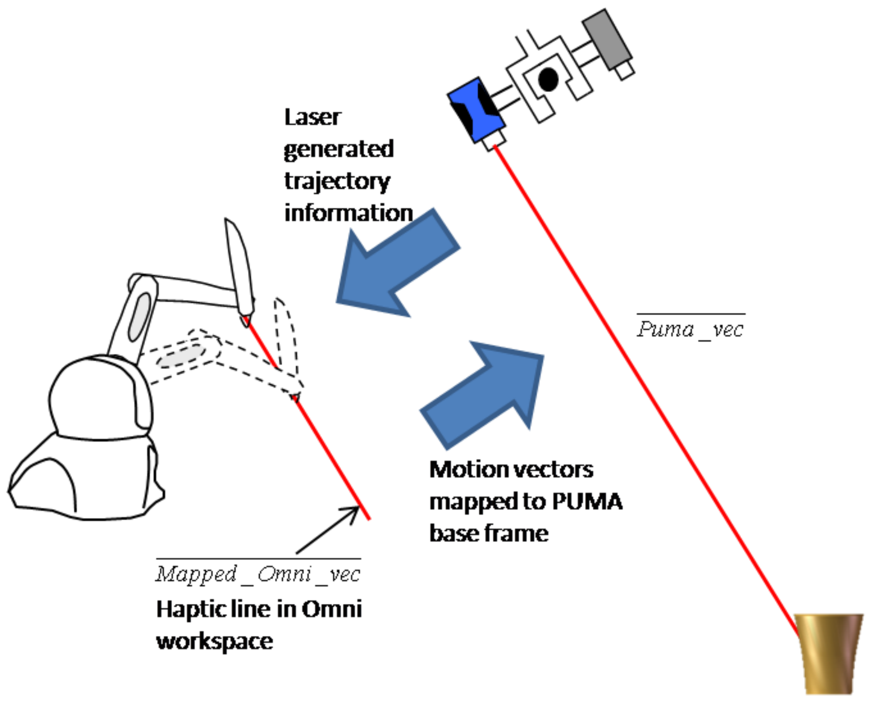

Laser generated trajectory information
$\overline{Puma\_vec}$
Motion vectors mapped to PUMA base frame
$\overline{Mapped\_Omni\_vec}$
Haptic line in Omni workspace

Figure 42: Force based virtual fixture concept

When the user presses the 'F' key on the Omni PC keyboard this information is sent to the QNX machine where the Cartesian co-ordinates of the source and destination points that form the desired trajectory are determined by the laser data and PUMA forward kinematics. We have already seen in the previous section, section 6.2.2 that ${}^{o}_{f}T$, the transformation matrix of the final point of the trajectory can be determined from ${}^{i}_{f}T$, the transformation matrix of the final point with respect to the initial point and ${}^{o}_{i}T$, the transformation matrix of the final point with respect to the initial point. ${}^{i}_{f}T$ is determined solely from laser data and ${}^{o}_{i}T$ is determined from forward kinematics. Let, ${}^{o}_{i}T = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and ${}^{o}_{f}T = \begin{bmatrix} n_{x_f} & o_{x_f} & a_{x_f} & P_{x_f} \\ n_{y_f} & o_{y_f} & a_{y_f} & P_{y_f} \\ n_{z_f} & o_{z_f} & a_{z_f} & P_{z_f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Then the co-ordinates of the source $(P_{x_i}, P_{y_i}, P_{z_i})$ and those of destination $(P_{x_f}, P_{y_f}, P_{z_f})$ are passed on to the Omni machine. At the Omni machine, a vector in the direction of the desired trajectory determined by the laser is generated using the co-ordinates. This is similar to the vector $\widehat{K\_vec}$ generated in scaled teleoperation. Let us call this vector $\overrightarrow{Puma\_vec}$.

$$\overrightarrow{Puma\_vec} = (P_{x_f} - P_{x_i})\hat{\imath} + (P_{y_f} - P_{y_i})\,\hat{\jmath} + (P_{z_f} - P_{z_i})\,\hat{k} \quad (48)$$

Let the unit vector in the direction of $\overrightarrow{Puma\_vec}$ be $\widehat{Puma\_vec}$. $\widehat{Puma\_vec}$ is in terms of the PUMA base frame. In order to render the line corresponding to this trajectory in Omni workspace, we need to map the line from the PUMA workspace to the Omni workspace. The mapping used here is the reverse of that used to transfer Omni incremental motion to PUMA workspace. Referring to equation 21,

$$\left(x_o, y_o, z_o\right) = (X_P, Y_P, Z_P) * M^{-1} \quad (49)$$

where $M = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$. We observe that $M^{-1} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.

Let the mapped unit vector be called $\widehat{Mapped\_Omni\_vec}$. When the line in the Omni workspace in the direction of $\widehat{Mapped\_Omni\_vec}$ is followed the PUMA traverses in the direction of the desired trajectory. However, to compel the user to follow the line it has to be rendered in the Omni workspace as a haptic line so that the user is able to follow it by getting a sense of force feedback along the line. So the line is rendered graphically in the OpenGL virtual environment and haptically in the Omni workspace. To render the line graphically a line starting from the origin of the graphical workspace in the direction of $\widehat{Mapped\_Omni\_vec}$ and that covers a considerable size of the graphical workspace is drawn using OpenGL APIs. $\widehat{Mapped\_Omni\_vec}$ is multiplied by a scalar, 1.4, for this purpose. As the graphical workspace of the Omni is mapped 1:1 to its haptic workspace a similar line can be rendered haptically. However additionally OpenHaptics APIs, specifically APIs from the HLAPI (refer section 3.5.3.2) library have to be used to render the line haptically. These APIs include commands to generate the line touch model and assign mechanical properties of stiffness, static friction and dynamic friction to it. It is this touch model and the mechanical properties that generate the sense of touch. A touch model of ‘constraint’ is used which constraints the Omni tip onto the line. The stiffness attributed to the line is 0.2 N/m. Moreover to enable snapping the Omni tip to the line when the Omni tip is around the line, the virtual force field generated around the line is modeled around the line. The forces that the user feels around the line are proportional to the distance of the Omni tip and the nearest point on the line and are modelled as spring forces. The graphic and haptic rendering of the line takes place at the graphic update rate of around 30 Hz. However, since the force rendering update rate has to be higher (1000 Hz) it is managed by the OpenHaptics software core and this layer is abstract from the programmer. The rendering of forces based on the assigned mechanical properties is also abstract from the user. Thus the constraining of the Omni tip onto the line and snapping it onto the line assists the user in traversing the laser generated trajectory in a force based virtual fixture formulation while the PUMA follows the desired trajectory in its workspace.

6.3. Laser based Task Plan Creation Capability

Here we demonstrate how range information from a laser range finder along with machine intelligence and high level human input can be used to generate task plans and execute them autonomously in unstructured environments. The task plan is in the form of a path that is planned and executed by the machine but specified by the human by teleoperating the arm recording information about the environment using the laser. The path is generated without using sophisticated sensory suite, extensive computer processing or slow Teach and Play methods. Using laser and high level human control through teleoperation, a simple, quick and easy way of generating and executing path plans in unstructured environments will be demonstrated. The user simply teleoperates the robotic arm, points to different points in the environment and presses certain keys on the keyboard and the system generates and executes the path autonomously.

The path can be generated from the current location of the robot end effector to anywhere in the workspace of the robot designated by the laser pointer and can have obstacles too. This capability of the path generation enables its application in unstructured environments. This also demonstrates the capability of the system to quickly generate basic 3D geometry information using laser data and human decision making. The generation and execution of the path in a Traded Control formulation where the path is generated by high level human input in teleoperation and is executed autonomously by the robot provides assistance to the user. It not only enables faster and more accurate execution of the path but also relieves the user of much mental load in teleoperating to reach the destination.

For generating a path the user, while executing a current ongoing task, teleoperates the PUMA and points the laser to the destination point which is supposed to be the final point of the path. Once the laser is pointing to the destination point the user ceases to teleoperate the arm thus locking the destination point with the laser pointer and presses the ‘D’ on the keyboard of the Omni PC. ‘D’ stands for destination point (figure 43). Then the user continues executing the current task. Once the user has positioned the PUMA at a random point due to the current task requirements and would like the arm to move over to the destination point (figure 44), the user presses the key ‘G’ and ‘A’, in succession. ‘G’ stands for go and ‘A’ for autonomous. Once the user gives the command the system generates the path online, in a matter of milliseconds and begins executing the path. The orientation of the end effector throughout the path traversal remains the same. The path that the system executes is divided into three legs. In the first leg the arm follows a vertical trajectory upwards from its current location to a certain height that it is clear from any obstacles. The height of the obstacles is presently hard coded but can be determined by pointing the laser to the tallest obstacle and recording the laser data. Once the arm reaches the end of the first leg, it stops and the user presses ‘G’ and ‘A’ again (figure 45). Each mode has to be activated and deactivated and therefore the keys have to be pressed again. This is similar to an on-off switch. To initiate the second leg the user presses ‘G’ and ‘A’ again. In the second leg the arm traverses a trajectory in the horizontal plane from the current point to a point vertically over the destination point (figure 46). When the arm stops the user hits ‘G’ and ‘A’ again. To initiate the third leg the user hits ‘G’ and ‘A’ again. In the third leg the arm follows a trajectory vertically down to a point 20 cm above the destination point (figure 47). The user again presses the keys ‘G’ and ‘A’ to switch off the path generation and execution mode and is back to the teleoperation control mode. The arm stops 20 cm vertically over the specified destination point as a safety measure so that the arm does not crash into the destination surface.

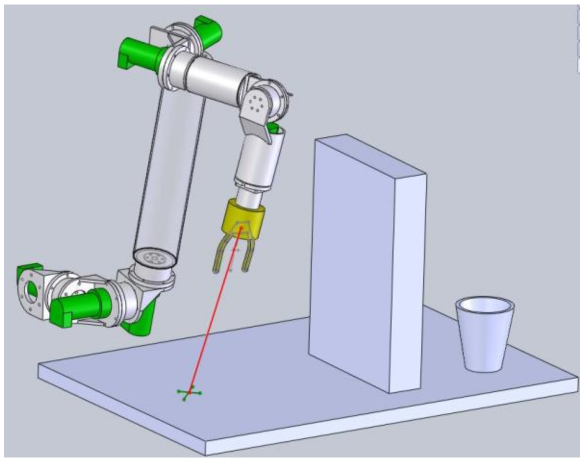


Figure 43: Recording the destination point with the laser and 'D' key

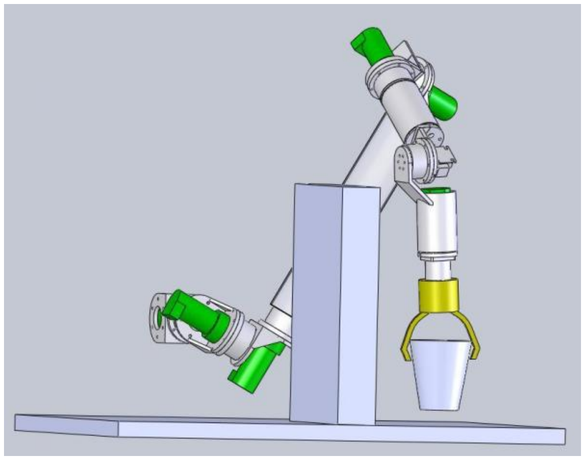


Figure 44: Ready to generate path plan and execute it

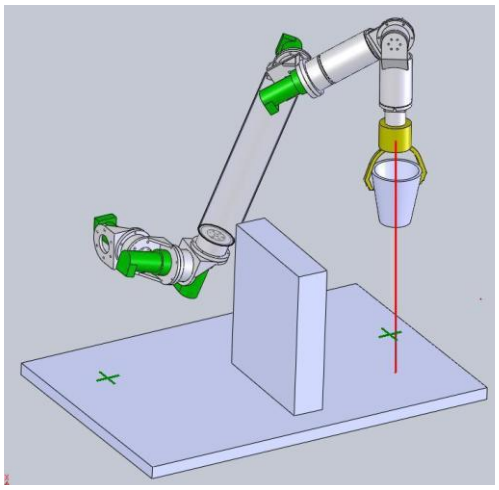


Figure 45: End of the first leg of the path

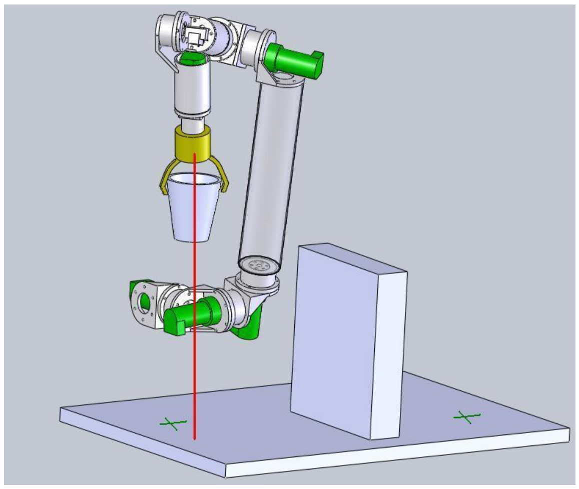


Figure 46: End of the second leg of the path

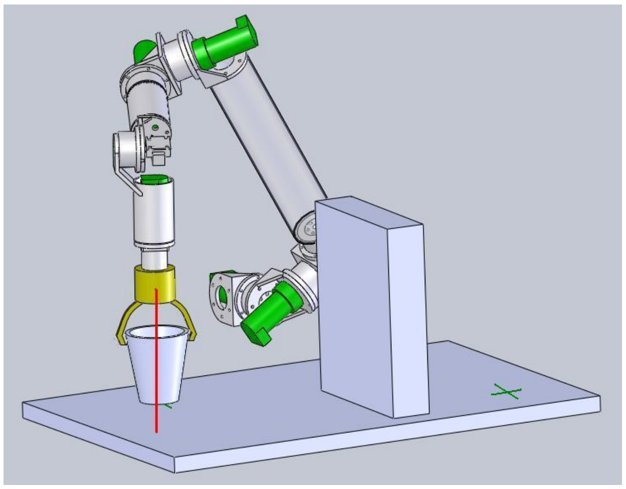


Figure 47: End of the path

When the user presses the ‘D’ key, the Cartesian co-ordinates of the destination points are recorded from the laser distance information, the transformation matrix of laser source with respect to the end effector which is constant and that of the end effector with respect to the robot base which is determined from forward kinematics. Using equation 40 and figure 36 we determine the transformation matrix of the point pointed to by the laser with respect to the PUMA base frame and the translation components of the matrix give the Cartesian co-ordinates. Let the transformation matrix be ${}^{O}_{d}T = \begin{bmatrix} n_{x_d} & o_{x_d} & a_{x_d} & P_{x_d} \\ n_{y_d} & o_{y_d} & a_{y_d} & P_{y_d} \\ n_{z_d} & o_{z_d} & a_{z_d} & P_{z_d} \\ 0 & 0 & 0 & 1 \end{bmatrix}$, then co-ordinates of the destination point are $(P_{x_d}, P_{y_d}, P_{z_d})$ with respect to the PUMA base frame.

To generate and execute a trajectory autonomously, the two inputs necessary are the initial and the final transformation matrix with respect to the PUMA base frame. For the first leg of the trajectory, when the user presses the keys ‘G’ and ‘A’ the following is executed in the code. The initial transformation matrix is determined by performing forward kinematics on the current PUMA location. Let this be ${}^{O}_{i}T = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Then the final transformation matrix would be,

$$ {}^{0}_{f}T = \begin{bmatrix} n_{x_f} & o_{x_f} & a_{x_f} & P_{x_f} \\ n_{y_f} & o_{y_f} & a_{y_f} & P_{y_f} \\ n_{z_f} & o_{z_f} & a_{z_f} & P_{z_f} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i} + 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Thus the final transformation matrix is the same as the initial transformation matrix with only the z co-ordinate changed so that the arm end effector rises to a height of 0.3 m above the current location. After this the number of via points are computed based on the distance of the initial and final points, linear interpolation and Equivalent Angle-Axis is carried out to determine the via points transformation matrices, Resolved Rate algorithm is carried out to generate joint angles at 200 Hz refresh rate and these are then sent to the torque generator thread to send appropriate voltages to the PUMA motors to move them. The PUMA comes to a stop at the end of the first leg of the trajectory. The keys ‘G’ and ‘A’ are pressed again to reset the trajectory generator.

For the second leg of the trajectory, when the keys ‘G’ and ‘A’ are pressed again, a new trajectory is computed and executed. The initial transformation matrix for the second leg of the trajectory is computed from the forward kinematics on the current location of the arm. Let his be

$$ {}^{0}_{i}T = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Then the final transformation matrix would be

$$ {}^{0}_{f}T = \begin{bmatrix} n_{x_f} & o_{x_f} & a_{x_f} & P_{x_f} \\ n_{y_f} & o_{y_f} & a_{y_f} & P_{y_f} \\ n_{z_f} & o_{z_f} & a_{z_f} & P_{z_f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{0}_{f}T = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{xd} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{yd} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix}. $$

Thus the components of the rotation part of the final transformation matrix of the second leg of the path remain the same as that of the initial transformation matrix and so does the z component of the translation. The z component remains unchanged as the arm has to traverse in the horizontal plane. The x and y components of the translation are the same as the x and y co-ordinates of the destination point (P_{x_d}, P_{y_d}, P_{z_d}) which was recorded earlier by the user. For executing the trajectory the same procedure for the autonomous trajectory execution is followed.

For generating trajectory for the third leg and executing it, again initial and final transformation matrices with respect to the PUMA base frame are determined. The initial transformation matrix is determined from the forward kinematics of the PUMA.

Let this be ${}^{O}_{i}T = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

The final transformation matrix will be

$$ {}^{O}_{f}T = \begin{bmatrix} n_{x_f} & o_{x_f} & a_{x_f} & P_{x_f} \\ n_{y_f} & o_{y_f} & a_{y_f} & P_{y_f} \\ n_{z_f} & o_{z_f} & a_{z_f} & P_{z_f} \\ 0 & 0 & 0 & 1 \end{bmatrix} {}^{O}_{f}T = \begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_d} + 0.35 \\ 0 & 0 & 0 & 1 \end{bmatrix}. $$

As the PUMA moves vertically down the x and y co-ordinates of the translation remain the same as that of the initial transformation matrix and the end effector comes to a halt at a point 35 cm above the destination point. The margin of 35 cm accounts for the gripper length of 23 cm and some margin for any object hanging out of the gripper and a safety margin. This generated trajectory is executed by the machine. After the arm completes the third leg, the keys ‘G’ and ‘A’ are pressed to reset the trajectory generator. Now the control mode is again teleoperation control mode and the user can resume with the task at hand.

This way the machine generates path plans and executes them autonomously based on the high level information provided by the user regarding destination co-ordinates by teleoperating the robot, pointing with the laser and pressing certain keys on the keyboard. This way the user gets assistance while doing a task in a Traded Control formulation and operates on a simple, quick and easy to use interface. The user thus need not be involved with complete task execution and this relieves the user of much mental load as will be seen in the results section. Moreover, the path can be generated in an unstructured environment where the location of destination points and obstacles are not known to the system prior to the task execution. The basic environment information comprising of location of destination points and paths is created online during task execution using a simple teleoperator interface with laser and minimal computing. Moreover there is a good distribution of work load between the human and the machine, the human involved with high level decision making while the machine involved in real-time computations and precision trajectory execution. The human still remains a supervisor and intermediates in case when action is needed.

6.4. Laser based Virtual 3D Geometry Generation Capability

The capability of the laser to generate virtual 3D geometries of the robot environment along with using machine intelligence and human input is demonstrated here. These 3D geometric models assist the user in doing a task in a Traded Control formulation where robot motion can be automated based on the 3D geometry information. Moreover, the 3D geometries can be generated in an unstructured environment using a quick and easy to use teleoperator interface. Here too the human makes high level decisions and the robot executes computationally intensive and precision motion tasks. Here we have demonstrated the generation of the equation of the perpendicular to a surface as an example of 3D geometry generation using the laser information. The knowledge of the 3D surface helps the system align the arm end effector and hence the hand autonomously with the surface. This gets the arm in a proper configuration to execute the next step of the task. For example, if the task is opening a door, then the alignment of the hand with the door surface gets the arm in a convenient configuration to operate on the door knob. If the task is a pick-and-place task then this will be helpful in getting the arm in a convenient configuration to pick the cup up. This is especially useful as it is extremely difficult for the users to orient the arm while teleoperating. This way the users are assisted in executing the task and it relieves them of much cognitive load in getting the arm in a proper configuration for the next step of task execution.

Next we describe the steps the user carries out to effect the 3D surface equation generation and autonomous alignment of the hand with the surface. For this the user points the laser to three distinct points on the surface of interest (figure 48). For opening a door task the door surface is the surface of interest and for a pick-and-place task the surface of interest is the one on which the target object is placed. Each time the user points to a specific point on the surface the user presses the ‘P’ key on the Omni PC keyboard twice. The user has to point to the three points in such a way that they are in a counter clockwise fashion when viewed from the direction of the robot approach. This constraint is necessary so that for the direction of the vector perpendicular to the surface is determined correctly (and not the opposite of what it should be). Now when the user is near the surface and would like to align the hand perpendicular to the surface (figure 49) the user presses the ‘T’ and ‘A’ key in succession. The hand starts orienting to align with the surface perpendicular autonomously and stops when it is aligned (figures 50, 51 and 52). Now the user presses the ‘T’ and ‘A’ keys in succession to deactivate the mode. Now the control is back to the unassisted teleoperation control mode and the user can resume with the task.

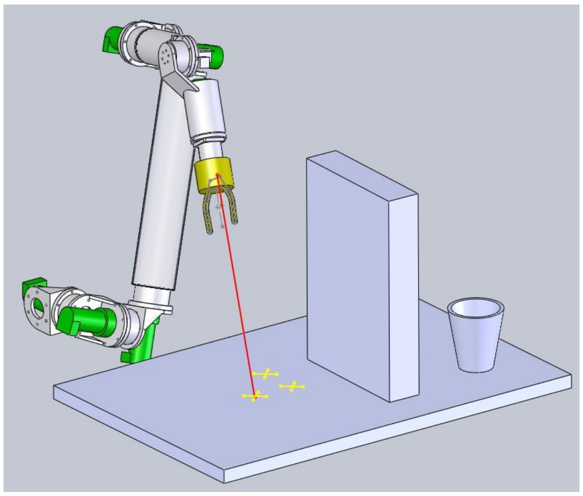


Figure 48: Pointing to 3 points on surface of interest

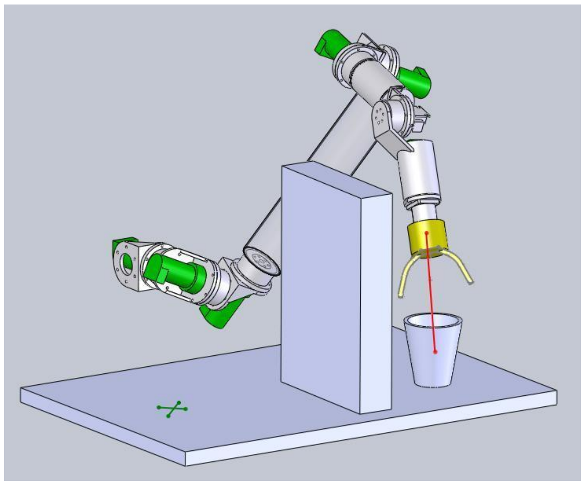


Figure 49: Arm in position for executing autonomous orientation

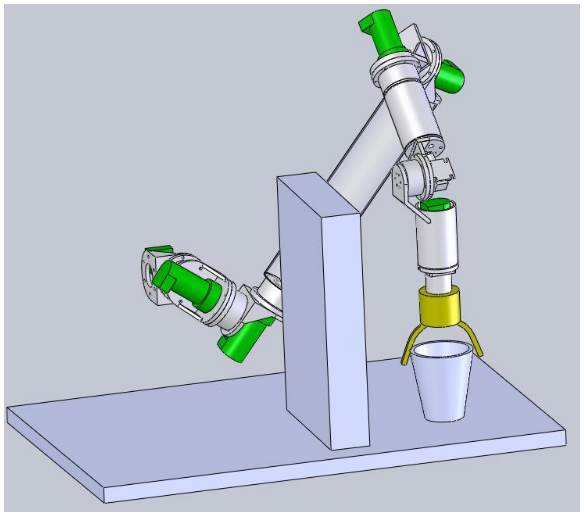


Figure 50: Arm aligned with perpendicular to the surface - perspective view

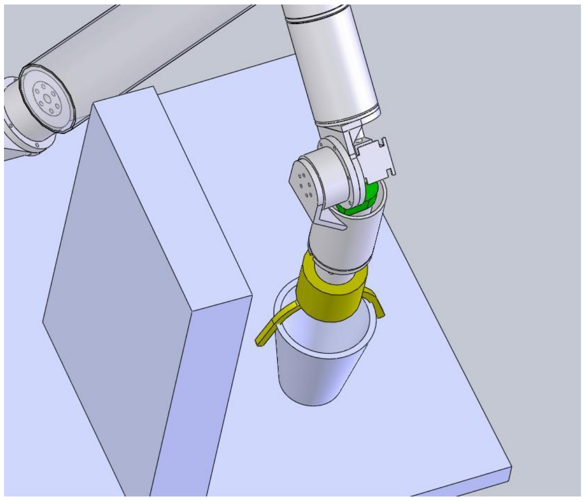


Figure 51: Arm aligned perpendicular to the surface - close up view

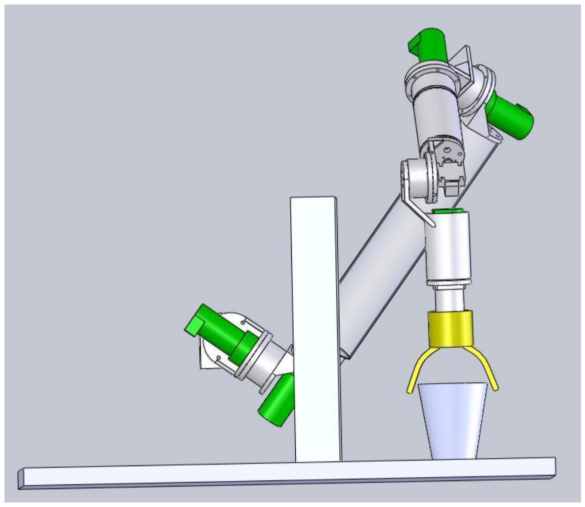


Figure 52: Arm aligned perpendicular to the surface - front view

Another example of 3D surface generation in an unknown environment for autonomous alignment using laser and human input coupled with machine intelligence is shown in the following figures.

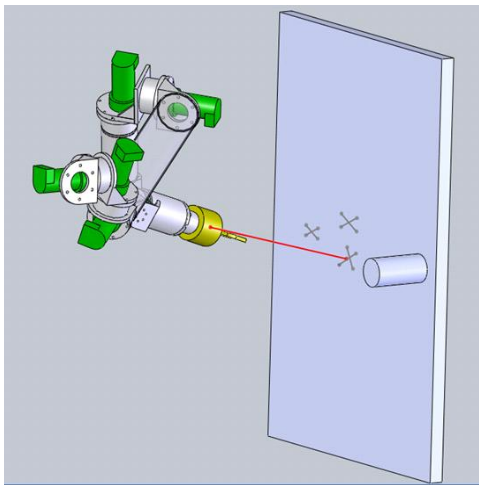


Figure 53: Pointing to 3 points on door

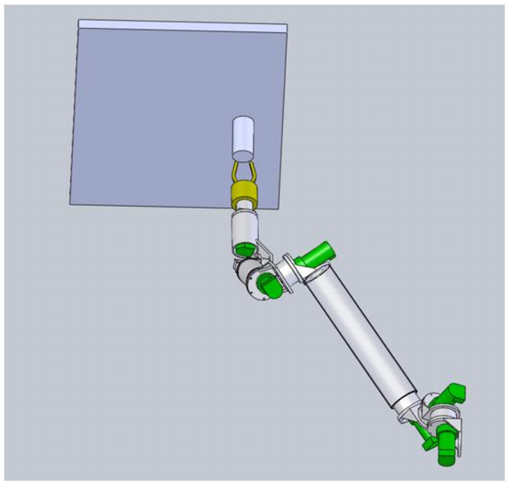


Figure 54: Autonomous door alignment

When the user presses the ‘P’ key twice for each point while teleoperating, the Cartesian co-ordinates of the points on the surface the laser is pointing too are recorded. This is similar to the recording of the destination point in the laser based path planning scheme. Again using equation 40 and figure 36 we determine the transformation matrices with respect to the PUMA base frame of the three points on the surface of interest and the translation components of the matrices give us the Cartesian co-ordinates of the three points. These co-ordinates also form the components of the vectors originating from the origin of the PUMA base frame and terminating at the points. Let the transformation matrices be

$${}^{O}_{P1}T = \begin{bmatrix} n_{x_{P1}} & o_{x_{P1}} & a_{x_{P1}} & P_{x_{P1}} \\ n_{y_{P1}} & o_{y_{P1}} & a_{y_{P1}} & P_{y_{P1}} \\ n_{z_{P1}} & o_{z_{P1}} & a_{z_{P1}} & P_{z_{P1}} \\ 0 & 0 & 0 & 1 \end{bmatrix},\ {}^{O}_{P2}T = \begin{bmatrix} n_{x_{P2}} & o_{x_{P2}} & a_{x_{P2}} & P_{x_{P2}} \\ n_{y_{P2}} & o_{y_{P2}} & a_{y_{P2}} & P_{y_{P2}} \\ n_{z_{P2}} & o_{z_{P2}} & a_{z_{P2}} & P_{z_{P2}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ and

$${}^{O}_{P3}T = \begin{bmatrix} n_{x_{P3}} & o_{x_{P3}} & a_{x_{P3}} & P_{x_{P3}} \\ n_{y_{P3}} & o_{y_{P3}} & a_{y_{P3}} & P_{y_{P3}} \\ n_{z_{P3}} & o_{z_{P3}} & a_{z_{P3}} & P_{z_{P3}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$. The co-ordinates of the points would be $(P_{x_{P1}}, P_{y_{P1}}, P_{z_{P1}})$, $(P_{x_{P2}}, P_{y_{P2}}, P_{z_{P2}})$ and $(P_{x_{P3}}, P_{y_{P3}}, P_{z_{P3}})$. The vectors to these points from the PUMA base frame origin would be

$\overrightarrow{P1} = P_{x_{P1}}\ \hat{\imath}\ + P_{y_{P1}}\ \hat{\jmath} +\ P_{z_{P1}}\ \hat{k}$ .... (50)

$\overrightarrow{P2} = P_{x_{P2}}\ \hat{\imath}\ + P_{y_{P2}}\ \hat{\jmath} +\ P_{z_{P2}}\ \hat{k}$ .... (51)

$\overrightarrow{P3} = P_{x_{P3}}\ \hat{\imath}\ + P_{y_{P3}}\ \hat{\jmath} +\ P_{z_{P3}}\ \hat{k}$ .... (52)

Form basic vector algebra we know that the cross product of two vectors is a vector perpendicular to the plane defined by the two vectors. So in this case the vector perpendicular to the surface of interest can be determined by taking the cross product of vectors connecting the three points. Let us define two vectors $\vec{A}$ and $\vec{B}$ such that

$\vec{A} = \overrightarrow{P1} - \overrightarrow{P2}$ .... (53)

$\vec{B} = \overrightarrow{P1} - \overrightarrow{P3}$ .... (54)

Now, $\vec{C}\ = \vec{A} \times \vec{B}$ is a vector perpendicular to the surface of interest. In order to align the hand to the perpendicular to the surface, the necessary condition is to align the z axis of end effector or joint 6 frame to the vector that is in opposite direction of the perpendicular to the surface i.e. to align the z-axis of joint 6 to -ve $\vec{C}$. Let the unit vector in the direction of –ve $\vec{C}$ be $\widehat{Z\_vec}$. This will be the direction of the joint 6 z-axis.

Another point to be considered is that we only need to orient the robot in order to align it to the perpendicular to the surface i.e. the robot will have the same position during this movement. Thus an autonomous trajectory will be generated that will have the same initial and final position but different rotation. The initial position and rotation are computed from the forward kinematics. The final position remains same as the initial. Thus we need only the final point rotation matrix. We know that the first, second and third columns of a rotation matrix of a joint represent unit vectors in the direction of the x, y and z axis of that joint. In other words to find out a rotation matrix we only need to determine the directions of the x, y and z axis of the Cartesian co-ordinate frame.

Thus to determine the rotation matrix of the joint 6, we need to determine the directions of the x, y and z axis of the joint six Cartesian co-ordinate frame. We already have determined the direction of the z axis of the joint 6 frame, which happens to be $\widehat{Z\_vec}$. The x and y axis of the joint 6 frame would then be in a plane parallel to the surface since the Cartesian frame of joint 6 is a right handed frame. We could assign any unit vectors in the plane parallel to the surface of interest such that they form a right handed set with each other and with $\widehat{Z\_vec}$. These vectors when assigned to the rotation matrix columns would form the joint 6 or end effector rotation matrix and this way we would be able to orient the arm perpendicular to the surface autonomously. However, assigning random x and y vectors in this way could make the arm orient more than necessary which could exceed joint limits or could waste the time in doing the task. Thus it is an undesirable motion.

In order to optimize so that the arm will have minimum rotation, we determine the projection of the current configuration joint 6 x-axis onto the surface of interest and compute the unit vector along it. This then will become the x axis of the joint 6 frame final configuration and the y axis will be computed by the right hand rule. Then these will form the final rotation matrix and the rotation can be effected autonomously. This methodology of computing the final rotation matrix would have minimum rotation of the end effector and thus the arm would be aligned perpendicular to the surface in minimum time.

Let the initial transformation matrix of the joint 6 or that in the current configuration be given by ${}^{O}_{i}T$ = $\begin{bmatrix} n_{x_i} & o_{x_i} & a_{x_i} & P_{x_i} \\ n_{y_i} & o_{y_i} & a_{y_i} & P_{y_i} \\ n_{z_i} & o_{z_i} & a_{z_i} & P_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix}$. The x axis of the joint 6 frame is then given by $\widehat{Xi\_vec}$.The projection of $\widehat{Xi\_vec}$ on the surface is given as follows. Let,

$$\overrightarrow{Temp\_vec} = \widehat{Z\_vec} \text{ X } \widehat{X\imath\_vec} \text{ ..... } (55)$$

Let the unit vector in the direction of $\overrightarrow{Temp\_vec}$ be $\widehat{Temp\_vec}$.

Let,

$$\widehat{X\_vec} = \widehat{Temp\_vec} \text{ X } \widehat{Z\_vec} \text{ ..... } (56)$$

Then, $\widehat{X\_vec}$ is the unit vector which is the projection of the x axis of the initial configuration joint 6 frame x-axis onto the surface. Also, $\widehat{Temp\_vec}$ is surprisingly the unit vector along the y axis formed by the right hand rule on x and z-axis that are obtained. Let it be called $\widehat{Y\_vec}$. Then, $\widehat{Y\_vec} = \widehat{Temp\_vec}$.

Thus these unit vectors form the column components of the final rotation matrix that the arm needs to have in order to align with the surface.

Let,

$$\widehat{X\_vec} = X\_vec_x\,\hat{\imath} + X\_vec_y\,\hat{\jmath} + X\_vec_z\,\hat{k} \text{ ..... } (57)$$

$$\widehat{Y\_vec} = Y\_vec_x\,\hat{\imath} + Y\_vec_y\,\hat{\jmath} + Y\_vec_z\,\hat{k} \text{ ..... } (58)$$

$$\widehat{Z\_vec} = Z\_vec_x\,\hat{\imath} + Z\_vec_y\,\hat{\jmath} + Z\_vec_z\,\hat{k} \quad (59)$$

Thus the final transformation matrix would be $${}^{0}_{f}T = \begin{bmatrix} X\_vec_x & Y\_vec_x & Z\_vec_x & P_{x_i} \\ X\_vec_y & Y\_vec_y & Z\_vec_y & P_{y_i} \\ X\_vec_z & Y\_vec_z & Z\_vec_z & P_{z_i} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Based on the initial and final transformation matrix the via points transformation matrices are determined and Resolved Rate inverse kinematics in them yields joint angles that are computed at 200 Hz update rate and are simultaneously supplied to the torque generator thread in order to transmit voltage information to the motors of the PUMA. This causes the hand to align with the perpendicular to the surface autonomously.

Thus we see that the machine is able to generate a virtual 3D geometry in the form of the equation of the perpendicular to a surface from basic range information from a laser sensor and high level human input coupled with machine intelligence in the form of computations. Moreover, equation of any flat surface in the environment can be generated by pointing the laser to any three points on the surface and pressing the 'P' key on the keyboard. This capability is useful to execute tasks in unstructured environments. The 3D geometry is generated online and by using a simple and easy to use teleoperator interface and without the need for extensive computing resources. Again the human is only a supervisor of the task and gives high level commands based on his decisions. The machine does the computationally intensive and precision tasks. This assists the user in a Traded Control formulation as the user is relieved from doing the task that requires high mental involvement. Especially there is a lot of cognitive load on the user when the user tries to orient the arm to a desired configuration by teleoperation so that the arm is in a convenient configuration for the next step of the task. Moving the arm from one point to another is also cognitively challenging due to the decoupling of user from the remote robot and due to lack of telepresence. However, orienting is more challenging. Thus, the assistance saves the user a lot of time and energy compared to executing the task in teleoperation and makes the task execution more accurate. This will be seen in detail in the results section.

6.5. Application of Laser based Features in Executing a Task

We have discussed several capabilities of laser combined with human high level decision making and machine intelligence that assists the user to execute tasks in unstructured environments. Here we present pick-and-place task, a very common remote manipulation task to demonstrate how the different capabilities can be combined together to assist the user to execute the task. This will show how the laser based capabilities along with human and machine inputs can not only help during parts of a task but to execute a complete task. Later in the results chapter we will show that these capabilities actually reduce the time in executing a task and improve the accuracy. Moreover, we will see that the cognitive load on the user is reduced by a great amount when using the laser based capabilities to do the task.

In a pick-and-place task which is executed by teleoperating the PUMA with the Omni master, the remote environment has a target object, commonly called as target, a source point where the target is located and from where it has to be picked up, a destination point where the target should be dropped and a ready position where the robot end effector is before the task starts. There can also be obstacles in the environment which have to be avoided. One way of executing the task is through teleoperation where the user operates the remote arm, maneuvers it through the remote environment avoiding obstacles and traversing it through trajectories. Here the user gets no assistance from the robot or the machine. The user uses its cognitive, sensorimotor, planning and decision making skills to do the task. Doing the task this way is difficult because the user is decoupled from the remote environment; the dexterity of the remote arm is not as good as that of the user and the transfer of motion from the master to slave may not be intuitive at times. Even if there is assistance from the machine in the form of information feedback there may be lack of kinesthetic feedback and the feedback from the remote environment may not be compatible with the user. These deficiencies can be removed by using the laser based capabilities of autonomous trajectory generation, virtual constraint generation, 3D geometry generation and path planning and execution.

When the arm is at the ready position for the pick-and-place task then the first step is to move the arm towards the target and to pick up the target. In an unassisted mode this is done by teleoperating the arm towards the target by following the shortest possible trajectory. Without proprioceptive feedback and telepresence in proper form, due to the decoupling of the user form the remote environment, poor remote arm dexterity and poor transfer of motion, it is extremely difficult to maneuver the remote arm to reach the target. At times there is deviation from the desired trajectory and at times the remote arm attains configurations from where it is difficult to bring it back to normal teleoperation control due to the Cartesian mapping. A combination of joint and Cartesian mapping could be one solution in this case. Again when the arm reaches in the vicinity of the target the user has to orient the arm so that the target can be grasped from the top or side or any other proper configuration. Orienting the arm suitable so that it is compatible for the next step like grasping is one of cognitively difficult tasks due to Cartesian mapping. Again activating joint mode could be a solution in this case but small joint movements of the joints 1, 2 and 3 of the Omni can lead to large end effector movements and this could be undesirable especially when the arm is near to target and the surface over which the target is. After grasping the target, the user has to teleoperate the PUMA while avoiding obstacles and reach the destination where the target can be dropped. This can be cognitively difficult for users new to the system, is time consuming and inaccurate as far as following trajectories is concerned e.g. shortest paths possible.

However, the laser based features assist the user to execute tasks without much cognitive load on the user, in a shorter time and with good accuracy. Before starting the task the user points the laser to the destination point and presses the 'D' key on the PC keyboard to record the destination point. This will be later used by the machine to generate path plan and execute the path (figure 55). The destination point can be anywhere in the environment. Then the user points the laser to three random points on the surface where the target is placed and presses the 'P' key on the keyboard twice for each point. The information generated as a result will be used by the machine for autonomous alignment of the PUMA end effector with the surface (figure 56). Here the human is making high level decisions to select destination points and recognize surfaces which need not be automated because of human-in-the-loop. Then the user points the laser to the target by teleoperating the PUMA and presses the ‘A’ key on the PC keyboard (figure 57). The machine generates a linear trajectory online; the PUMA follows the trajectory and stops at a threshold distance close to the target. The laser point remains on the target at the same point throughout the trajectory (figure 58). The point-n-click method of autonomous trajectory generation and execution is quick, the interface simple and the target could be located anywhere in the environment. The human only makes high level decisions and the robot executes the computationally intensive and precision motion tasks. Now the user presses the ‘T’ and ‘A’ key on the keyboard and the machine orients the end effector and hence the hand mounted on it autonomously so that the hand is aligned with the perpendicular to the surface for easy grasping (figure 59). The machine actually generates a 3D geometry of the surface online on the press of the keys based on the information it recorded earlier when the ‘P’ key was pressed and the laser pointed to on the surface. The 3D surface here is in the form of a vector in the 3D space perpendicular to the surface. Again the surface can be any surface in the environment. Now with minimum teleoperation, the target is grasped (figure 60) and the ‘G’ and ‘A’ keyboard keys are pressed. On doing this the machine generates a path from the current location to the destination point recorded earlier and executes the path. The path is in the form of three segments or linear trajectories, one from the current location to a point vertically over it (figure 61), the other from this location to a point vertically above the destination along the horizontal plane (figure 62) and the third from this location to a point 20 cm over the destination point (figure 63). Again there is minimum human involvement in traversing the path as the human only commands and the robot performs the computations and executes the task. As we see the method is quick and the interface is very easy to use. The user is relieved of much cognitive load in traversing the trajectory up to the destination. Next the user, with the help of minimum teleoperation is able to place the target over the destination and complete the task (figure 66).

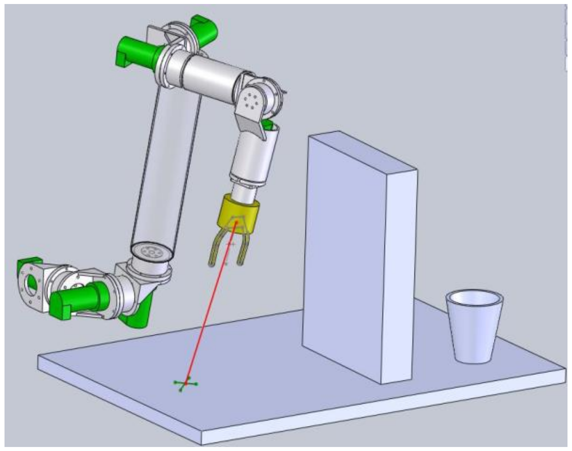


Figure 55: Point to destination for path planning

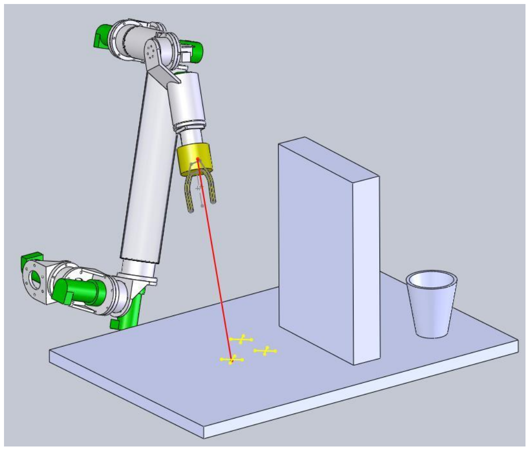


Figure 56: Point to 3 points for surface geometry generation

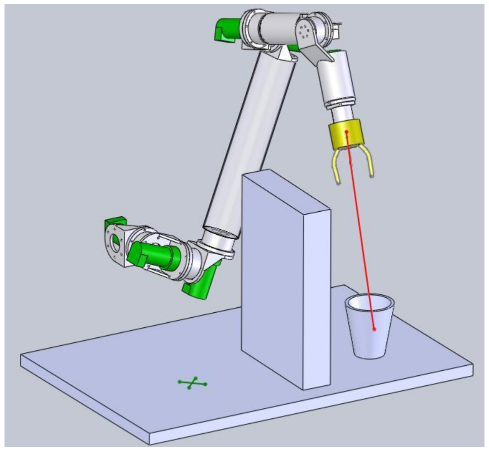


Figure 57: Point to target for autonomous trajectory

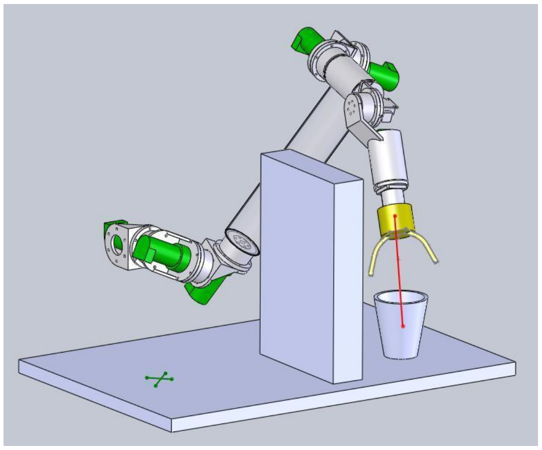


Figure 58: End of autonomous trajectory

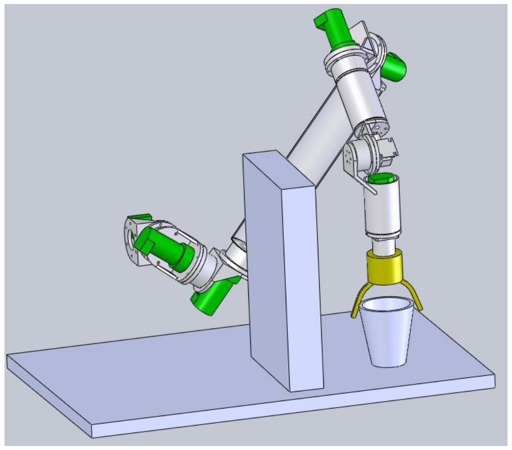


Figure 59: Autonomous hand alignment

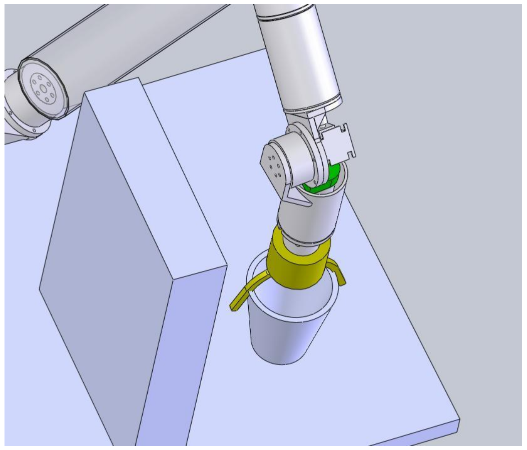


Figure 60: Autonomous alignment – close up

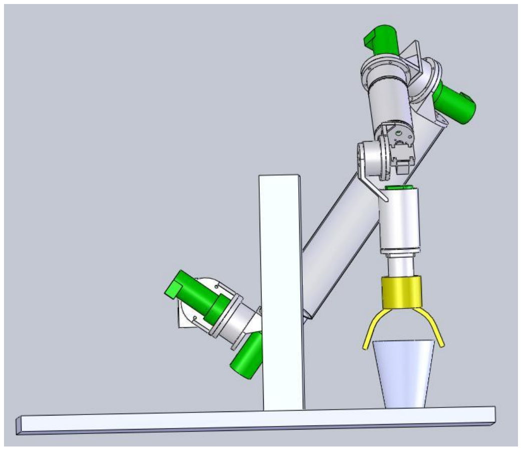


Figure 61: Autonomous hand alignment - front view

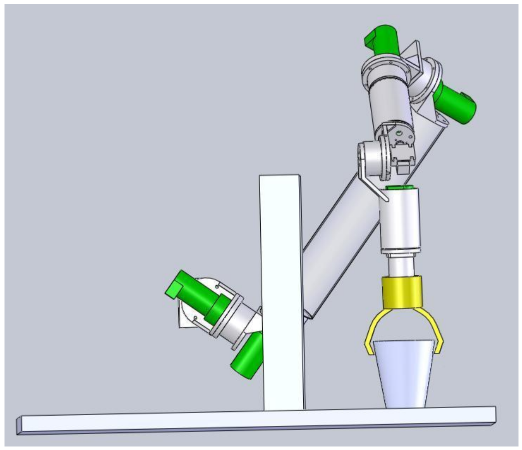


Figure 62: Target grasped

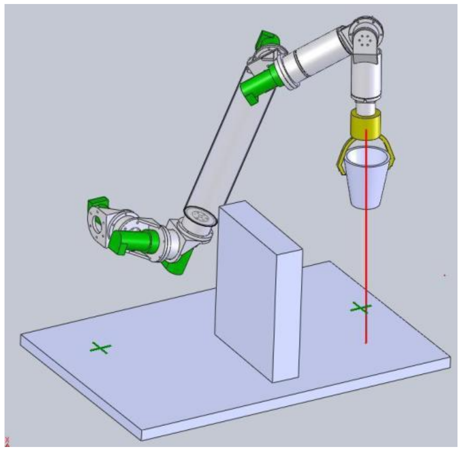


Figure 63: Autonomous path - first leg

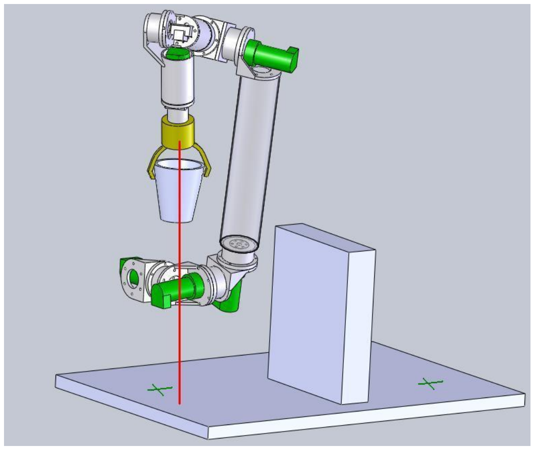


Figure 64: Autonomous path - second leg

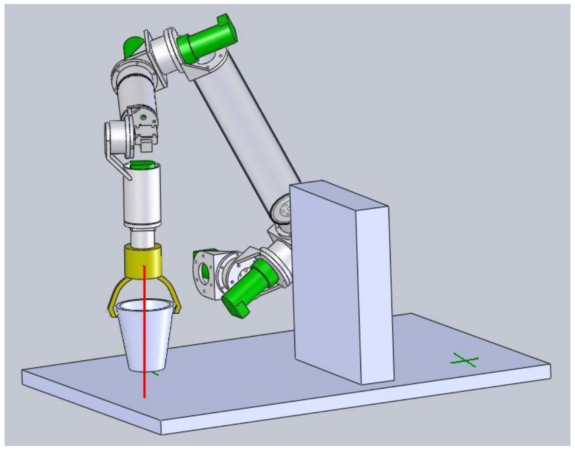


Figure 65: Autonomous path: third leg

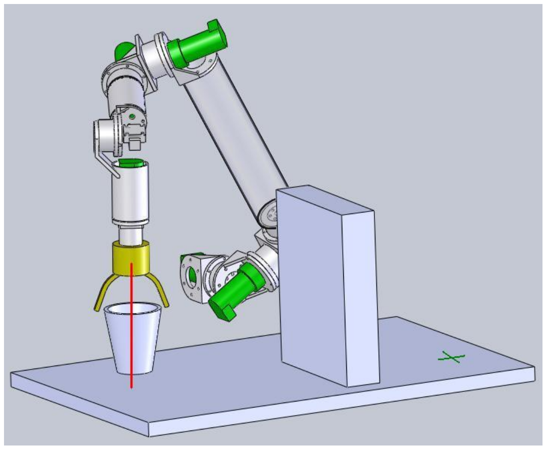


Figure 66: Target dropped at destination

Chapter 7: Results and Discussions

In this chapter we present the results relating to the work in the thesis project. First we present the results relating to the accuracy and reproducibility of the remote robot, the PUMA. The experiments carried out to achieve the results have been explained too. The accuracy of the PUMA is determined by comparing its actual motion with that in simulation. The reproducibility has been measured by commanding the PUMA to go to a certain location and measuring its transformation matrix each time it reaches that location. The validation of the motion control algorithms of the PUMA is also presented in this section.

Next we present the results from the testing carried out to validate the hypothesis that the laser assisted method of task execution is better in terms of time, accuracy and cognitive load on the user. The outcome of the testing confirms the hypothesis. We present the experiments carried out by the users to determine the metrics for validating the hypothesis.

In the third section of this chapter we present the results related to the improvement in the robot accuracy and repeatability as a result of the new laser characteristics. In the last section we demonstrate the results from the singularity avoidance technique implementation.

7.1. Accuracy and Reproducibility Results on PUMA

In this section we present the results to evaluate the accuracy of the PUMA and its reproducibility. We also validate the motion control algorithms that run on PUMA. Matlab robotics toolbox, which has a kinematic and graphic PUMA model, was used for obtaining results in simulation.

First, the motion control algorithms running on the PUMA were validated. These algorithms included forward kinematics, inverse kinematics, trajectory generation based on initial and final transformation matrices and trajectory generation based on laser data. The tests for validation of these algorithms were very basic tests and so an overview of these tests is given here. For validating the forward kinematics the PUMA was taken to a few positions like the home position, the zero position and so on. The transformation matrix at these PUMA configurations is already known from theory. Transformation matrices for a few other configurations were determined by using the PUMA Matlab model. The PUMA was moved to a certain configuration and its joint angles were recorded. The same joint angles were fed to the PUMA Matlab model and a forward kinematics on that was carries out. A forward kinematics on the actual PUMA was also carried out and the transformation matrices were compared. The inverse kinematics was done using Resolved Rate algorithm. It is known from the theory (chapter 6) that the PUMA follows a straight line trajectory in the direction of the laser beam in case of the laser generated trajectory i.e. the laser pointer remains on the same spot over the target as the PUMA follows the trajectory. When this was observed by running the trajectory on the PUMA, the inverse kinematics and trajectory generation algorithms were confirmed as validated. The congruence of transformation matrices and joint angles through the trajectory from tests on simulation and actual PUMA also validated the algorithms. However, this alone was not sufficient as the PUMA may still be generating and following an altogether different trajectory. The laser test was required in this case.

The accuracy of the PUMA has been determined by comparing its trajectories in simulation with those actually executed on the PUMA. First the PUMA was commanded to go at a certain initial point in joint mode. The sequence of joint movements that were needed to command the PUMA to that particular point were recorded in case the same trajectories were needed to be replicated later on. This point was considered as the initial point of the trajectory. The joint angles and the transformation matrix at this point were recorded so that this can be replicated in simulation and for later replication on the actual arm. Next the PUMA was moved in joint mode to another random point in its workspace which was considered as the final point of the trajectory. The joint angles and the transformation matrix at this point were also recorded. The positioning of the PUMA at the initial and final points could have also been achieved by teleoperating the PUMA, however, the replication of the initial point would have been difficult in this case as the sequence of movements would not have been recorded. The trajectory between this initial point and the final point was generated in simulation and was run on the actual arm. Five such trajectories were generated.

The translation components of the via points transformation matrices obtained from simulation were used to determine the trajectory in simulation or the ideal trajectory whereas the x-y-z fixed angles [24] are used to determine the ideal orientation of the end effector through the trajectory. The joint angles computed from Resolved Rate algorithm were also recorded as the ideal joint angles. After it was observed that the values of via point's transformation matrices and those of the joint angles were the same in simulation and on the PC system that controlled the actual arm, these values were directly taken from the PUMA PC and the simulation was not run anymore. Thus the computations on the PUMA PC became the simulated values. For determining the actual PUMA trajectory, transformation matrices were computed by carrying out forward kinematics on the encoder angles that were recorded through the trajectory that the PUMA traversed. The encoder angles were recorded as actual joint angles. The following plots were generated from the gathered data:

1. 3D plot of the ideal trajectory and the actual trajectory.
2. Trajectory error plot that showed the distance of corresponding points on the ideal and actual trajectory against time steps.
3. Plots of the ideal and actual end effector angles i.e. the x-y-z fixed angles against time steps.
4. Error plot of the ideal and actual end effector angles that showed the difference in the ideal and actual alpha, beta and gamma angles at each point on the trajectory against time steps.

5. Plot of ideal and actual joint angles for the six PUMA joints against time steps.
6. Error plot of joint angles showing the difference in actual and ideal joint angles at different points on the trajectory against time steps. One plot for each joint was generated.

The five trajectories are listed below:

1. Initial joint angles: -1.05628 -2.1085 0.05893 0.17227 -0.74605 0.01345. Final joint angles: -1.68307 -3.10591 0.79649 -0.51276 -0.58633 0.09572.
2. Initial joint angles: -1.09191 -2.35503 0.29164 -0.4043 -0.78129 0.01054. Final joint angles: -1.68307 -3.10591 0.79649 -0.51276 -0.58633 0.09572.
3. Initial joint angles: -1.72401 -2.14383 0.12948 -0.33586 -0.64322 0.00296. Final joint angles: -1.18062 -3.11768 0.56893 0.99229 -0.48414 -0.42776.
4. Initial joint angles: -0.81073 -2.21201 -0.01081 0.5445 -0.77945 0.01955. Final joint angles: -1.18062 -3.11768 0.56893 0.99229 -0.48414 -0.42776.
5. Initial joint angles: -1.51358 -2.70128 0.53898 0.27469 -0.78304 0.01727. Final joint angles: -1.18062 -3.11768 0.56893 0.99229 -0.48414 -0.42776.

The plots for the five trajectories are shown in the figures 67 though 71.

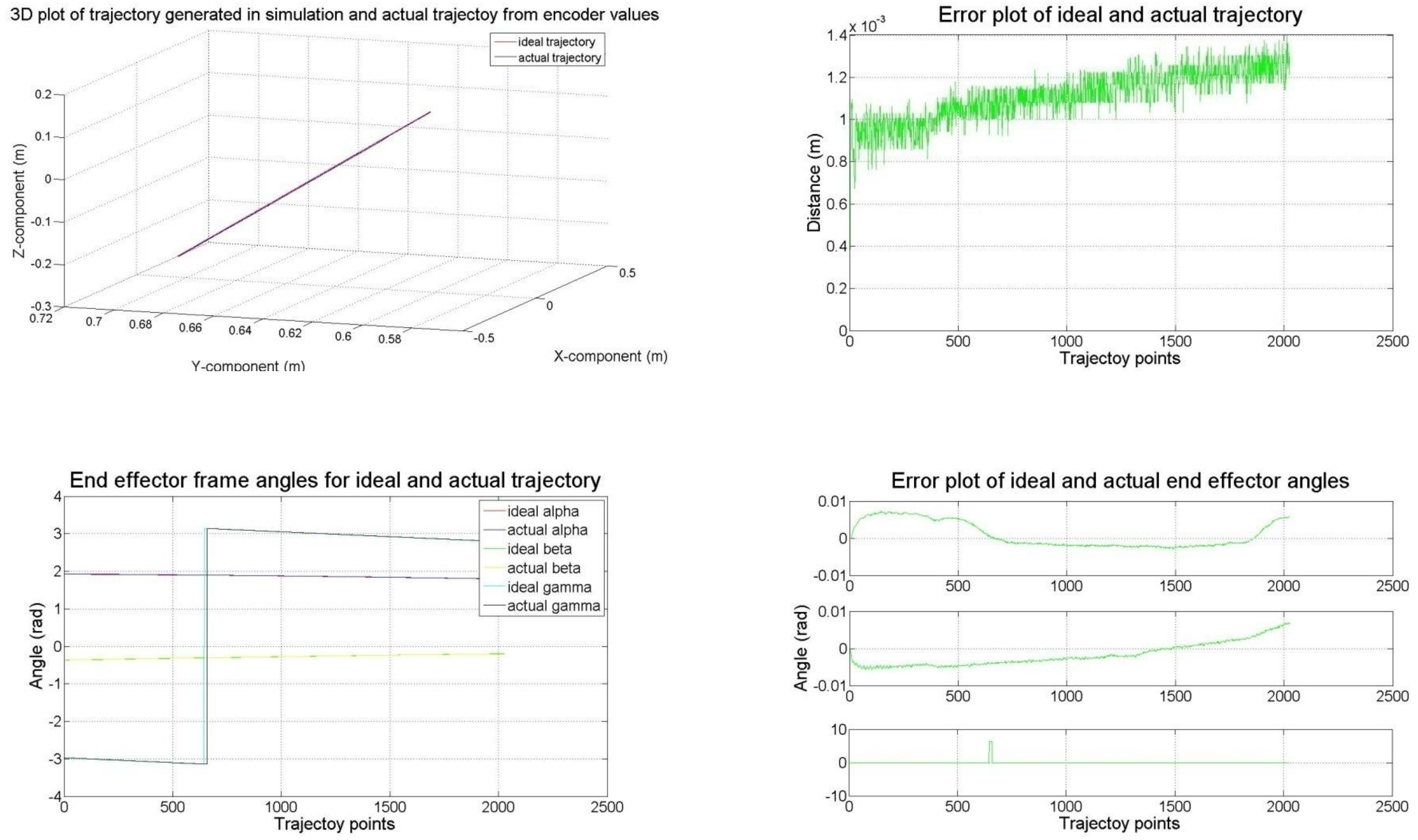


Figure 67: Accuracy plots for sample trajectory 1

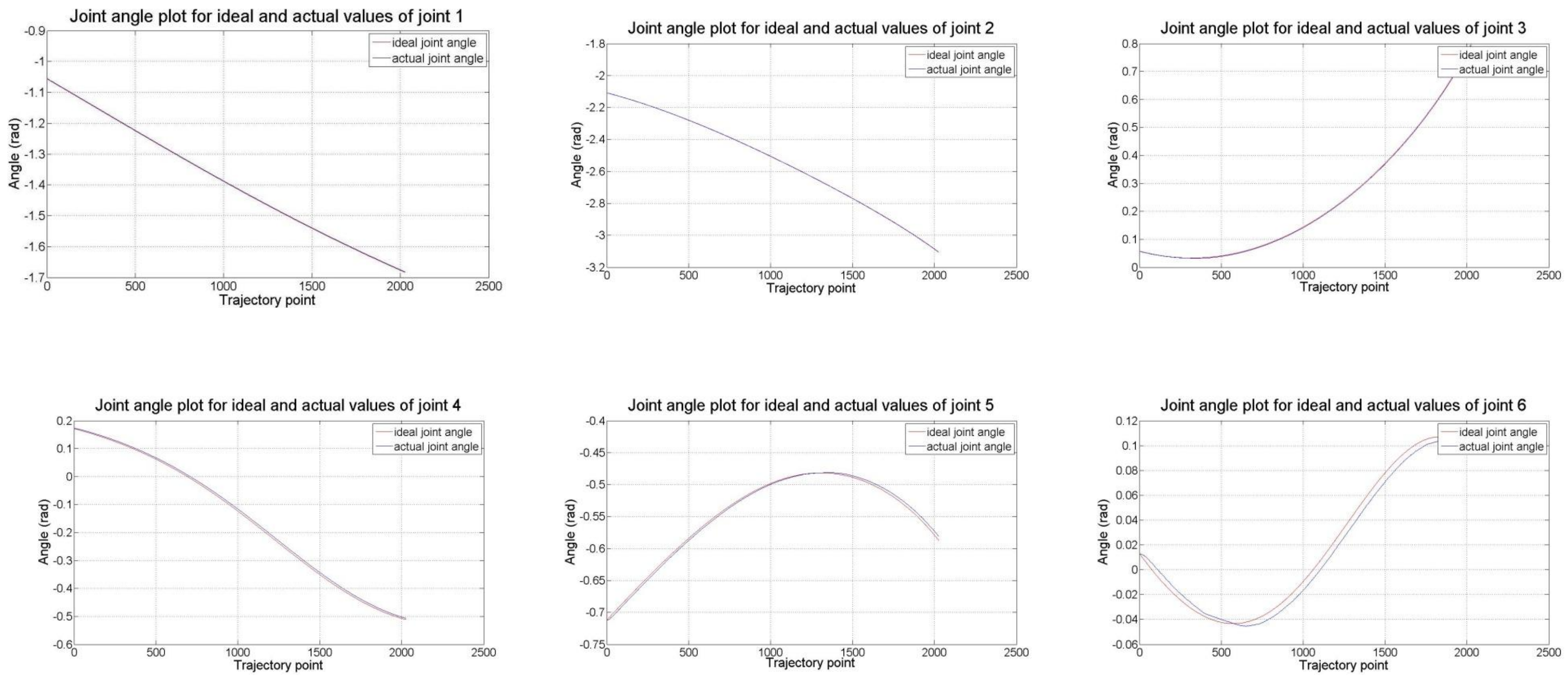


Figure 67 (Continued)

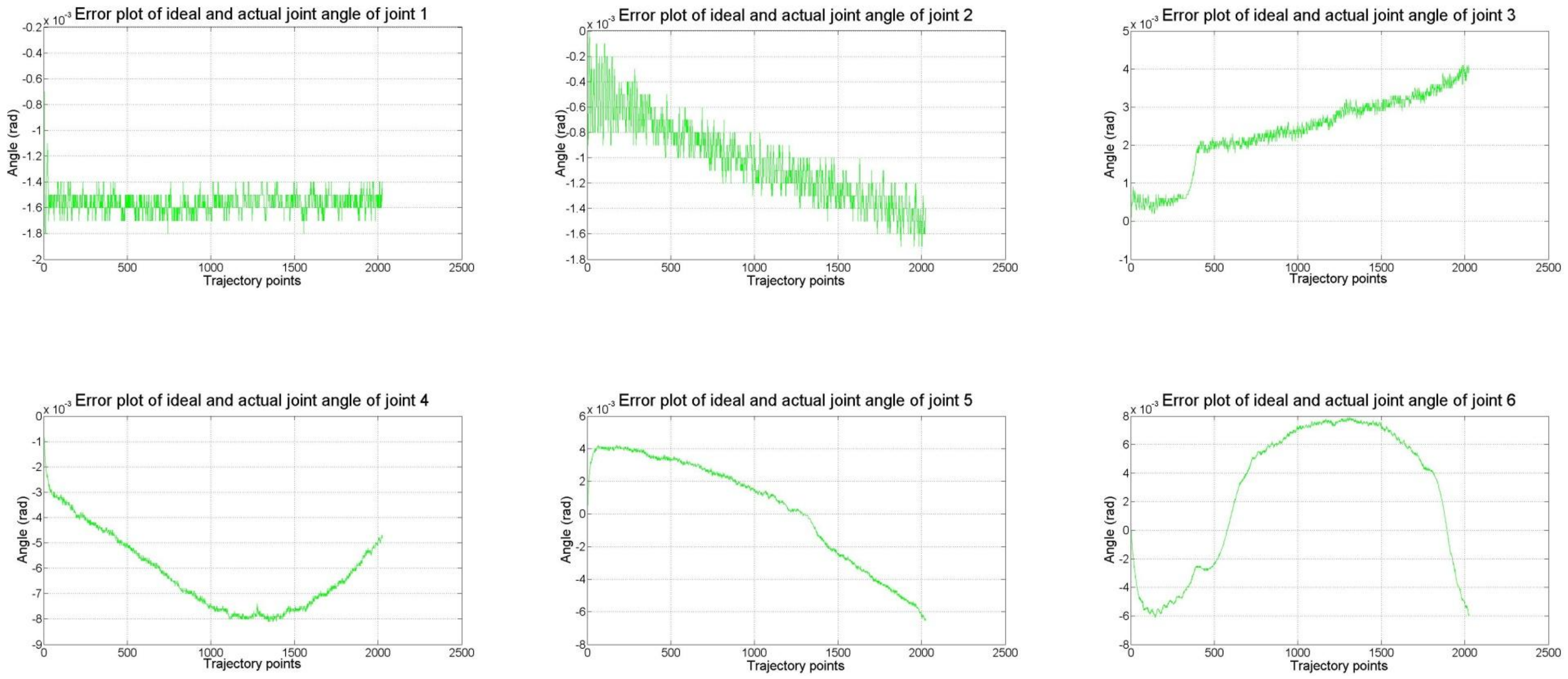


Figure 67 (Continued)

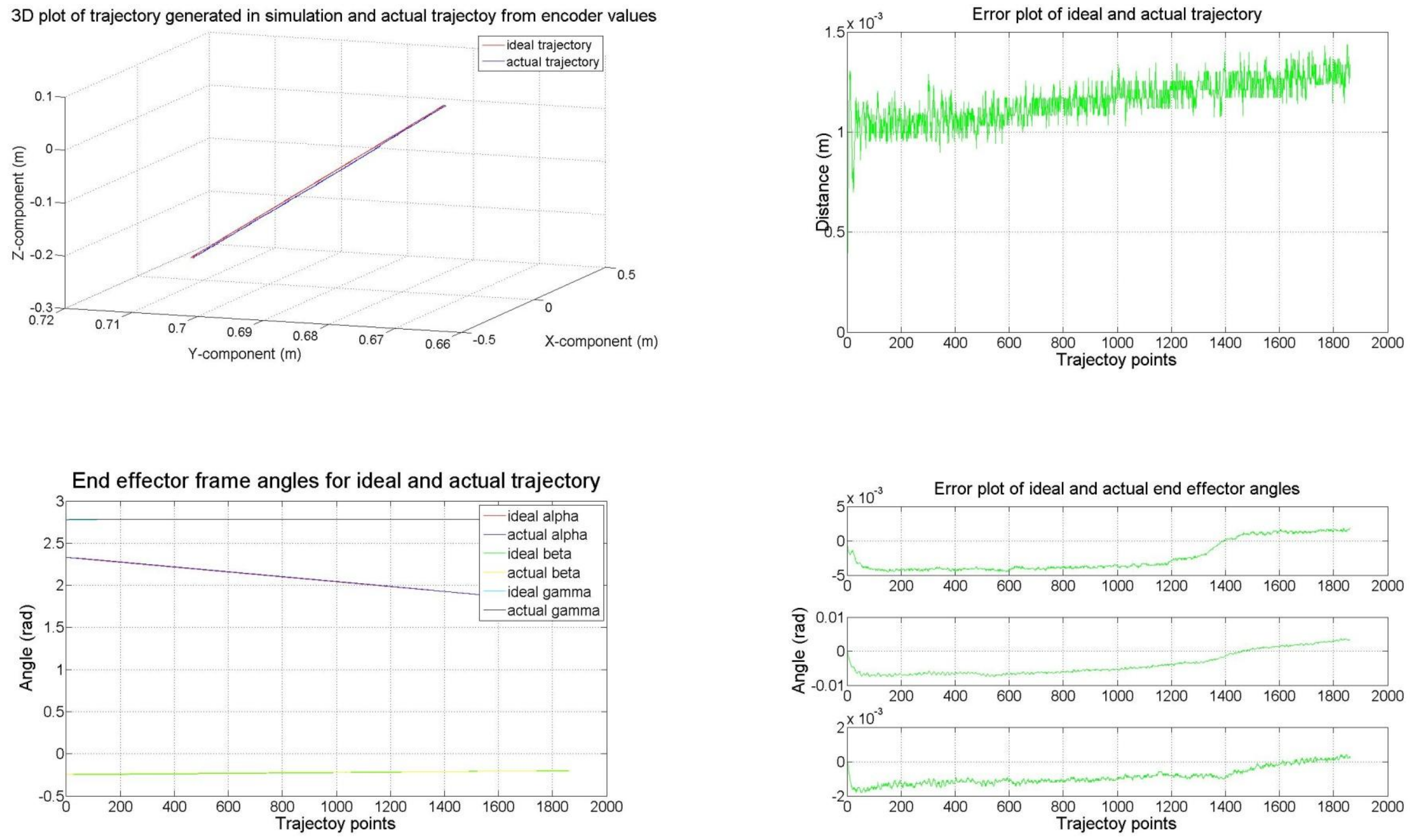


Figure 68: Accuracy plots for trajectory 2

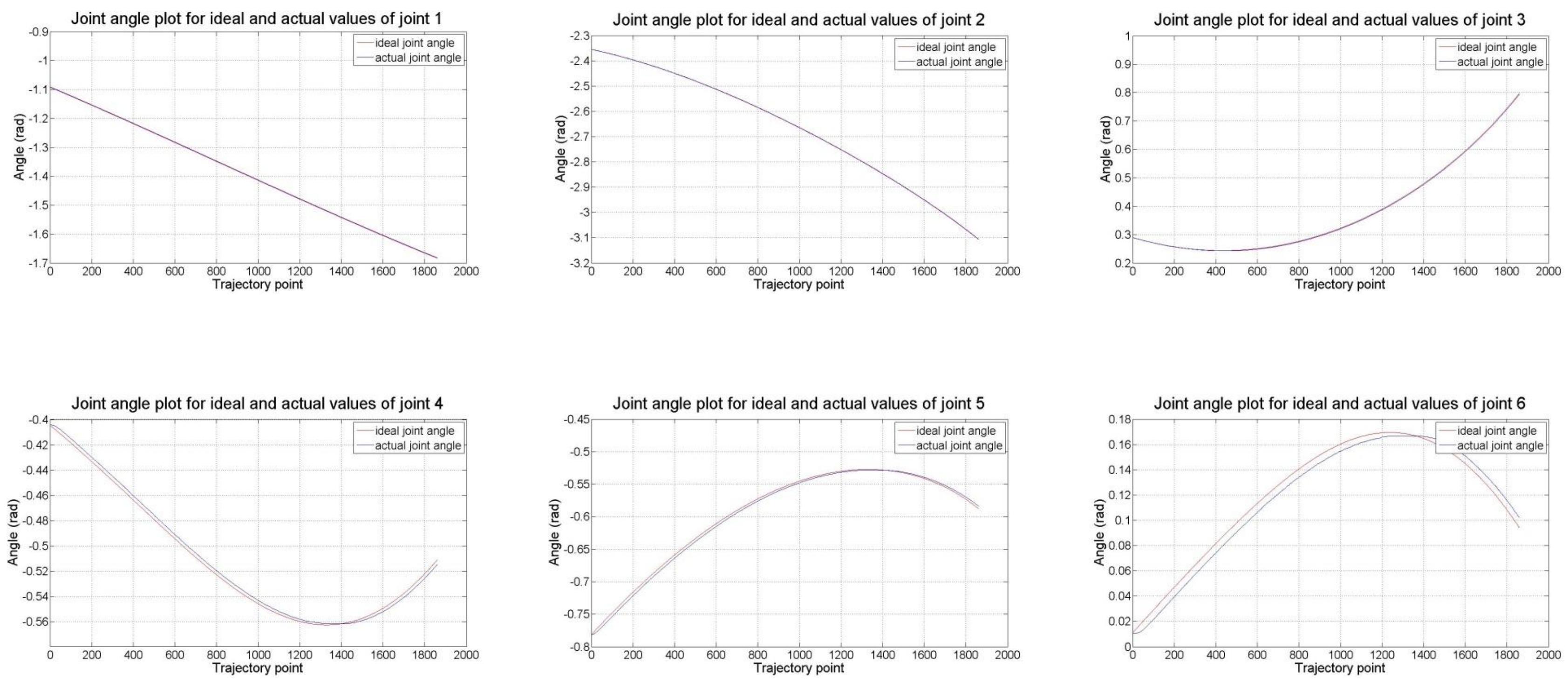


Figure 68 (Continued)

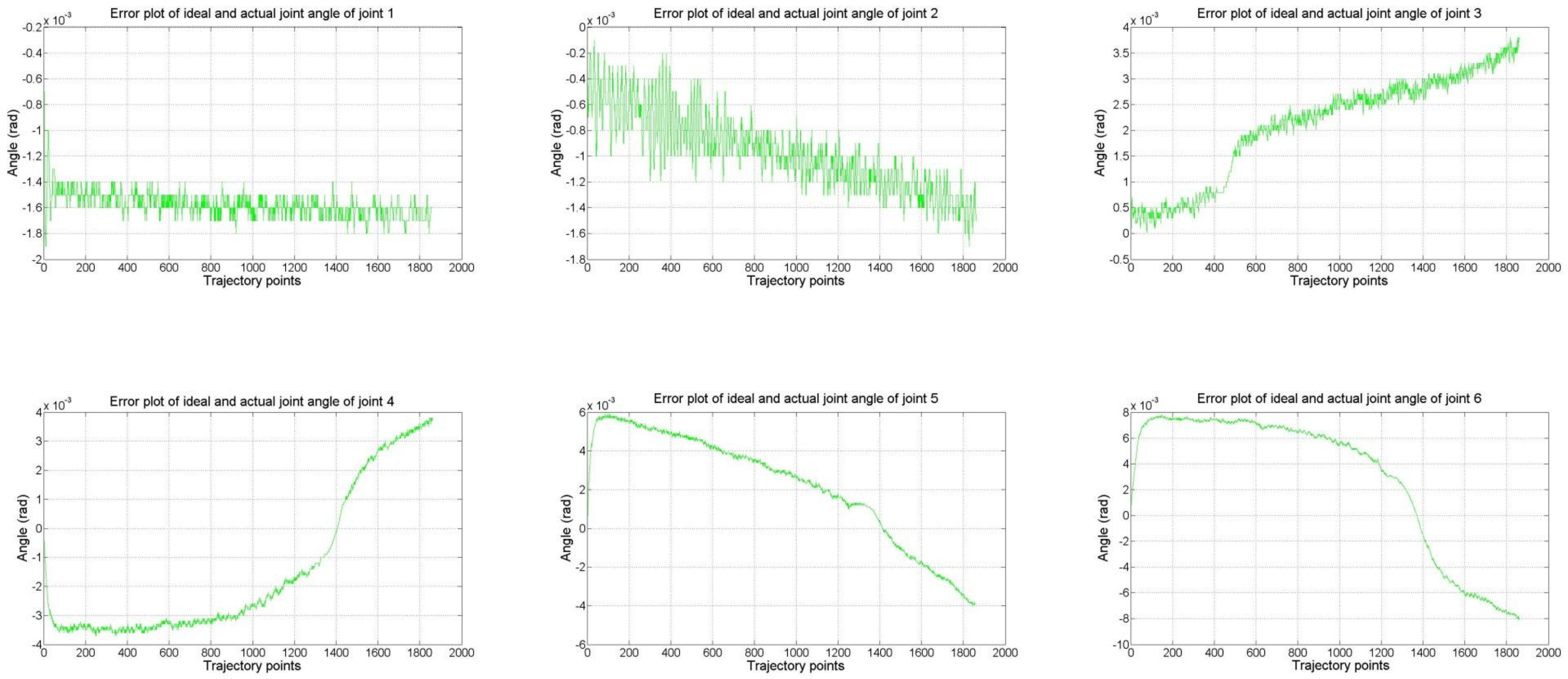


Figure 68 (Continued)

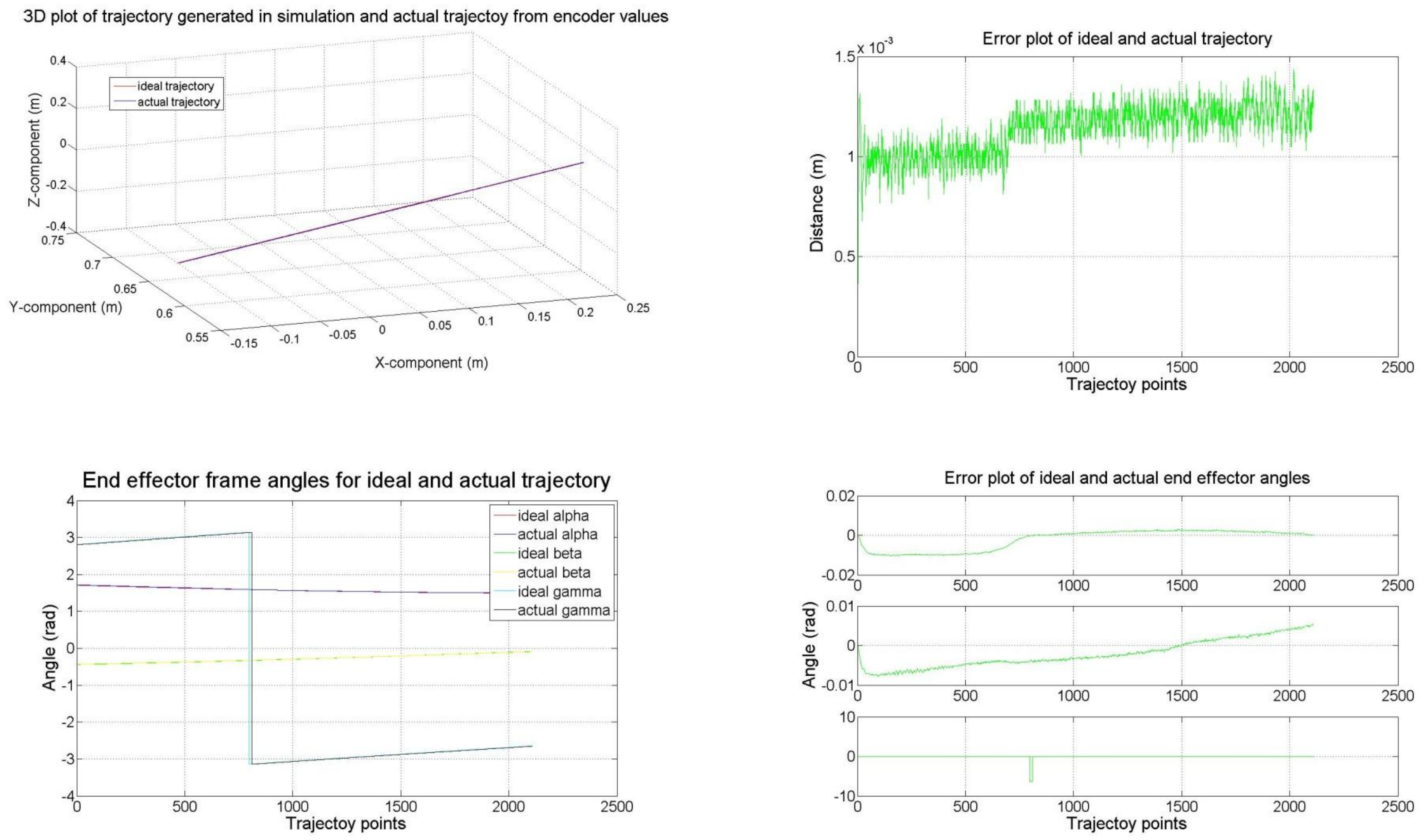


Figure 69: Accuracy plots for trajectory 3

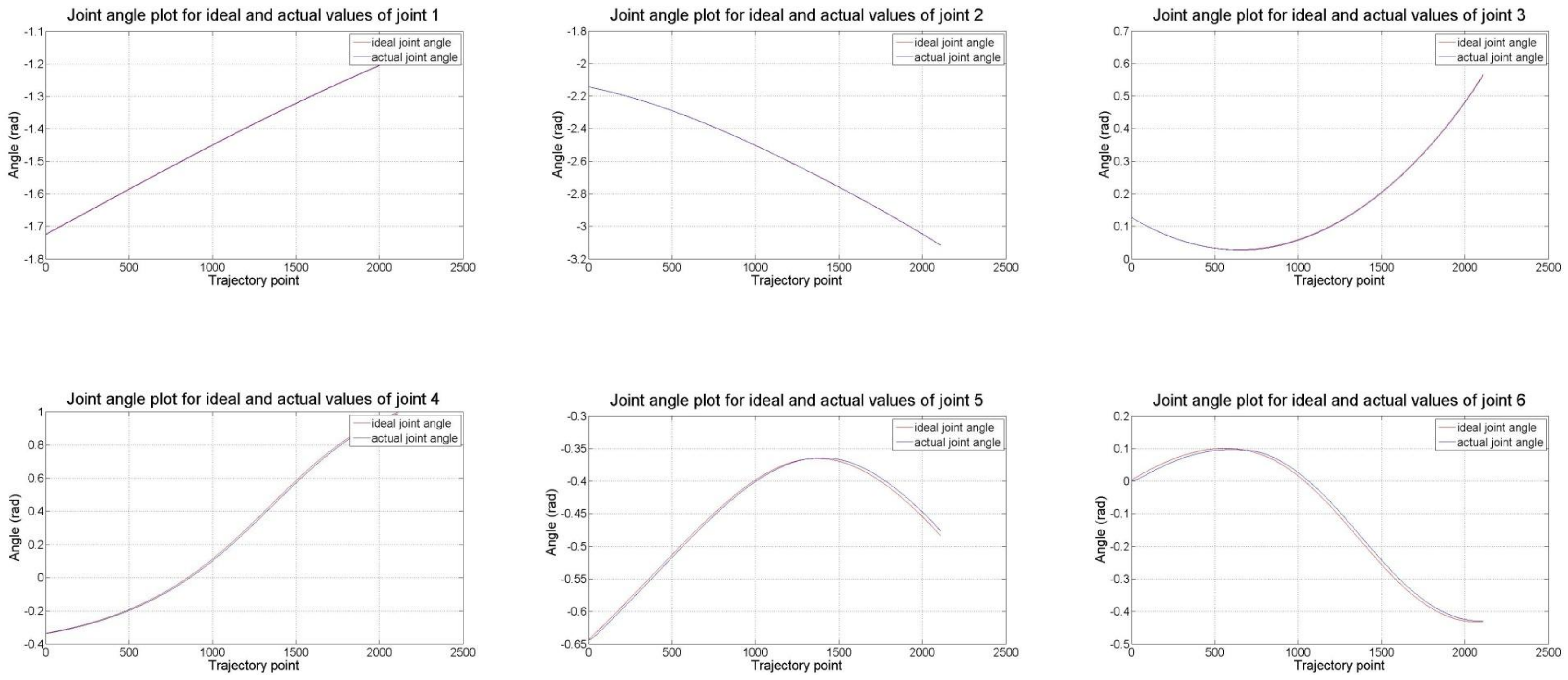


Figure 69 (Continued)

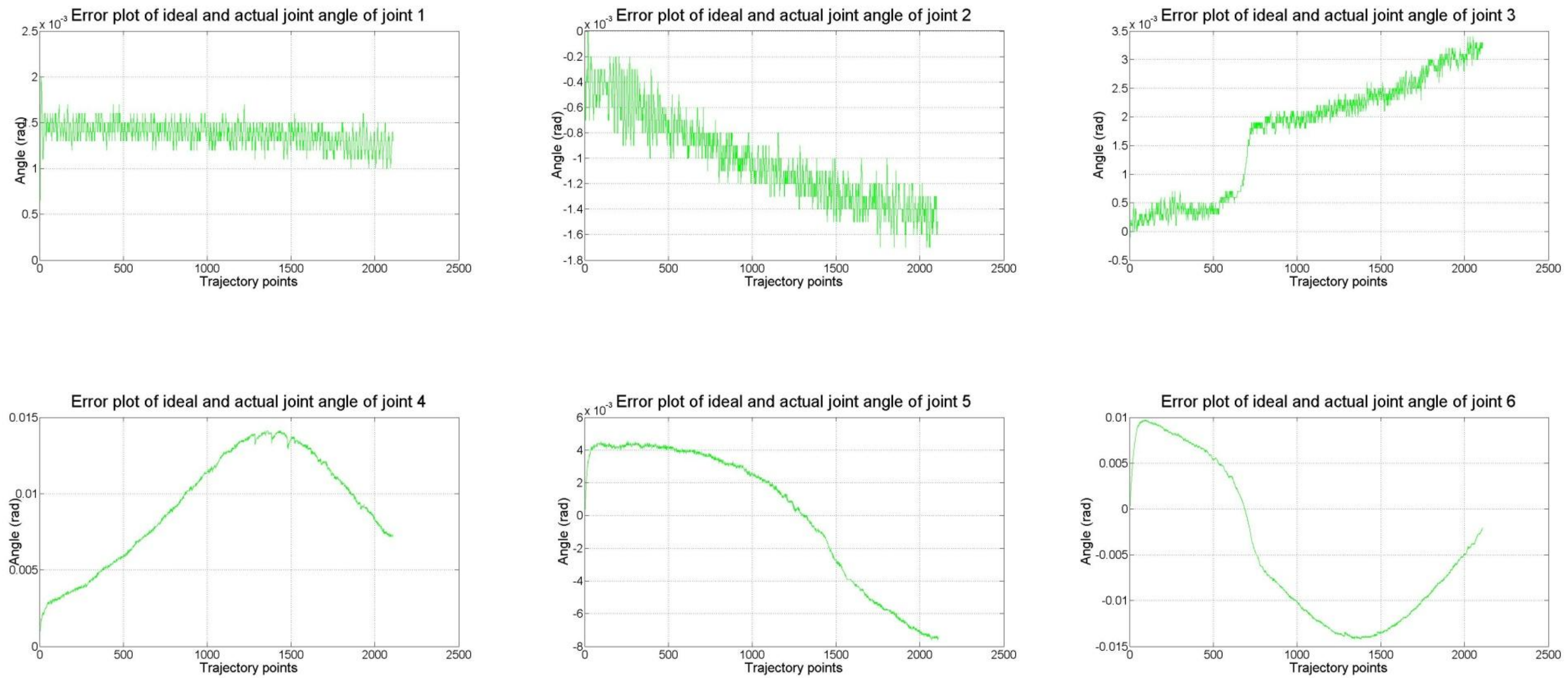


Figure 69 (Continued)

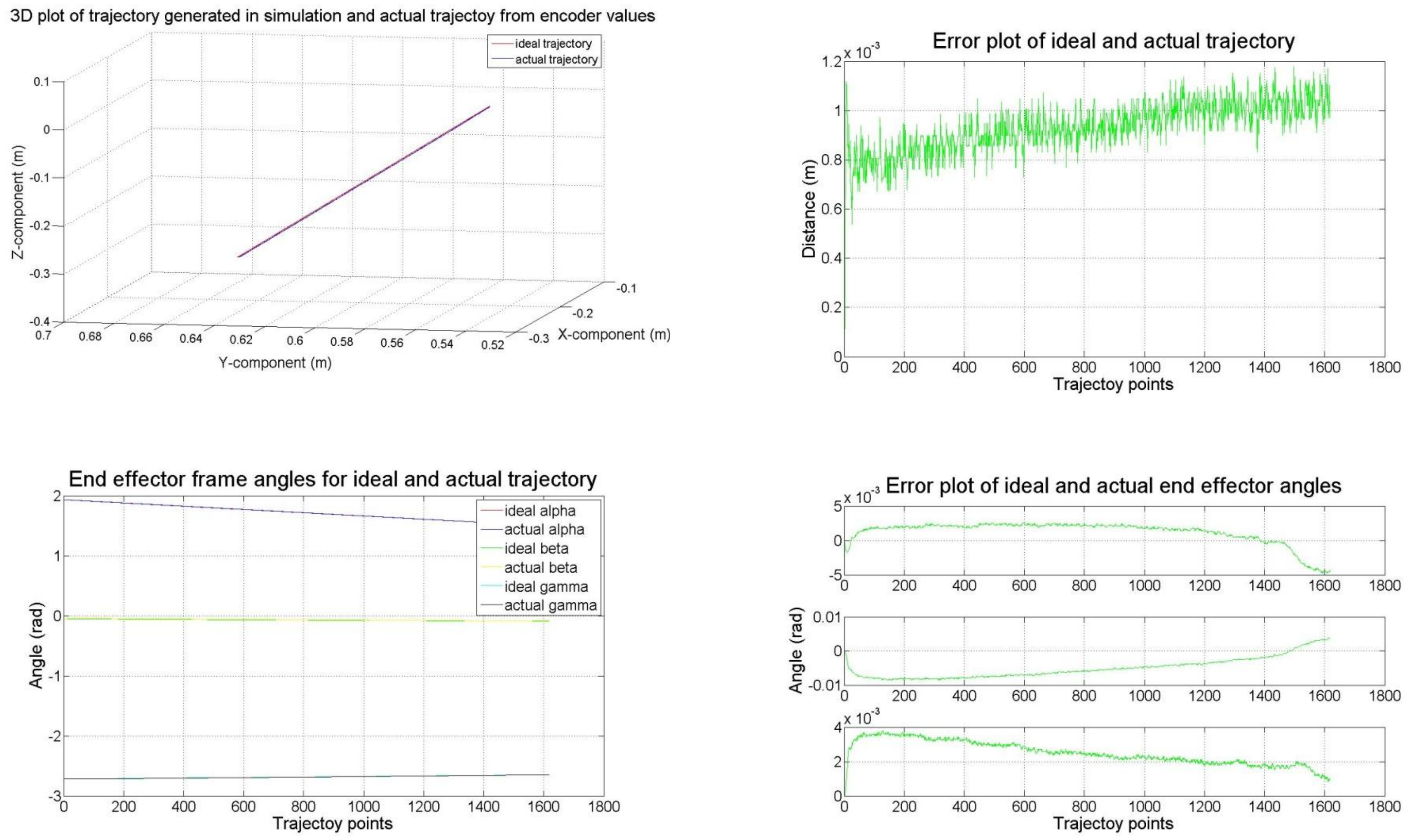


Figure 70: Accuracy plots for trajectory 4

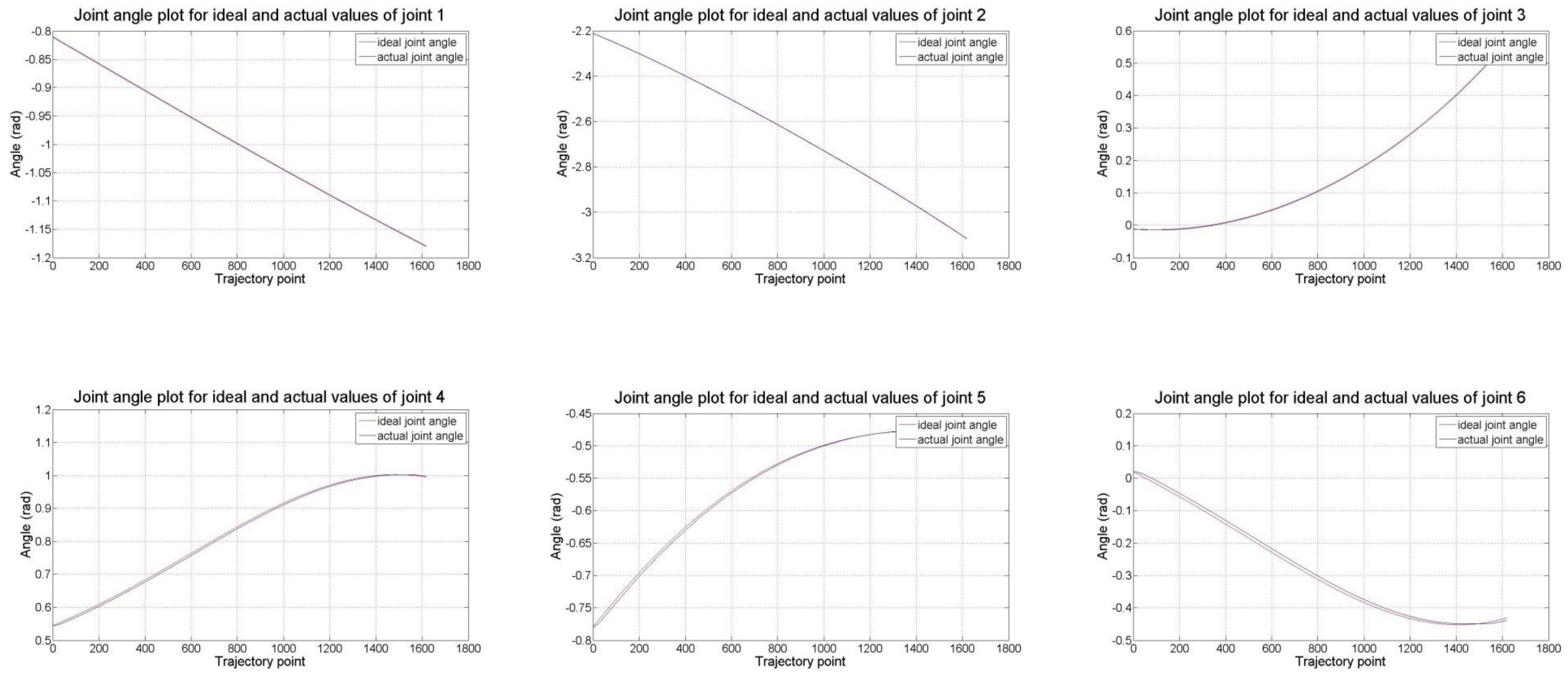


Figure 70 (Continued)

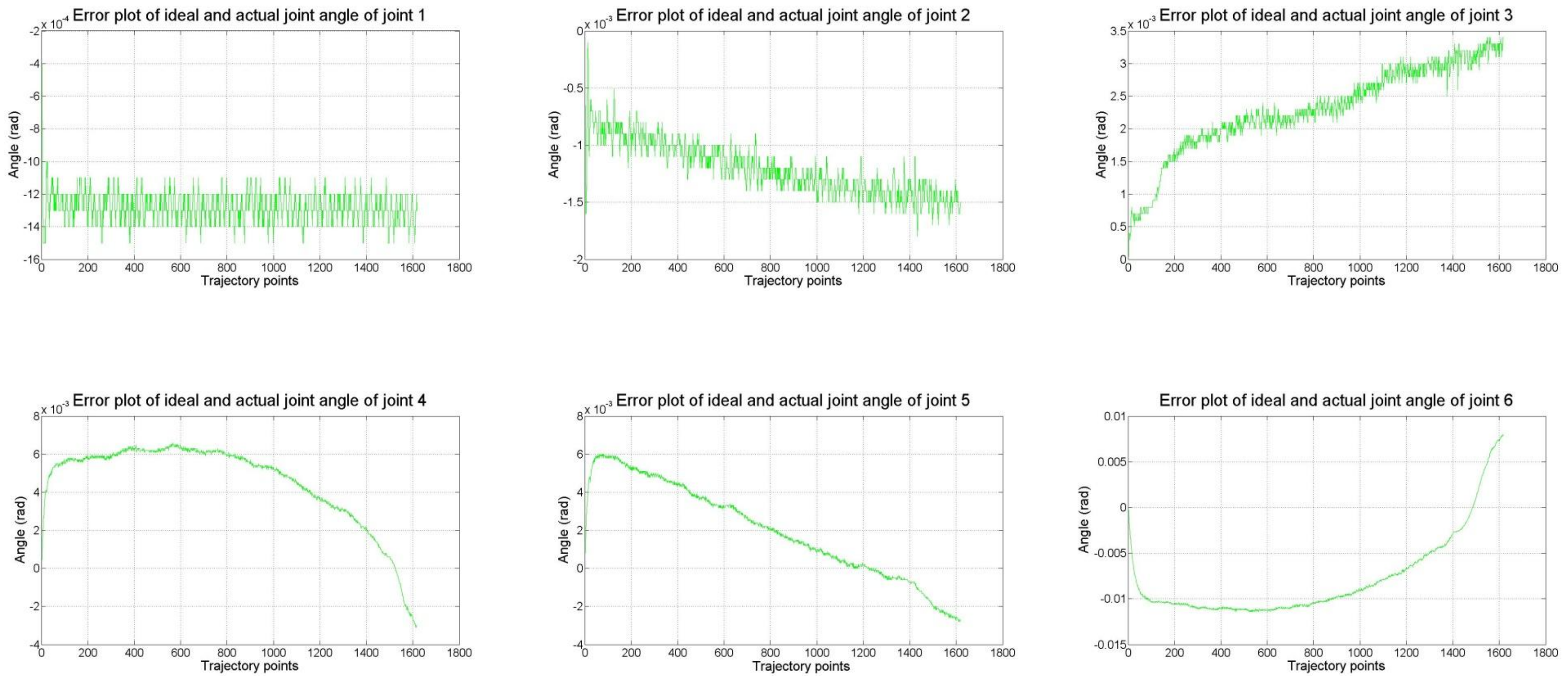
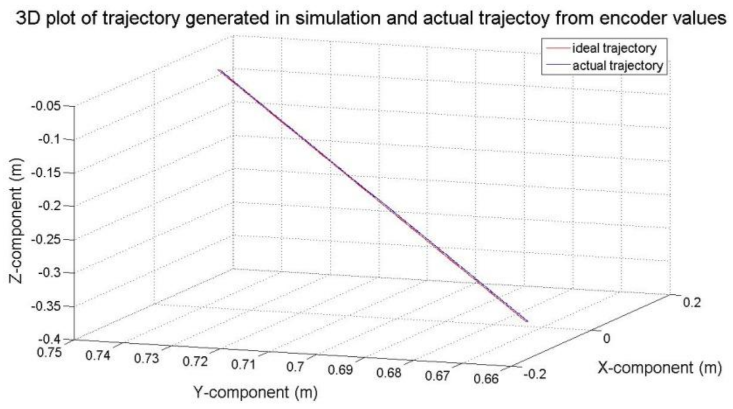
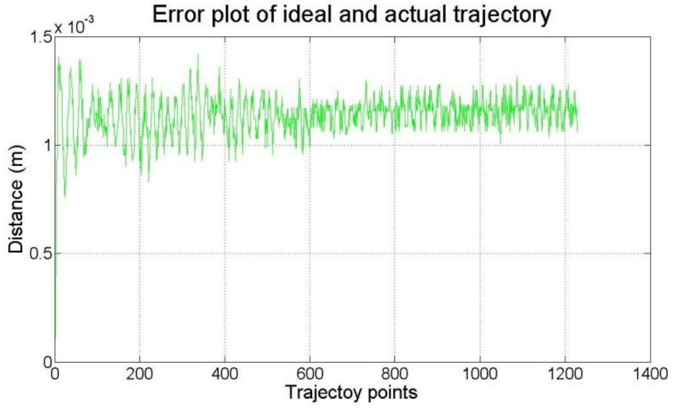
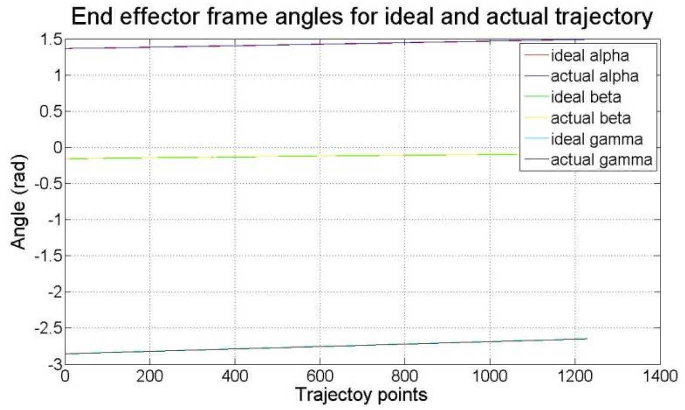


Figure 70 (Continued)







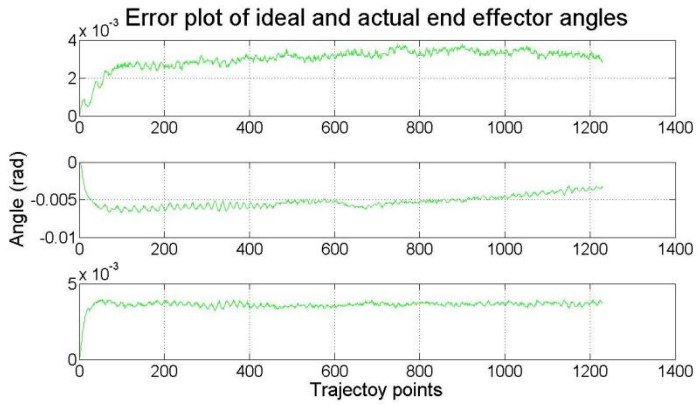


Figure 71: Accuracy plots for trajectory 5

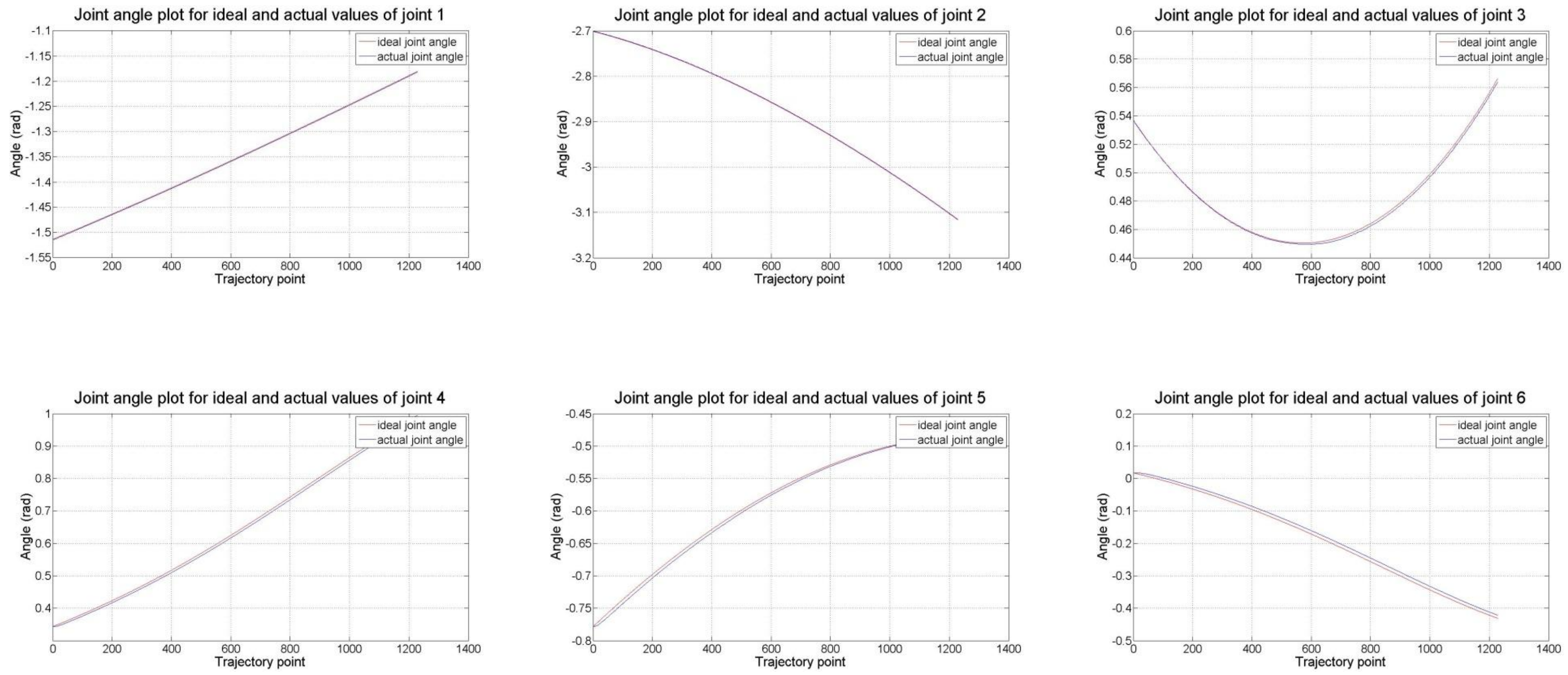


Figure 71 (Continued)

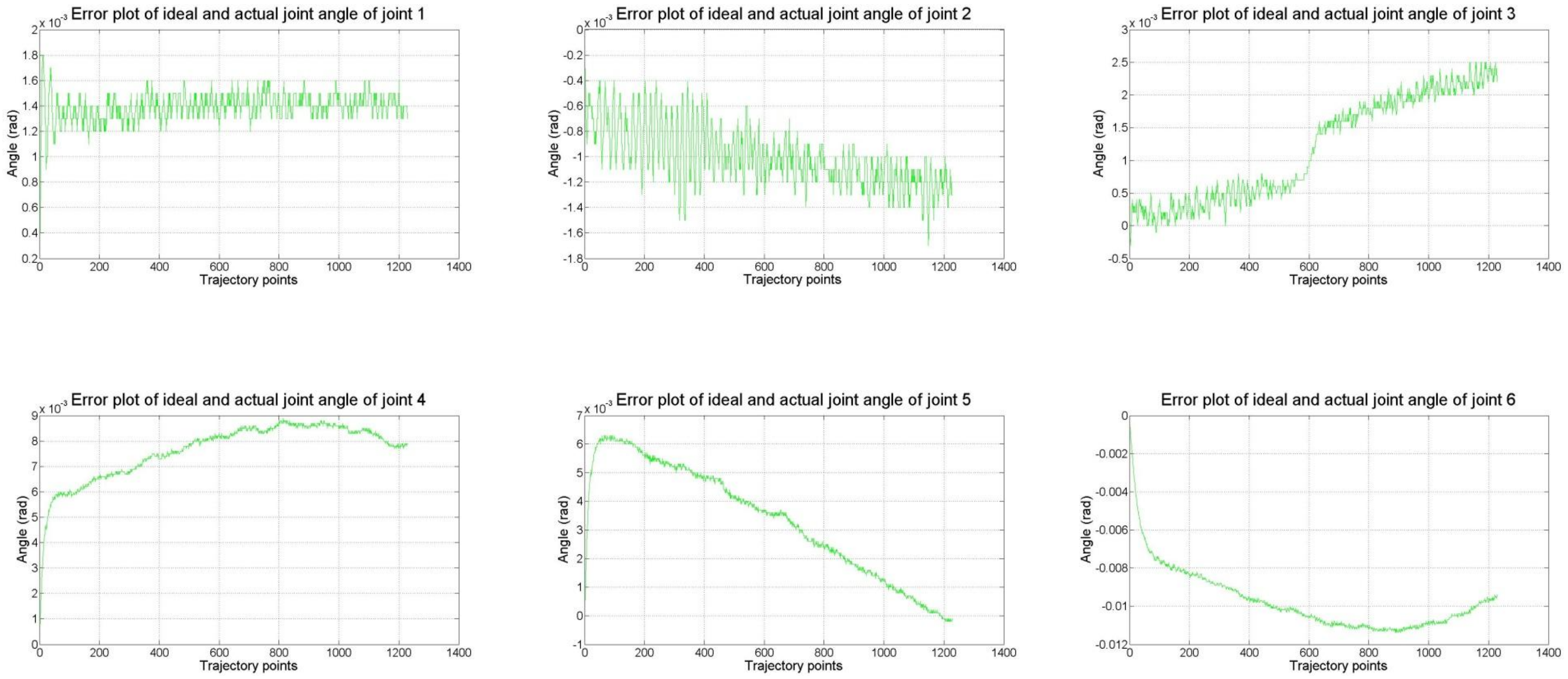


Figure 71 (Continued)

The accuracy results from the plots above are summarized in the following table. The values in the table are show the maximum error in that particular trajectory.

Table 3: Accuracy values for PUMA

Trajectory No.	Error in trajectory points (mm)	Error in joint angles (x 10e-3 rad)
1.	1.4	+/- 8
2.	1.4	8
3.	1.4	-13
4.	1.2	-11
5.	1.4	-11

Repeatibility of the robot is defined as the precision with which it returns to the same point again and again. It can be measured in terms of the difference between the maximum and minimum values in a range of values that the robot attains in going to the specific point. These values could be transformation matrices or joint angles. For determining the repeatibility of the robot, 20 trajectories, 10 each having the same destination point but all having different initial points, were generated. Thus there were two sets of 10 trajectories each, each set having a common destination point. The Cartesian co-ordinates of the final point of each trajectory were recorded after they were derived from the final point transformation matrices which were generated by forward kinematics on the encoder angles at the destination or final point. For the two sets, the final point Cartesian co-ordinates are given in the following table.

Table 4: Cartesian Co-ordinates (m) of final points for first set of trajectories

First set of trajectories (A)			
1	0.2306	0.7165	-0.291
2	0.2306	0.7166	-0.2909
3	0.2314	0.7166	-0.2913
4	0.2323	0.7165	-0.2912
5	0.2323	0.7163	-0.2913
6	0.2316	0.7167	-0.2912
7	0.2315	0.7166	-0.2913
8	0.2316	0.7167	-0.2912
9	0.2315	0.7167	-0.2912
10	0.2315	0.7167	-0.2913

Table 5: Cartesian Co-ordinates (m) of final points for second set of trajectories

Second set of trajectories (B)			
1	-0.1097	0.6635	-0.3595
2	-0.1117	0.6631	-0.3595
3	-0.1098	0.6639	-0.3594
4	-0.1098	0.6639	-0.3594
5	-0.1098	0.6638	-0.3594
6	-0.1098	0.6637	-0.3595
7	-0.1114	0.6636	-0.3593
8	-0.1098	0.6639	-0.3594
9	-0.1107	0.6636	-0.3599
10	-0.1108	0.6641	-0.3596

Next, for each set the distance of the final point of a certain trajectory to the final point of another trajectory was determined and a matrix of distances between the final points of the same set was generated. The matrix of distances for the two sets are given in the following tables. Understandably, the diagonal elements are zero.

From the first set we see that the maximum distance between the same point produced by the robot is 1.8 mm and from the second set it is 2.1 mm. This is the repeatibility of the robot. More data can be gathered to determine if there is larger value of repeatibility. Thus in going to the same point again and again, the robot would end up at a point 2.1 mm from its previous run in the worst case scenario.

Table 6: Matrix of distances between final points in the first set (m)

	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)	A(10)
A(1)	0	0.0001	0.0009	0.0017	0.0017	0.001	0.001	0.001	0.0009	0.001
A(2)	0.0001	0	0.0009	0.0017	0.0018	0.001	0.001	0.001	0.001	0.001
A(3)	0.0009	0.0009	0	0.0009	0.0009	0.0002	0.0001	0.0002	0.0002	0.0001
A(4)	0.0017	0.0017	0.0009	0	0.0002	0.0007	0.0008	0.0007	0.0008	0.0008
A(5)	0.0017	0.0018	0.0009	0.0002	0	0.0008	0.0009	0.0008	0.0009	0.0009
A(6)	0.001	0.001	0.0002	0.0007	0.0008	0	0.0002	0	0.0001	0.0001
A(7)	0.001	0.001	0.0001	0.0008	0.0009	0.0002	0	0.0002	0.0001	0.0001
A(8)	0.001	0.001	0.0002	0.0007	0.0008	0	0.0002	0	0.0001	0.0001
A(9)	0.0009	0.001	0.0002	0.0008	0.0009	0.0001	0.0001	0.0001	0	0.0001
A(10)	0.001	0.001	0.0001	0.0008	0.0009	0.0001	0.0001	0.0001	0.0001	0

Table 7: Matrix of distances between final points in the second set (m)

	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)	A(10)
A(1)	0	0.002	0.0004	0.0004	0.0003	0.0002	0.0017	0.0004	0.0011	0.0013
A(2)	0.002	0	0.0021	0.0021	0.002	0.002	0.0006	0.0021	0.0012	0.0013
A(3)	0.0004	0.0021	0	0	0.0001	0.0002	0.0016	0	0.0011	0.001
A(4)	0.0004	0.0021	0	0	0.0001	0.0002	0.0016	0	0.0011	0.001
A(5)	0.0003	0.002	0.0001	0.0001	0	0.0001	0.0016	0.0001	0.001	0.0011
A(6)	0.0002	0.002	0.0002	0.0002	0.0001	0	0.0016	0.0002	0.001	0.0011
A(7)	0.0017	0.0006	0.0016	0.0016	0.0016	0.0016	0	0.0016	0.0009	0.0008
A(8)	0.0004	0.0021	0	0	0.0001	0.0002	0.0016	0	0.0011	0.001
A(9)	0.0011	0.0012	0.0011	0.0011	0.001	0.001	0.0009	0.0011	0	0.0006
A(10)	0.0013	0.0013	0.001	0.001	0.0011	0.0011	0.0008	0.001	0.0006	0

7.2. Results on Laser Assisted Telerobotic Task Execution

This section of the results is the most important from the thesis hypothesis point of view. This results presented in this section validates the hypothesis that the laser assisted method of task execution is faster, more accurate and easier compared to the unassisted mode of task execution.

For generating results, 10 subjects were used for testing on the system. The eligibility criteria for the subjects were that were conversant in English and were mentally and physically sound. Each subject was asked to perform a pick-and-place task thrice in two modes, one was the laser assisted mode of doing the task and the other was the unassisted mode. The laser assisted mode comprised of using the laser information along with machine intelligence for assisting the user in a Traded Control formulation. The various features of laser based task assistance like autonomous trajectory generation, autonomous end effector alignment, autonomous path planning that were explained in detail in section 6.5 of chapter 6 were used. The unassisted mode was the general case of direct teleoperation where the manipulator was controlled by the master manipulated by the human without any machine assistance. The operation of this mode is explained in section 6.1 of chapter 6. For each run, the time and the end effector Cartesian co-ordinates were recorded. Also, the subjects' experience in doing the task in the two modes was noted.

Before starting the tests the subjects were given sufficient time to acclimatize with the system. They were given more time specifically in executing the general teleoperation mode as it is non-intuitive for humans to effect manipulation in a remote environment when they are decoupled from it and there is not sufficient telepresence. The subjects were asked to start testing only when the author realized that the subject had reached a comfort level with operating the system that was uniform with other subjects. In general, each subject was given 5 to 6 trails in teleoperation mode and 2 to 3 trials in the laser assisted mode.

The experimental set up for the pick-and-place task is shown in the following figure. The cup is the target object which is to be picked from its current location, the source and placed at the cross mark on the orange sticky which is the destination point. The folder between the source and destination points simulates an obstacle. As seen, the source and the destination are such that their x, y and z Cartesian co-ordinates are different. The task here is to start from the ready position, go towards the target, pick it up by grasping it, take it to a certain height so that it is clear from the obstacle and then place it over the destination point. The height to which the target is raised depends on the user estimation in the general teleoperation mode and it is currently hard coded for the laser assisted mode. However as mentioned in chapter 6, the information from the laser can be used to determine the height of the tallest obstacle and the target can then be raised accordingly autonomously. Also in teleoperation while teleoperating the PUMA end effector towards the target the user was instructed has to try and follow the shortest possible path. Again while moving the target from the source to the destination, the user had the option to either follow a rainbow like path which would be the shortest or follow an inverted ‘U’ shaped path similar to the one generated in autonomous mode. This option is given to the user so that human intelligence, which the machine lacks, that of generating the shortest path in an unstructured environment can be utilized for a better comparison.

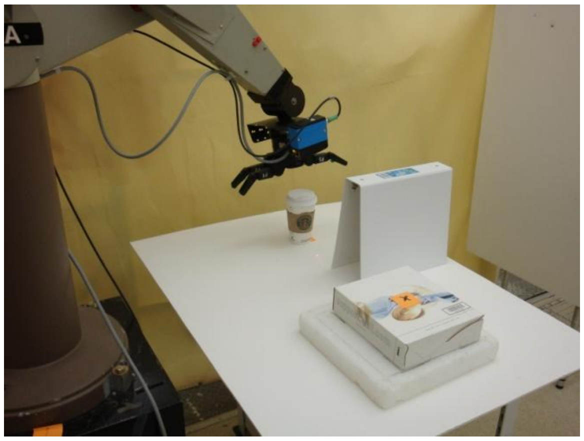


Figure 72: Experimental set up for pick-and-place task - side view

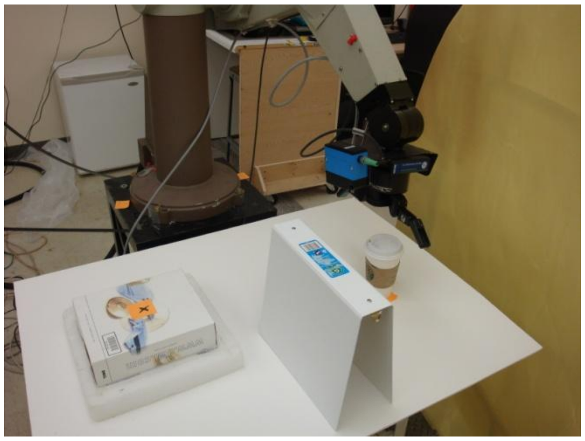


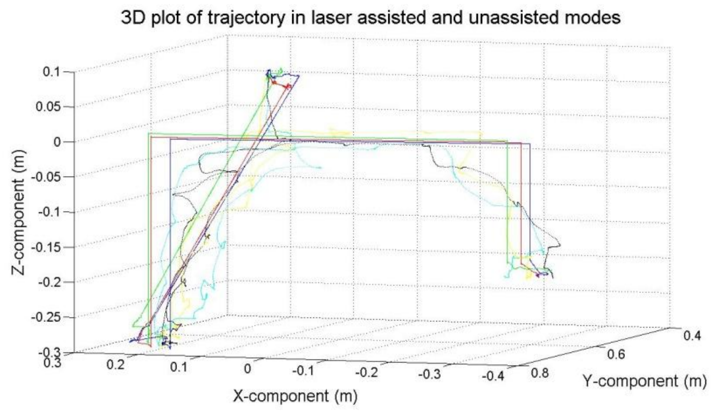
Figure 73: Experimental set up for pick-and-place task - front view

The comparison of accuracy in task execution using the laser assisted and the unassisted modes is presented in a qualitative form. This is in the form of the end effector Cartesian co-ordinates along the path generated by the PUMA in the two modes.

Figure 74 presents these paths. Each plot shows the path executed by each subject in the two modes. Understandably, the straight line paths are those generated using the laser assisted mode and the unsteady lines are those generated using the general teleoperation mode. As we can see clearly from the plots, the subjects had tremendous difficulties in teleoperating the PUMA without any assistance from the machine. They had problems with maintaining a steady path, avoiding deviating away from the trajectory, keep a continuous motion of the PUMA and so on. Thus the PUMA motion was highly unstable. In contrast the motion generated by laser assisted telerobotic control was stable, continuous and accurate.

Table 8: Legend for Figure 73

🟥 🟦 🟩	Laser Assisted Modes
🟨 ⬛ 🟦	Unassisted Modes



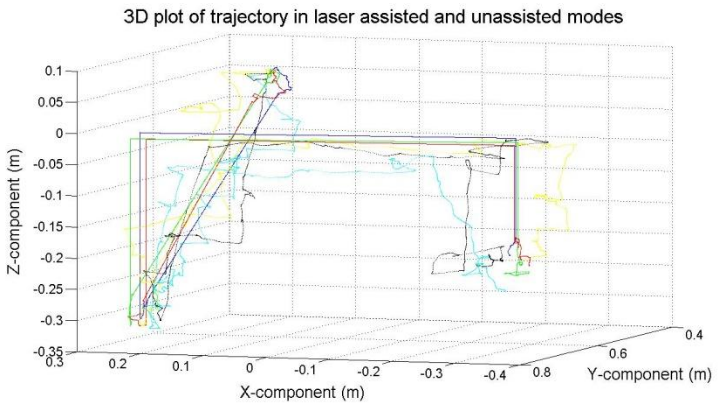
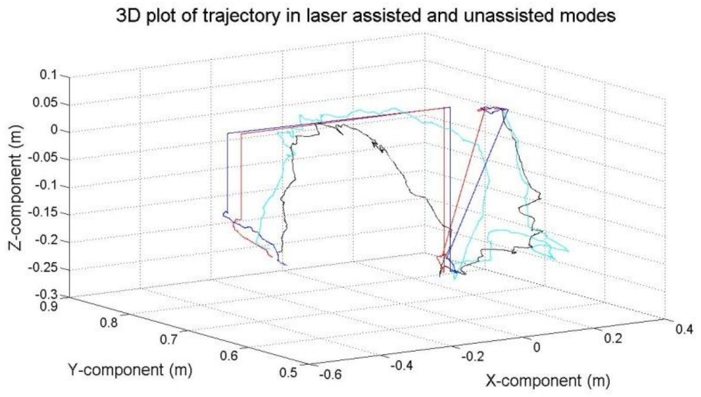
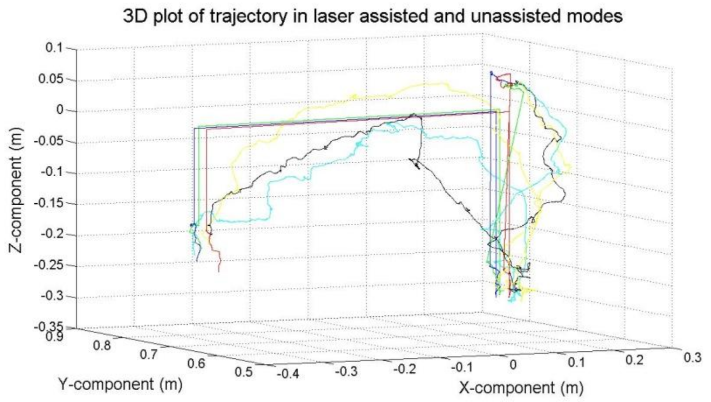
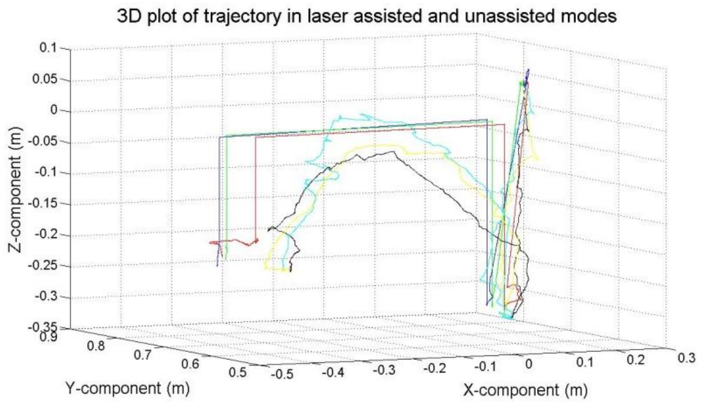


Figure 74: Accuracy in path following plots for pick-and-place task







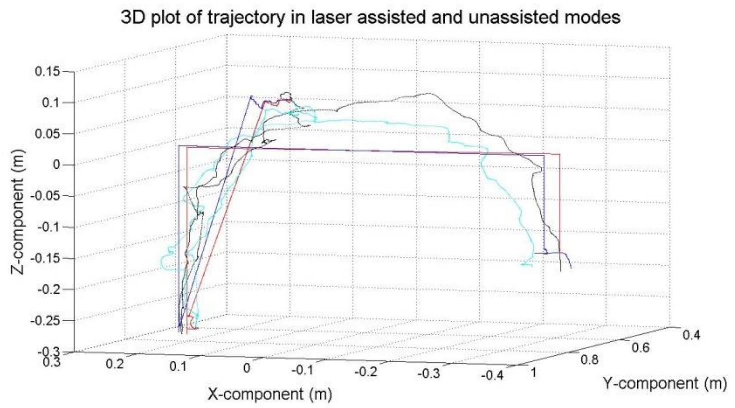


Figure 74 (Continued)

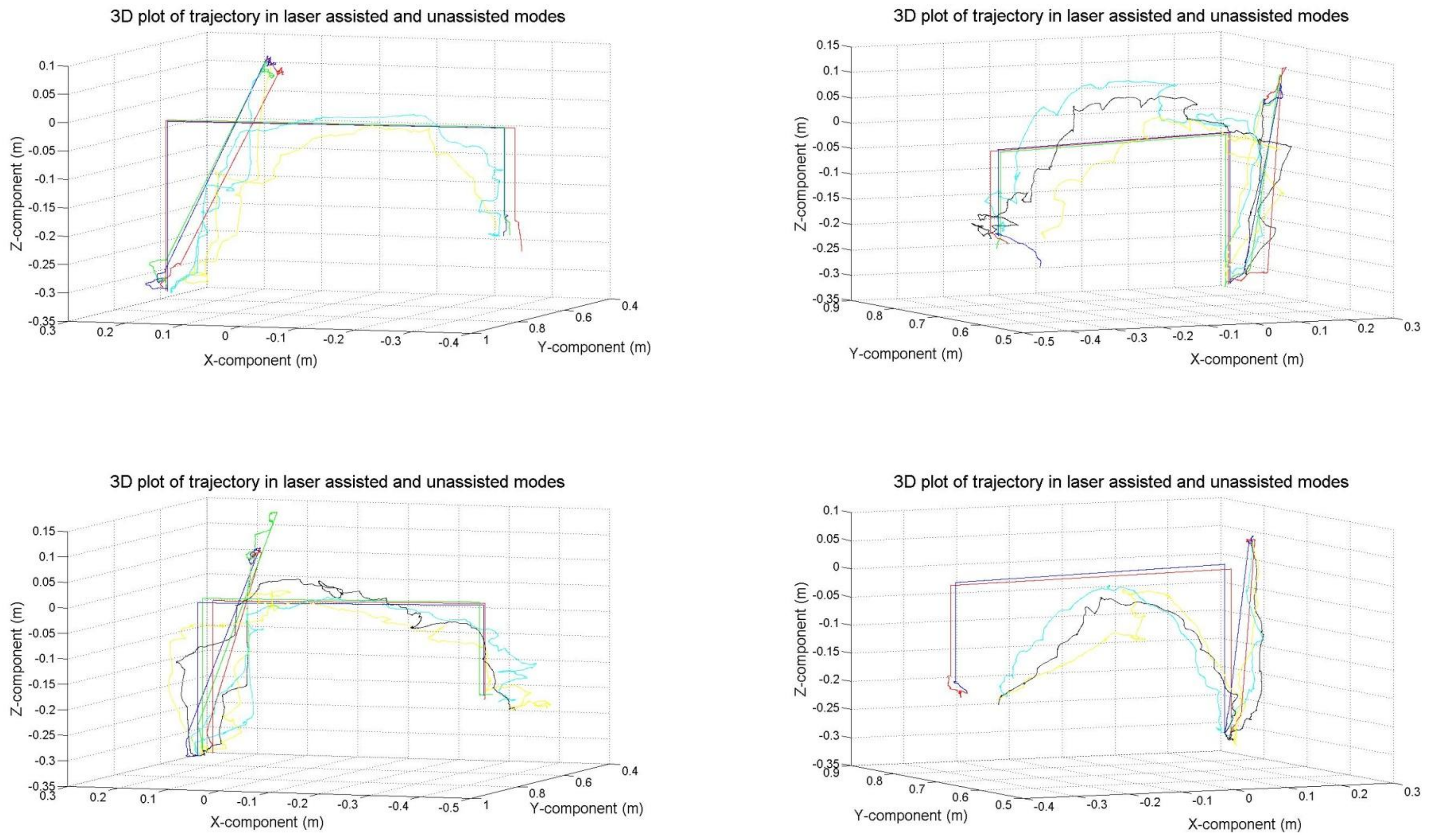


Figure 74 (Continued)

We also observe loops of paths when the PUMA is near the pick-up point. These are due to the difficulty the users face in orienting the arm properly so that it is in a configuration compatible for the execution of the next step in the task e.g. the hand is aligned to the surface for grasping the target. This is because of the Cartesian mapping. Again switch to the joint control mode could help in this case. In case of laser assisted control the laser assisted autonomous surface alignment provides an easy solution.

The time that each subject spent to complete a task in each run was recorded. The average of the time for each mode for a subject was calculated and a comparison is presented below.

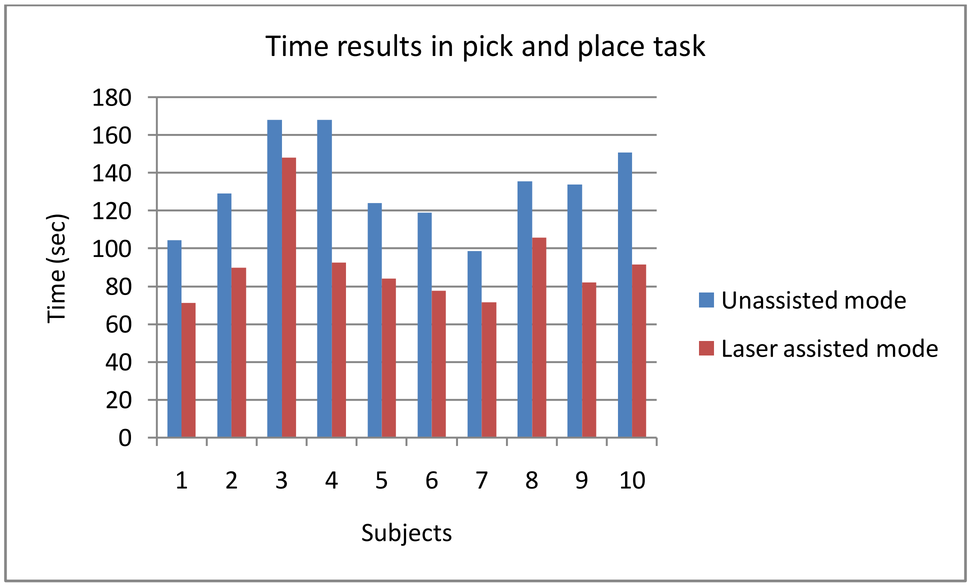


Figure 75: Time to execute pick-and-place task in unassisted and laser assisted modes

From the chart we see that for each subject the time taken by that subject in executing the unassisted mode was always more than that taken to execute the same task in the laser assisted control mode. In the laser assisted mode the subjects are only involved in making high level decisions like pointing the laser to specific points and pressing specific keys for specific functions. They are not involved in making the PUMA follow trajectories or orienting the hand properly so that the target is ready to be grasped. These time consuming sub-tasks are executed by the machine which is fast and accurate. The machine is good in computations and in precision execution but lacks decision making which is filled in by the human, Though there is some initial time spent in pointing the laser to specific locations in the environment and pressing keys on the keyboard to record those locations so that virtual geometries and paths can be generated later on, once these locations are recorded then the user only commands the robot to execute the portion of the task and the machine handles the execution part. The human is a mere supervisor.

Though time and accuracy results demonstrate the improvement in task performance, the user experience in executing the task is also very important. One of the points hypothesized is that the laser assisted mode reduces the cognitive load or the mental load on the user while executing the task. This is indeed the case as we shall see. For recording user experience, the subjects were asked about their experience in executing the task. They were asked to compare the two modes based on the mental load they experienced in executing the task in each mode. Some of the responses were:

1. In the laser assisted mode the user is not involved in computations like shortest path generation and following, simply following a steady path or orienting the hand to make it compatible for grasping. All these computations put a lot of mental load on the user in general teleoperation. The user simply gives commands and does not have to worry about anything. This is simply fantastic!
2. Generating task plans and commands by picking up points with the laser is very easy. The user is just not concerned about the complexities of the task.
3. There is a lot of mental load on the user in teleoperating for complete task execution. Sometimes it is frustrating to teleoperate to get the arm back into its initial configuration or when orienting it to align with, say a surface. The user can get used to the load with practice and with practice will also know better as to what inputs from the master translate into what outputs at the slave. However, the user will still be involved in the task on a continuous basis.
4. Simply moving the wrist in laser assisted mode while teleoperating to point the laser to certain points in the workspace makes it a lot easier for the user to do the task.

The user does not have to worry about going over the obstacles or to orient the hand for proper grasping which makes it very easy for doing the task.

5. Orienting the end effector so that it is in a proper configuration for the next step of the task or when the arm reaches an awkward configuration and the user has to orient it to get it back into its original configuration from where it is easier to carry on with teleoperation is one of the most mentally challenging activities. It makes you mentally and physically tired.
6. Autonomous mode takes care of precision task execution leaving me no room for making errors. It is significantly easier to do the task in laser assisted mode.

Thus we see that it is a lot easier for the user to execute a task using laser assisted control and the user is relieved of much cognitive load in controlling the arm through a trajectory or orienting it. The user would always prefer the combined autonomous and teleoperation control mode in which the user is a supervisor and the machine executes computationally intensive and precision tasks.

7.3. Impact of New Laser Data Processing Scheme on Robot Performance

In section 3.3 of chapter 3 the author had introduced a new laser data acquisition and processing scheme in which the laser was calibrated and its data was filtered with a 50 point moving average at a rate of 100 Hz. The new characteristics combined with data filtering lead to improved accuracy and reproducibility of the distance values measured by the sensor. Here we demonstrate this improvement in range data measurement and later on also demonstrate the impact of the improved laser readings on the telerobotic task execution in the unstructured environment. The accuracy and repeatability of the telerobotic system improves as a result of the new laser scheme which is useful in task execution in the unstructured environment.

The following plots show around a thousand distance outputs from the laser sensor for eight distances with the new and old laser calibration and filtering schemes. The two schemes are compared by juxtaposing the plots for the same distance by the two schemes.

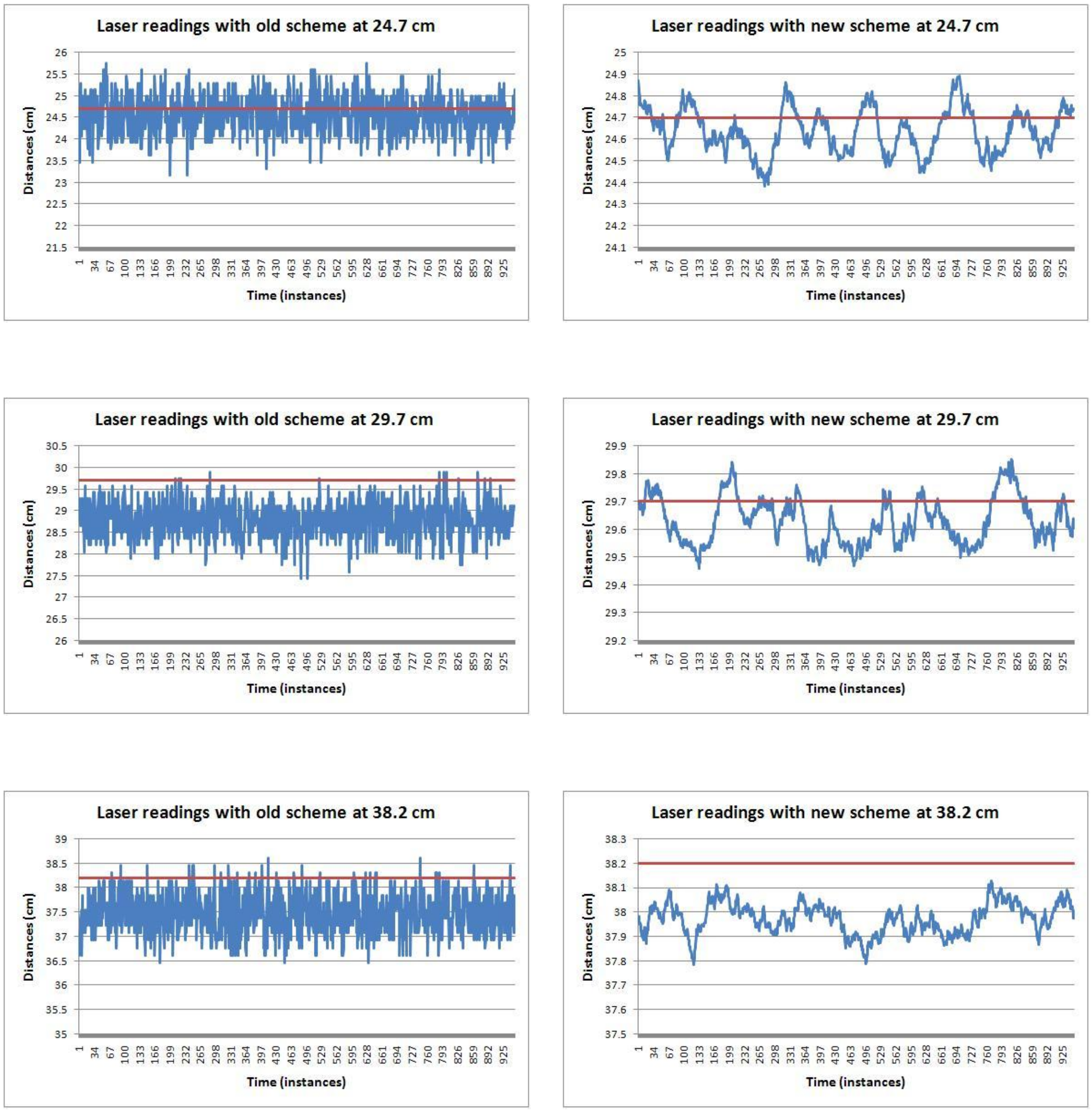


Figure 76: Laser distance measurements for 8 distances using old and new schemes

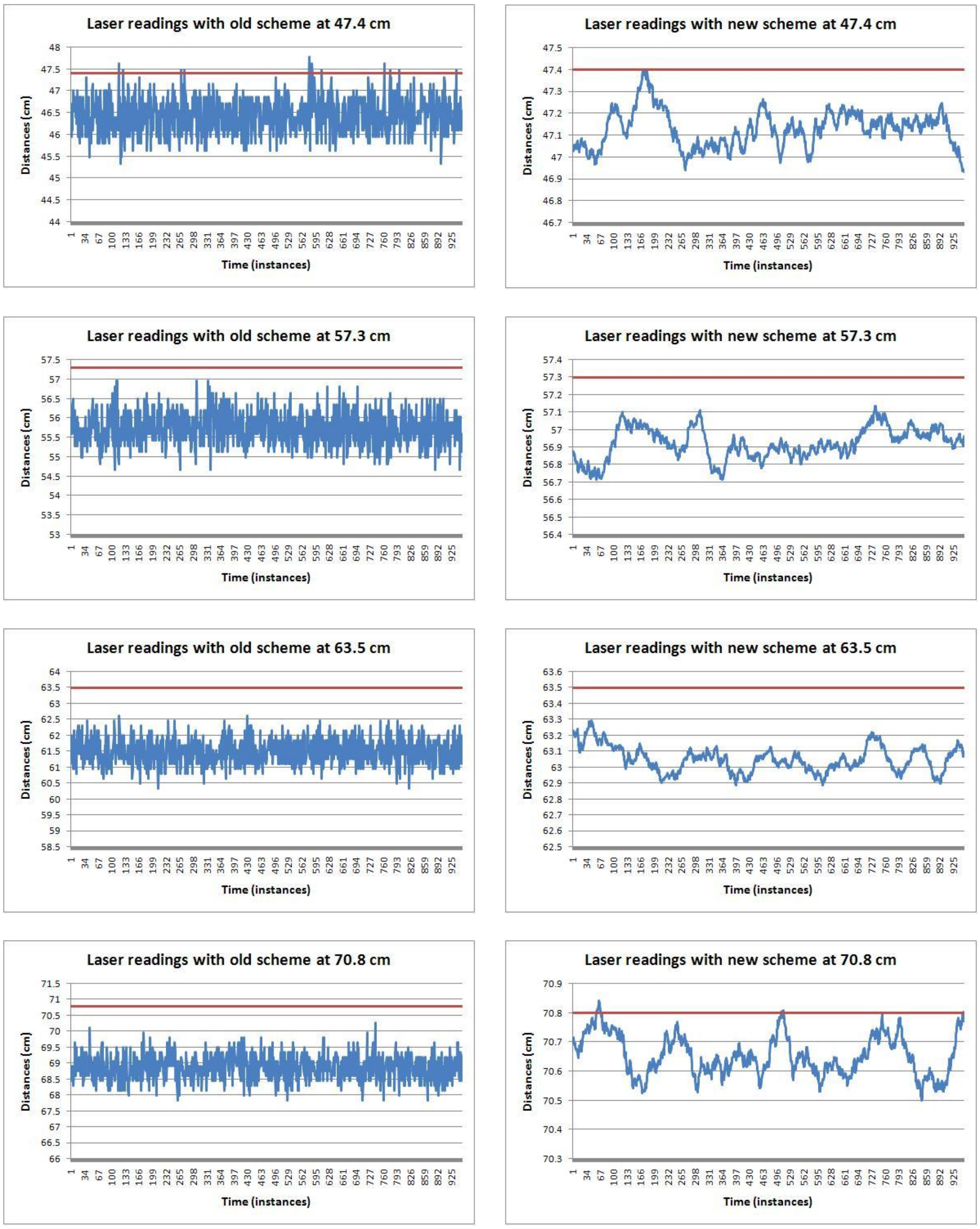
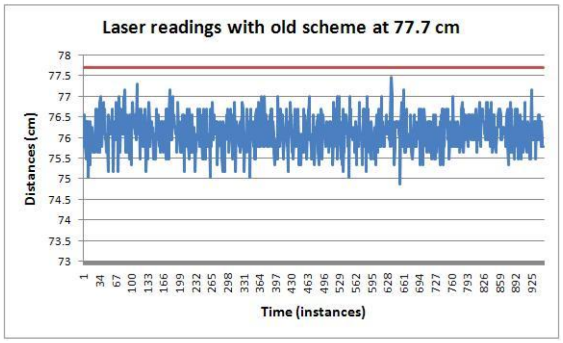


Figure 76 (Continued)



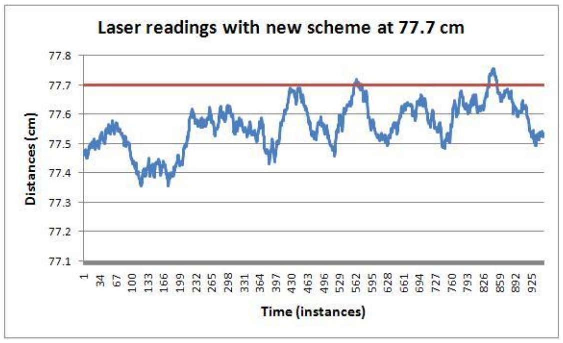


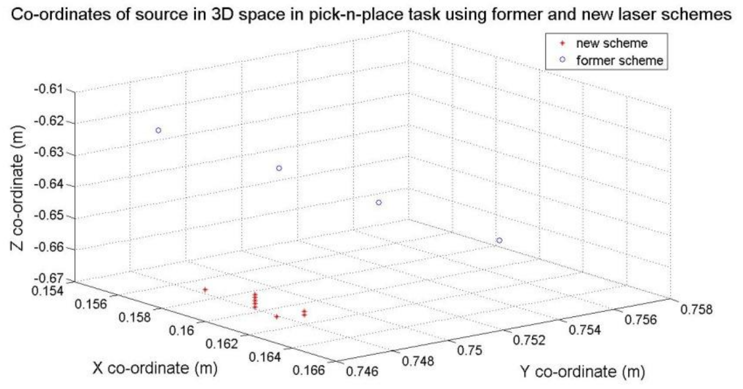
Figure 76 (Continued)

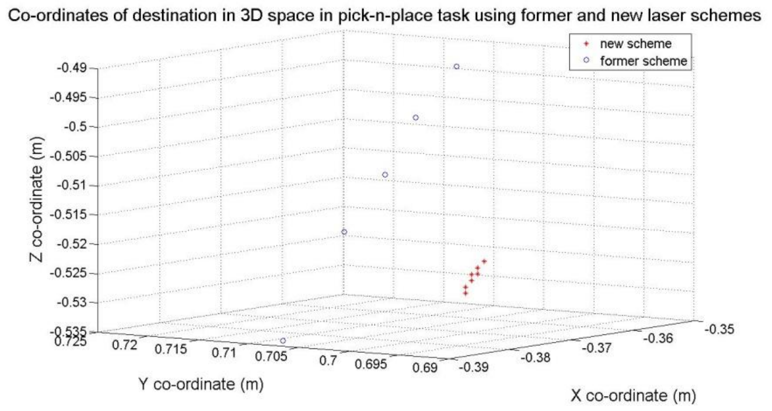
The accuracy and reproducibility of the distance values from the laser sensor using old and new schemes are summarized in the table below from the plots of figure 76.

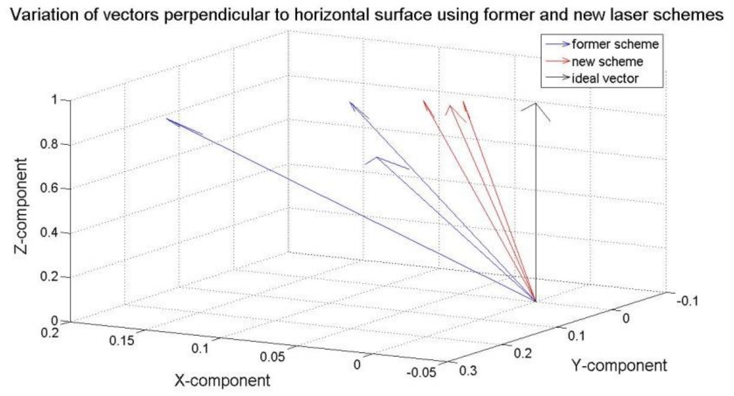
Table 9: Accuracy and reproducibility comparison using old and new laser schemes

Distance (cm)	Accuracy (cm)		Reproducibility (cm)	
	Old scheme	New scheme	Old scheme	New scheme
24.7	-1.6 and +1.0	-0.3 and +0.2	2.6	0.5
29.7	-2.3 and +0.1	-0.2 and +0.2	2.4	0.4
38.2	-1.8 and +0.4	-0.4 and 0	2.2	0.3
47.4	-2.1 and +0.3	-0.5 and 0	2.4	0.5
57.3	-2.7 and 0	-0.6 and 0	2.3	0.4
63.5	-3.2 and 0	-0.6 and 0	2.3	0.4
70.8	-3.0 and 0	-0.3 and 0	2.4	0.3
77.7	-2.9 and 0	-0.3 and +0.1	2.6	0.4

Thus from the table above we see that using the old scheme, we get an accuracy of -3.2 cm and +1.0 cm whereas using the new scheme we get an accuracy of -0.6 cm and +0.2 cm. The repeatability in the former and new schemes is 2.6 cm and 0.5 cm respectively. Thus we see that there is a significant improvement in the laser distance measurement capability. As the distance measured by the laser sensor forms the basis of the computations involving recording of points in the workspace for path planning, in generating virtual geometries and linear trajectories, the improvement in laser distance measurement capability directly improves the performance of the telerobotic task execution in the laser assisted mode. The overall accuracy and repeatability of the system improves. With this the points can be recorded with fewer errors which directly results in precise trajectory and path generation and execution and precise virtual geometry creation. This will enable the PUMA following describe a more accurate destination point and path and a more accurate surface for hand alignment in the pick-and-place task. These facts have been shown in the figures below. The figures below show recording of the points and generation of surfaces using the old and new laser schemes. As we can see from the figures, the same point recorded using the old scheme is dispersed more than that recorded using the new scheme. This means that the system will drop off the objects at a different point each time in the pick-and-place task or will record a surface tilted with respect to the ideal surface. The latter is shown in the figures below where the vectors (arrows) are the perpendiculars to the surface of interest and are computed using the laser data. This will result in an improper hand alignment with the surface.







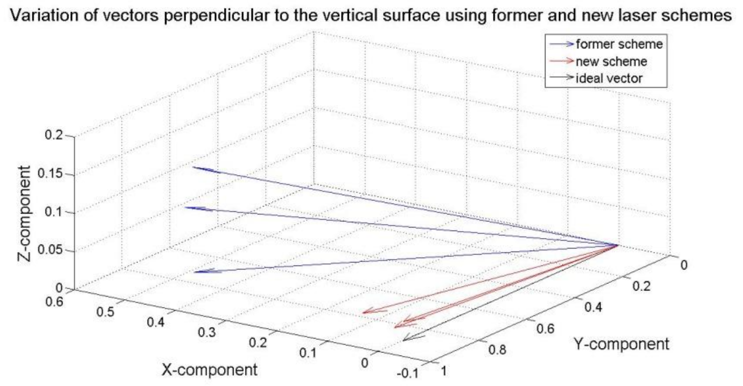


Figure 77: Impact of improved laser characteristics on telerobotic system performance

7.4. Results on Singularity Avoidance Implementation

In chapter 5 we had introduced the reader to the concept of singularity, gave the theory behind singularity avoidance using singularity-robust (SR) inverse of the Jacobian and had explained the implementation of the SR inverse of the Jacobian for singularity avoidance. At singular configurations the arm generates extremely high joint velocities for small movements in the 3D Cartesian space. Thus the movements are unstable and dangerous and so singular configurations are undesirable. SR inverse of the Jacobian prevents the arm from reaching singular configurations by inducing errors in joint movements. Thus the arm deviates away from the desired trajectory as it nears singular points. It continues to deviate in case of external singularities but gets back onto the desired trajectory in case of internal singularities.

Here we generate two trajectories from the laser data by pointing the laser to a target in simulation. The first trajectory has an external singularity and the second one has an internal as well as an external singularity. Both these trajectories are generated twice, once using the general inverse of the Jacobian and the second time using the SR inverse of the Jacobian. We compare the performance of the PUMA as it traverses these trajectories with general inverse and with SR inverse. We generate the following plots for the trajectories generated using general inverse:

1. Determinant of Jacobian: The determinant of the Jacobian gives an indication of when the robot enters into a singular configuration. We know that when the determinant of the Jacobian becomes zero, the robot has reached a singular configuration.
2. Joint angles: When the robot reaches a singular configuration, the joint velocities as well as the joint angles reach very high values.

For the plots trajectory generated using SR inverse following plots have been generated:

1. Manipulability: Manipulability is a measure of singularity. When the arm is at a singular point, the manipulability is zero and its value increases as the end effector goes away from singular point. The SR inverse method prevents the manipulability from taking the value of zero.
2. Joint angles: The joint rates as well as the joint angles remain in their limits and do not take unnecessarily high values when SR inverse method prevents the arm from reaching singular points along its trajectory.
3. Error in trajectory: This is represented by error in end effector co-ordinates and error in end effector angles. The SR inverse method of singularity avoidance avoids singularities by inducing error in robot motion.

The first trajectory was generated with the PUMA having joint angles [-1.44424 -2.85717 0.36547 -0.34528 -0.47725 -0.09663] at the initial point and the laser pointing to a target 0.35 m away from it. As we can see from the plots in figure 79, the determinant of the Jacobian hits the zero value at approximately 0.63 seconds. When this happens the joint velocities assume very high values and the PUMA goes out of control. From the plots we see that all the joint angles except joint 1 assume very high values (of the order of 1000 degree). The PUMA motion is dangerous this way. On the other hand, with SR inverse of the Jacobian, the manipulability starts taking values closer to zero as the PUMA approaches the singular point along its trajectory but it never becomes zero so that the singular point is never reached. Also the change in the values of manipulability from being same as the determinant of the Jacobian away from the singular point to being close to zero as PUMA approaches the singular point is gradual and smooth. As a result no jerks are obtained in the PUMA motion. This was because the values of the scale factor at

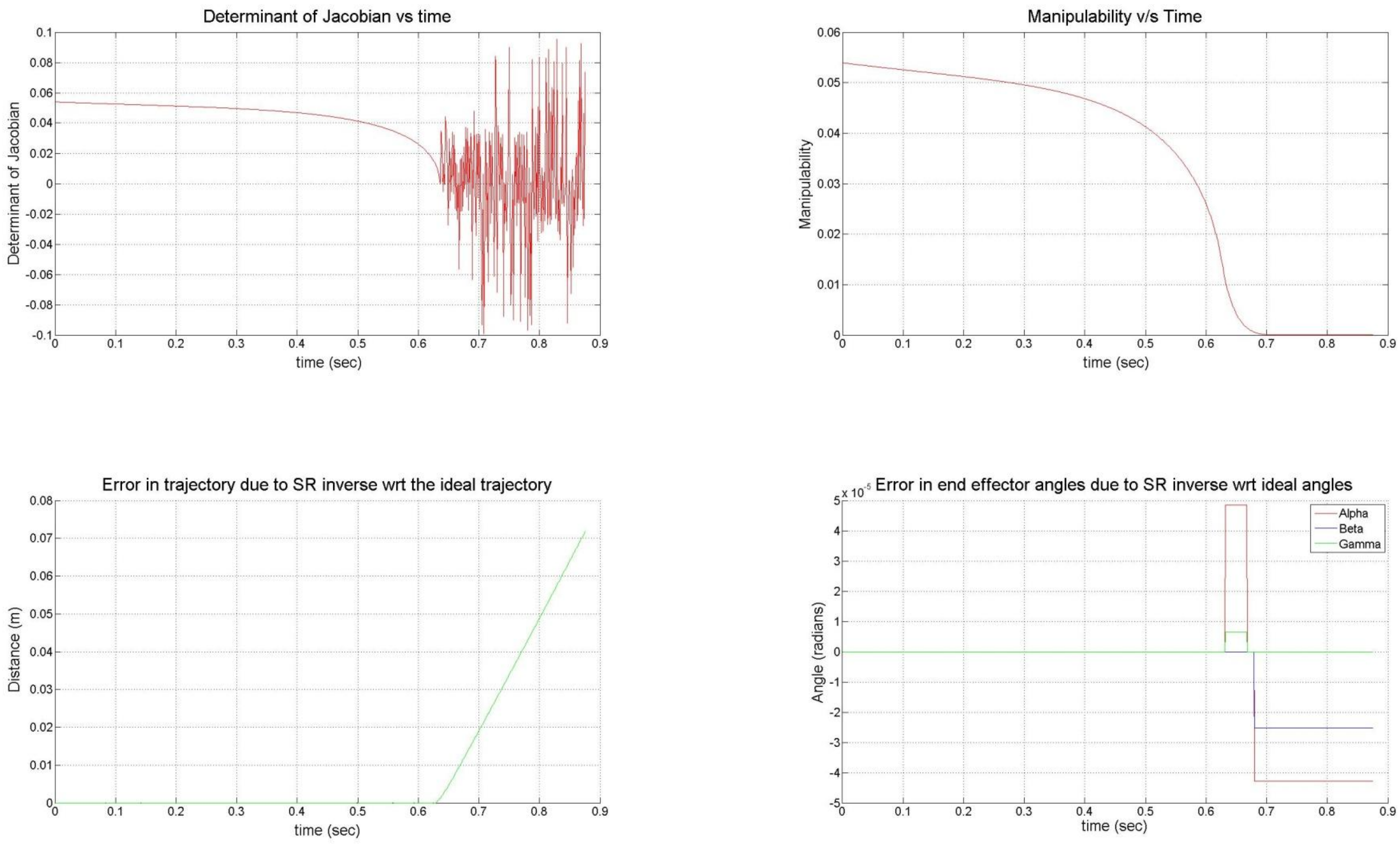


Figure 78: Plots comparing trajectory 1 with and without singularity avoidance

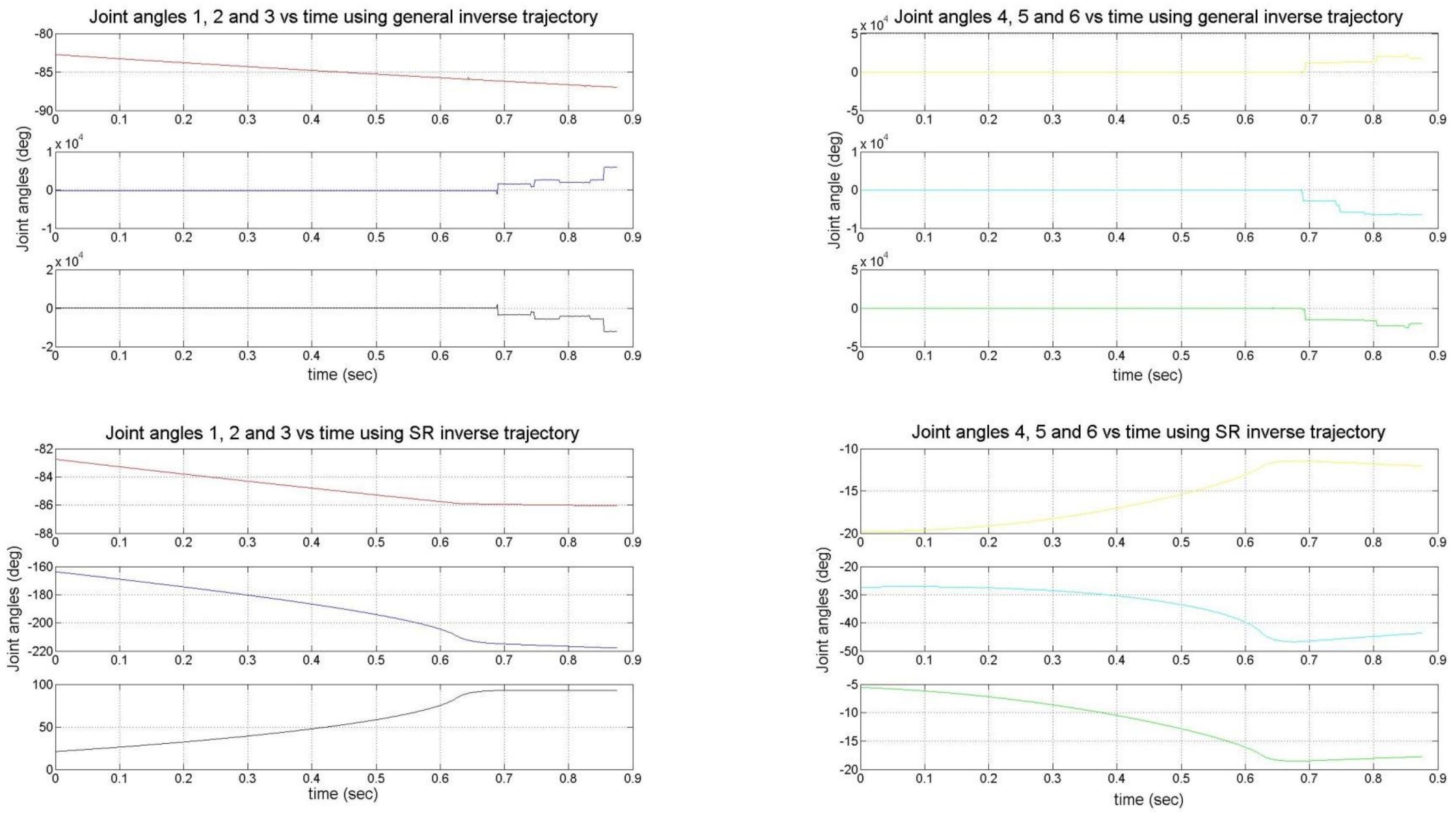


Figure 79 (Continued)

singularity, k_0 and that of manipulability, w_0 at the neighborhood of singular point were determined optimally as explained in chapter 5. We can also see that the joint angles are in their acceptable range. Thus the singular point is avoided but the error that is introduced in the trajectory is seen form the plots. From the plot we see that the error goes on increasing until the robot stops. This is because this is the case of an external singularity. Here the arm tries to stretch itself out of its workspace. But due to singularity avoidance instead of trying to stretch itself more than it can it starts following a trajectory with it fully stretched along its outer boundary. Therefore the error constantly goes on increasing.

The second trajectory is generated with the initial joint angles of the PUMA at [-1.37229 -2.27639 -0.01268 0.01852 -1.53292 0.27288] and the laser pointing to a target at 1.3 m. This trajectory has both external and internal singularities. As we see from the plot of the determinant of the Jacobian versus time in figure 80, the determinant hits zero at about 2.1 seconds and the PUMA goes out of control. This is when the PUMA hits an internal singularity. Then at approximately 2.8 seconds the PUMA is back on its original trajectory and the singular point is past. But then the PUMA hits the outer boundary of its workspace which is the external singularity and the Jacobian again becomes zero. The PUMA motion becomes unstable again. All the joints assume very high joint angle values. On the other hand with the trajectory generated using SR inverse for singularity avoidance, the manipulability never becomes zero but comes close to it when the PUMA is near a singular point. The error in the trajectory increases and then it decreases. The error in the trajectory becomes zero at around 2.7 seconds. This is when the internal singularity is passed and the robot is back on its original trajectory. Then again at around 2.8 seconds, when the robot has reached the end of its workspace the error goes on increasing. This is external singularity.

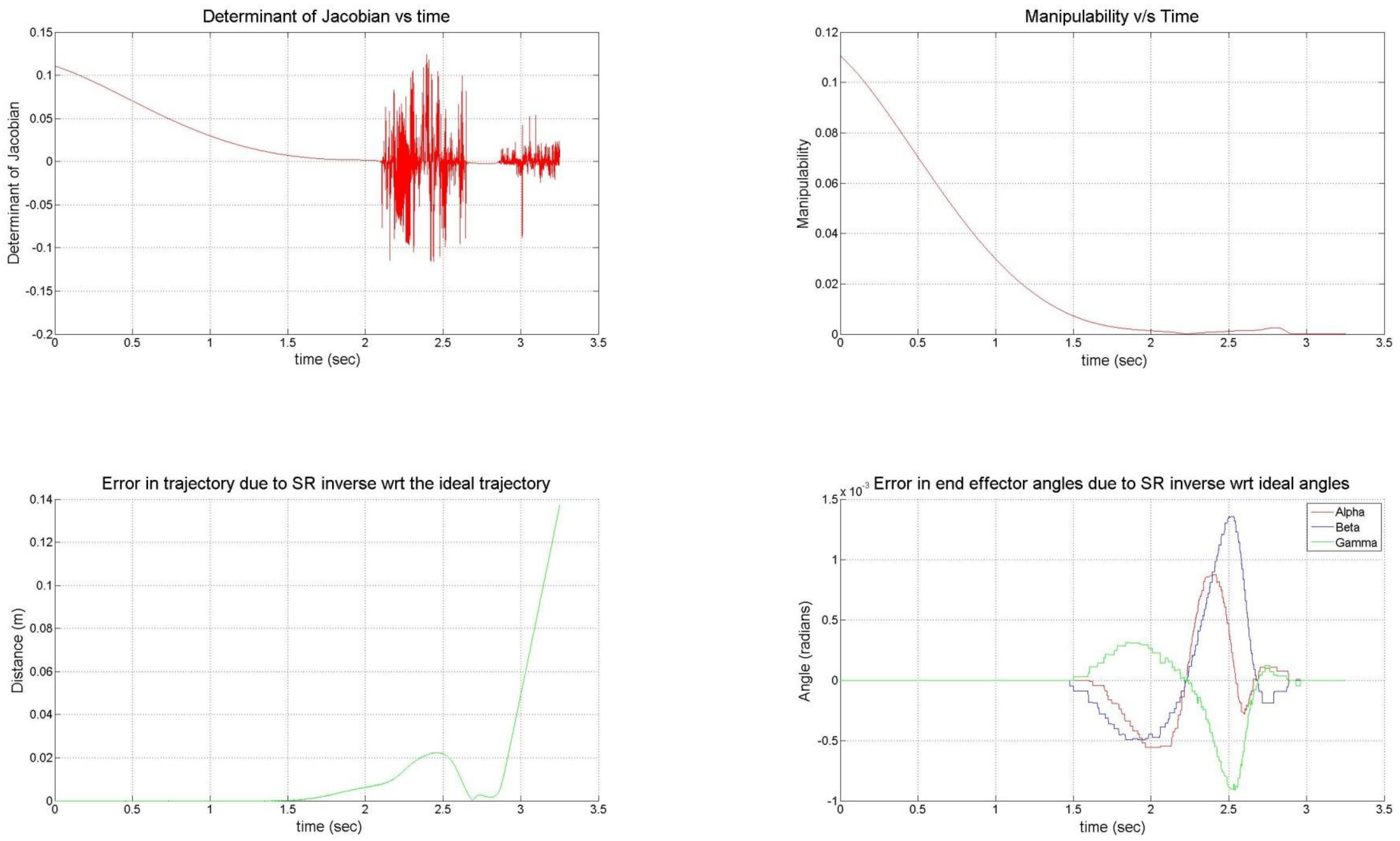


Figure 79: Plots comparing trajectory 2 with and without singularity avoidance

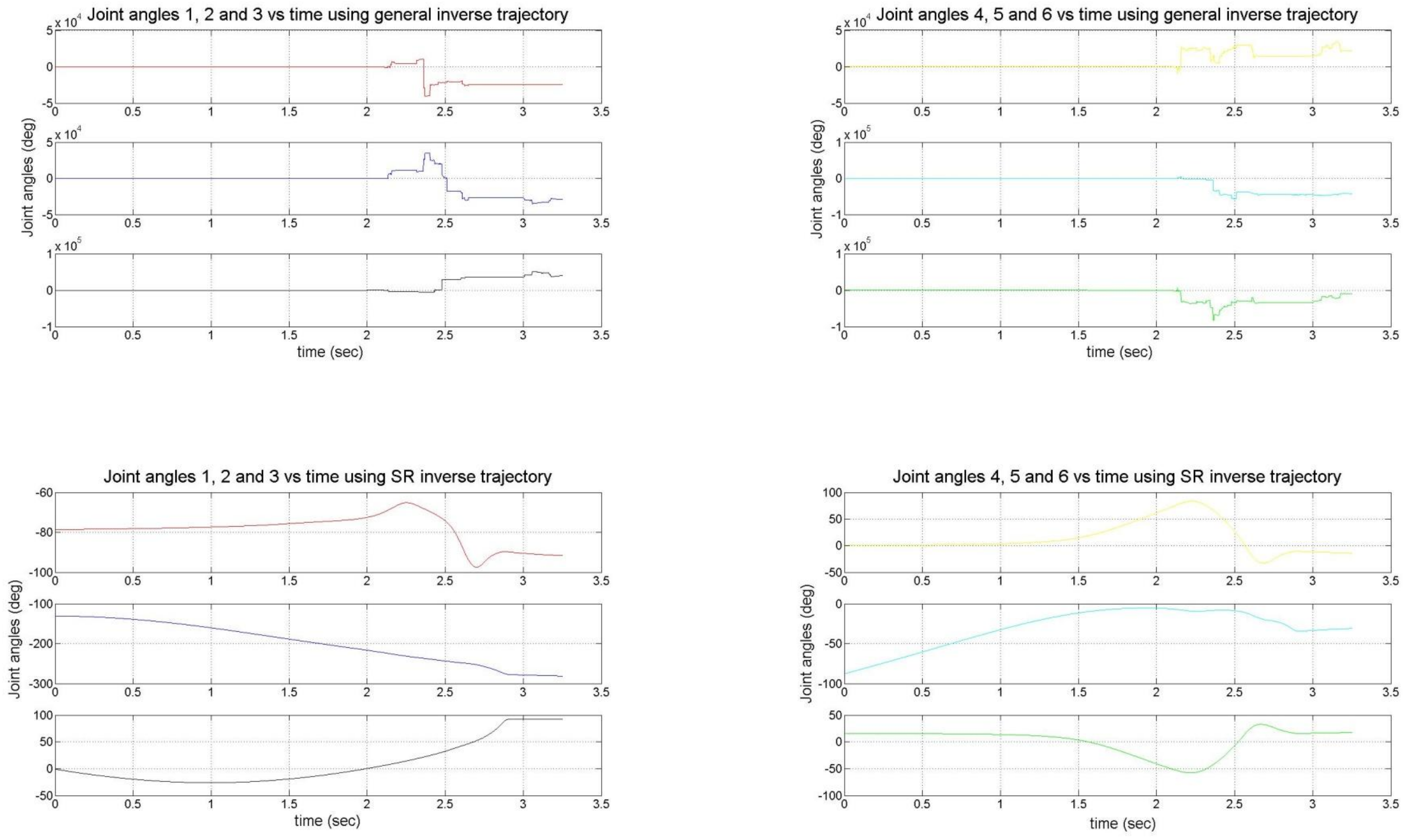


Figure 80 (Continued)

Chapter 8: Conclusions

In this chapter we will make concluding remarks for the thesis project and describe the scope of future work.

8.1. Concluding Remarks

The project started about a year ago with the basic telerobotic system in place. The basic system included forward and inverse kinematic algorithms, trajectory generation algorithms, communication between the PCs controlling the different hardware like master robot, slave robot, and laser sensor. Due to improper working of the telerobotic system, it required validation of these basic algorithms. The project and author's contribution took off from this point. The novelty of the project, that of using the laser information effectively to assist the user executing the task in the remote unstructured environment originally came from Dr. Rajiv Dubey, the author's advisor, and was later on improved and built up upon by the author, initially with assistance from the former PhD student Dr. Eduardo Veras and later on his own. The major challenges in this project were in managing its complexities due to the heterogeneous nature of the telerobotic system and in dealing with the system breakdowns that called for troubleshooting skills.

The telerobotic system has various software like GNU C++ on QNX, MS VC++ on Windows, OpenGL library, and OpenHaptics library that made the system complex from management point of view. Additionally Matlab was used for simulations, sensor calibration and data analysis and data generation by porting data to it from the telerobotic system. Working on a number of programming languages at a time is a challenge that the author faced. System breakdowns due to robot calibration and robot system file's errors (problem of PUMA deviation from its path due to wrong initialization of joint 3), communication problem with wrong data flow from Omni to PUMA (jerking PUMA due to its sudden engagement in teleoperation and the buffer being empty between engagements), system shutdown due to improper dynamic memory handling (QNX machine crash due to allocating memory to a pointer not initialized with maximum memory it can hold), concurrency problem in dealing with different devices with different update rates (high speed serial port had to be read from a separate thread as the OpenGL which ran at 30 Hz refresh rate was not synchronized) and communication problem in data getting corrupted when sending it from Omni PC to QNX machine (data structure size cut down). This list is not exhaustive but these were the some of the main challenges that the author faced and was able to meet them successfully.

The main objective of the project was to utilize the laser data innovatively along with machine intelligence and high level human input to assist the user in executing manipulation tasks in remote unstructured environments through a telerobotic system. The assistance has been demonstrated in the form of a Traded Control formulation where the human makes high level decisions, commands the robot and acts as a supervisor while the machine carries out computationally intensive and precision tasks. The mission was to implement the above without extensive computational and algorithmic resources which includes extensive 3D environment models, without the use of sophisticated sensors and without using slow methods like Teach and Play. The stress was also on creating a very simple and quick to activate interface which has been achieved. The user simply teleoperates the wrist of the PUMA with the Omni, points the laser to critical points in the workspace and presses certain keys on the keyboard. The machine with this information and that from the PUMA kinematics generates the online trajectories, virtual constraints, virtual geometries or path plans, depending on the task at hand, that assist the user in executing the task. The user is then a mere supervisor and only gives high level commands. The machine does the low level activities.

Although accuracy and time results have been gathered and presented and they suggest that improvement in the task performance from speed and precision point of view, the easing of the mental load on the user that does the task with the robot has also been achieved. This has been achieved based on the feedback that the author received from the users who were the subjects in the testing of the laser based telerobotic system. From the responses obtained from the subjects it was clear that they overwhelmingly were in favor of using the laser assisted mode in executing the task. According to them, picking up the points with the laser with minimum teleoperation and not being concerned about the complexities of the task which is handles by the machine is fantastic. Activities like orienting the hand properly for grasping an object or traversing along a trajectory while following the shortest path and avoiding obstacles puts a lot of mental load on the user. With the autonomous control of these functions the user need not worry about anything.

Though there are studies that have proved that Traded Control of this form is preferred than solely teleoperation control mode, the significance of this project is that this has been made possible in an unstructured environment, with online generation of basic environment information and task plans with minimum computations and using basic range data from the laser sensor innovatively. Moreover, the interface is very simple and quick to activate.

The effort towards the end of the project in improving laser data acquisition and processing characteristics though do not drastically improve the system performance in task execution as the human is always there in the loop and can make adjustments when the sensor data is inaccurate, it is still helpful to get more accurate range data. Also, this would be helpful in the long run as the system evolves further.

8.2. Future Work

The next step would be to test the capability of the telerobotic system in handling tasks in which the robot makes extensive contacts with the environment such as the door opening task, shelf opening task, assembly tasks like inserting peg in a hole and so on. Compliance in the arm control needs to be incorporated by means of a force sensor for this purpose. The force sensor information would also be used to provide force feedback to the user via the haptic master, the Omni. The haptic capabilities of the Omni also are needed to be explored.

Issues like mechanical maintenance of the arm, arm repair, aging of the arm need to be taken care of as an immediate step. Recalibration of the PUMA and implementing joint limit avoidance is also on immediate lines. Other features like combined joint control and Cartesian control in teleoperation for easier control of the arm must be experimented with. Mapping the Omni motion onto the PUMA in such a way that the users hand motion is directly mapped to the PUMA instead of the Omni motion being mapped to the PUMA is something that the author is looking at experimenting for easier control.

Some improvements to the system can spring out additional projects. Creating a rich and smart user interface that immerses the user in the remote environment needs to be developed. The current graphical interface can be done away with. Instead buttons on an interface and status messages can be used to engage and disengage the PUMA with the Omni. Stereo vision still cameras need to be mounted in the remote environment to give at least two views of the remote environment. This video feedback from the cameras should be fused with the user interface. The capability of forward simulation with augmented reality is also desirable with the interface. GUI libraries running on C++ would be desirable for creating the user interface to maintain uniformity in programming language.

Another project would be implementing and testing the hard real-time capabilities of the system by prioritizing task execution. The system has priorities assigned to various threads and these threads do exchange data amongst themselves. However they are independent as far as any conflicts with either data or execution priority are concerned. Various task priorities should be assigned to tasks and testing must be carried out to validate the hard real-time capabilities of the system.

Reformatting the code so as to give the code an object oriented structure has the capability to be implemented as a project. The capabilities of object oriented programming (OOP) like data encapsulation, combining data and associated member functions into objects entity, polymorphism, inheritance etc. can be exploited to make a more manageable software system.

The long term goal of the author is to infuse artificial intelligence (AI) in the telerobotic system. Combining human decisions with machine decisions for task execution, the machine decisions being based on artificial learning and planning is an area that can be implemented on the current system. The idea would be on the lines of human-machine interaction where there would be real-time weighing of the decisions from the human and the machine and those that lead to optimum task execution considering the various parameters like obstacle avoidance, shortest path, sensor data accuracy, assistance to human and so on would be implemented. The learning from each experience would be used to make the future decisions. Concrete implementation plans are yet to be finalized.

List of References

[1] Sheridan, T.B., 1992, "Telerobotics, automation, and human supervisory control," MIT Press, Cambridge, Mass., pp. 393.

[2] Hayati, S., and Venkataraman, S., 1989, "Design and Implementation of a Robot Control System with Traded and Shared Control Capability," IEEE International Conference on Robotics and Automation, Anonymous IEEE, USA, pp. 1310-1315.

[3] Backes, P., and Tso, K., 1990, "UMI: An Interactive Supervisory and Shared Control System for Telerobotics," IEEE International Conference on Robotics and Automation, Anonymous IEEE, USA, pp. 1096-1101.

[4] Yokokohji, Y., Ogawa, A., Hasunuma, H., 1993, "Operation modes for cooperating with autonomous functions in intelligent teleoperation systems," IEEE International Conference on Robotics and Automation, Anonymous IEEE, USA, **3,** pp. 510-515.

[5] Tarn, T., Xi, N., Guo, C., 1996, "Task-Oriented Human and Machine Co-Operation in Telerobotic Systems," Annual Reviews in Control, **20**pp. 173-178.

[6] Hirai, S., Sato, T., and Matsui, T., 1990, "Intelligent and Cooperative Control of Telerobot Tasks," IEEE Conference on Decision and Control, Anonymous IEEE, **1,** pp. 5-7.

[7] Conway, L., Volz, R., and Walker, M., 1987, "Tele-autonomous systems: Methods and architectures for intermingling autonomous and telerobotic technology," IEEE International Conference on Robotics and Automation, Anonymous IEEE, **4,** pp. 1121-1130.

[8] Anderson, R., 1996, "Autonomous, Teleoperated, and Shared Control of Robot Systems," IEEE International Conference on Robotics and Automation, Anonymous IEEE, **3,** pp. 22-28.

[9] Hamel, W., Zhang, G., and Murray, P., 2002, "A Real Time Controller for Human-Machine Cooperative Telerobotics," IEEE International Conference on Robotics and Automation, Anonymous IEEE, **3,** pp. 2880-2885.

[10] Everett, S., and Dubey, R., 1998, "Human-machine cooperative telerobotics using uncertain sensor or model data," IEEE International Conference on Robotics and Automation, Anonymous IEEE, **2,** pp. 1615-1622.

[11] Hasegawa, T., Suehiro, T., and Takase, K., 1991, "A Robot System for Unstructured Environments Based on an Environment Model and Manipulation Skills," IEEE International Conference on Robotics and Automation, Anonymous IEEE, **1,** pp. 916-923.

[12] Hager, G., Okamura, A., Kazanzides, P., 2008, "Surgical and Interventional Robotics: Part III," IEEE Robotics and Automation Magazine, **15**(4) pp. 84-93.

[13] Fischer, P., Daniel, R., and Siva, K. V., 1990, "Specification and Design of Input Devices for Teleoperation," IEEE International Conference on Robotics and Automation, Anonymous Cincinnati, OH, USA, **1,** pp. 540-545.

[14] Veras, E., 2008, "A Hard Real-Time Telerobotic Control System using Uncertain Sensory Data Fusion and Sensor-Based Assistive Functions", University of South Florida, USA.

[15] Joly, L., and Andriot, C., 1995, "Motion Constraints to a Force Reflecting Telerobot through Real-Time Simulation of a Virtual Mechanism," IEEE International Conference on Robotics and Automation, Anonymous IEEE, **1,** pp. 357-362.

[16] Aigner, P., and McCarragher, B., 1997, "Human Integration into Robot Control utilizing Potential Fields," IEEE International Conference on Robotics and Automation, Anonymous IEEE, **1,** pp. 291-296.

[17] Tsai, C., Lin, H., and Lin, C., 2004, "Trajectory Tracking Control of a Laser-Guided Wheeled Mobile Robot," Trajectory tracking control of a laser-guided wheeled mobile robot, Anonymous **2,** pp. 1055-1060.

[18] Rizun, P., and Sutherland, G., 2005, "Tactile Feedback Laser System with Applications to Robotic Surgery," First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, Anonymous IEEE, **1,** pp. 426-431.

[19] Surmann, H., Nüchter, A., and Hertzberg, J., 2003, "An Autonomous Mobile Robot with a 3D Laser Range Finder for 3D Exploration and Digitalization of Indoor Environments," Robotics and Autonomous Systems, **45**(3-4) pp. 181-198.

[20] Takahashi, Y., and Yashige, M., 2000, "Robotic manipulator operated by human interface with positioning control using laser pointer," IEEE 26th Annual Conference of the Industrial Electronics Society, Anonymous IEEE, **1,** pp. 608-613.

[21] Cheung, E., and Lumelsky, V., 1990, "Motion planning for a whole-sensitive robot arm manipulator," IEEE International Conference on Robotics and Automation, Anonymous IEEE, **1,** pp. 344-349.

[22] Nayak, N., and Ray, A., 1990, "An integrated system for intelligent seam tracking in robotic welding: Part I - Conceptual and analytical development, Part II - Design and implementation," IEEE International Conference on Robotics and Automation, Anonymous IEEE, USA, pp. 1892-1903.

[23] Sensable Technologies, 2009, "Http://www.Sensable.com/products-Openhaptics-Toolkit.Htm," **2009**(June/14) .

[24] Craig, J., 2005, "Introduction to Robotics: Mechanics and Control," Prentice Hall, USA, pp. 408.

[25] Unimation, 1987, "Unimate PUMA Mark III Robot, 500 Series, Models 552/562 Equipment Manual 398AH1," .

[26] Fu, K., Gonzalez, R., and Lee, C., 1987, "Robotics: control, sensing, vision, and intelligence," Mcgraw-Hill Book Company, USA, pp. 580.

[27] Whitney, D., 1969, "Resolved Motion Rate Control of Manipulators and Human Prostheses," IEEE Transactions on Man Machine Systems, **10**(2) pp. 1536-1540.

[28] Yoshikawa, T., 1990, "Foundations of Robotics: Analysis and Control," THe MIT Press, Japan, pp. 285.

[29] Nakamura, Y., 1991, "Advanced Robotics: Redundancy and Optimization," Addison-Wesley, USA, pp. 337.

Appendices

Appendix A: Phantom Omni Specifications

Table 10: Phantom Omni haptic master technical specifications

Force feedback workspace	~6.4 W x 4.8 H x 2.8 D in. > 160 W x 120 H x 70 D mm.
Footprint (Physical area device base occupies on desk)	6 5/8 W x 8 D in. ~168 W x 203 D mm.
Weight (device only)	3 lbs. 15 oz.
Range of motion	Hand movement pivoting at wrist
Nominal position resolution	> 450 dpi. ~ 0.055 mm.
Backdrive friction	< 1 oz (0.26 N)
Maximum exertable force at nominal (orthogonal arms)position	0.75 lbf. (3.3 N)
Continuous exertable force (24 hrs.)	> 0.2 lbf. (0.88 N)
Stiffness	X axis > 7.3 lbs. / in. (1.26 N / mm.) Y axis > 13.4 lbs. / in. (2.31 N / mm.) Z axis > 5.9 lbs. / in. (1.02 N / mm.)
Inertia (apparent mass at tip)	~0.101 lbm. (45 g)
Force feedback	x, y, z
Position sensing [Stylus gimbal]	x, y, z (digital encoders) [Pitch, roll, yaw (± 5% linearity potentiometers)
Interface	IEEE-1394 FireWire® port: 6-pin to 6-pin
Supported platforms	Intel or AMD-based PCs
OpenHaptics® Toolkit compatibility	Yes
Applications	Selected Types of Haptic Research, FreeForm® Modeling™ system, ClayTools™ system

Appendix B: Laser Sensor Specifications

Table 11: DT60 laser sensor technical specifications

Measurement range min ... max:	200 mm ... 5,300 mm
Type of light:	Laser, red light
Laser protection class:	2 (EN 60 825-1)
Note:	Invertable
Remark:	After 30 minutes on-time
Dimensions (W x H x D):	38 mm x 104 mm x 87 mm
Supply voltage min ... max:	DC 11 ... 30 V
Light spot diameter:	Ø 10 mm at 2 m distance
Resolution:	1,5 mm
Accuracy:	+- 10 mm
Reproducibility:	+- 8 mm typ.
Response time:	50 ms ... 250 ms
Switching outputs:	PNP, Q
Analogue output:	Yes
Analogue output min ... max:	4 mA ... 20 mA
Ripple:	<= 5 Vss
Connection type:	Connector, M12, 5-pin
Enclosure rating:	IP 67
Ambient operation temperature, min ... max:	-25 °C ... +55 °C
Ambient storage temperature, min ... max:	-25 °C ... +75 °C
Housing material:	Plastic