2011

Design and Development of a Tele-operated Surgical Simulation Environment

Jeremy Chad Breetzke

Design and Development of a Tele-operated Surgical Simulation Environment

(Spine Title: Design of a Tele-operated Surgical Simulation Environment)
(Thesis Format: Monograph)

by

Jeremy C. Breetzke

Faculty of Engineering Science
Department of Mechanical and Materials Engineering

2

Submitted in partial fulfillment
of the requirements for the degree of
Master of Engineering Science

School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Certificate of Examination

THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF GRADUATE STUDIES
CERTIFICATE OF EXAMINATION

Chief Adviser(s):	**Examining Board:**
Dr. Michael Naish	Dr. Samuel Asokanthan
Dr. Rajni Patel	Dr. Ralph Buchal
Advisory Committee:	Dr. Robert Sobot

The thesis by
Jeremy Breetzke
entitled:
Design and Development of a Tele-operated Surgical Simulation Environment
is accepted in partial fulfillment of the
requirements for the degree of
Master of Engineering Science

Date: ____________

Chair of Examining Board
Dr. George Knopf (Chair)

Design and Development of a Tele-operated Surgical Simulation Environment

Jeremy C. Breetzke
M.E.Sc. Thesis, 2011
Department of Mechanical and Material Engineering
University of Western Ontario

Abstract

With the introduction of robots into laparoscopic surgery, surgeons have difficulties in selecting the placement of the incisions required to insert the robots instruments into the body and also determine which patients are suitable for robotically assisted surgery. Poor selection of these two items mentioned above can result in a conversion to a more invasive form of surgery during the procedure. This work introduces the design and development of a surgical simulation environment to assist in the research for optimal incision placement and patient selection.

The simulator allows importing any serial link robot that was designed in a computer aided modelling package. With minimal added information, the imported robot can be controlled using a multi-degree of freedom user input device. The simulator allows for importing patient geometries along with the robot to allow for the simulation of surgical procedures. A Jacobian transpose algorithm was added onto the simulator in a modular format to control the simulated robots, as well as to allow for other control systems to be created and implemented.

Experiments were performed to determine the effects of patient geometry models on rendering speeds. The control system could control the tested robots with a maximum lag time of 15 ms between moving the input device and the simulated robot moving to the correct desired position.

The simulator makes importing and controlling robots a simple and intuitive matter, without putting a large restriction on the type of robots to be simulated. The simulator also allows for importing models of a patient, to make real world analysis of a patient possible. Further improvements on the presented simulator include the addition of collision detection and more testing on the control system for stability and response over a larger range of robots.

Keywords: Medical robotics simulator, da Vinci medical robot, Kinematic control, VTK rendering libraries.

Contents

List of Figures

List of Tables

Nomenclature and Acronyms

Latin Letters

$\dot{b}$	Translational rate of a prismatic joint
e	Subscript used to define an error measure
$\mathbf{e}$	Joint unit vector
${}^{j}_{i}\mathbf{J}$	Jacobian matrix of frame i defined in frame j. If j is omitted then it is defined in the world frame. If i is omitted then it is the Jacobian of the end effector frame.
${}^{j}\mathbf{P}_{i}$	Vector pointing from origin of frame i to frame j. If j is omitted then the vector is pointing from the world frame origin.
${}^{i}_{i+1}\mathbf{R}$	Matrix that rotates a vector in frame $i+1$ to frame i
$\mathbf{R}_{\mathrm{d}}$	Desired rotation matrix
$\mathbf{R}_{\mathrm{e}}$	Rotation matrix that is used as the orientation error measure between two frames
$\mathbf{R}_{\mathrm{input}}$	The rotation matrix obtained from the user input
$\mathbf{R}_{\mathrm{init}}$	The initial rotation matrix of the end effector
${}^{i}_{i+1}\mathbf{T}$	Matrix that transforms a vector in frame $i+1$ to frame i
$\mathbf{x}_{\mathrm{d}}$	The desired position of the end effector
$\mathbf{x}_{\mathrm{init}}$	The initial position of the end effector
$\dot{\mathbf{y}}$	First time derivative of a vector $\mathbf{y}$
W	Subscript or superscript to define the world coordinate frame

Greek Letters

ϵ	Vector part of quaternion
η	Scalar value of quaternion
θ	Joint angle
$\dot{\theta}$	Joint velocity
$\mathcal{Q}$	Unit quaternion
$\boldsymbol{\tau}$	Joint torque vector
ϕ_e	Orientation error vector
χ_e	Position error vector
${}^j\omega_i$	Angular velocity of frame i defined in frame j. If j is omitted then the velocity is defined in the world frame.

Acronyms

2-D	Two-Dimensional
3-D	Three-Dimensional
CABG	Coronary Artery Bypass Graft surgery
CAD	Computer Aided Design
CT	Computed Tomography
D-H	Denavit-Hartenberg parameters
DOF	Degree(s) Of Freedom
EE	End Effector
GPU	Graphics Processing Unit
GUI	Graphic User Interface
LAD	Left Anterior Descending artery

LIMA	Left Internal Mammary Artery
MIS	Minimally Invasive Surgery
MRI	Magnetic Resonance Imaging
OCS	Open Chest Surgery
RAM	Random Access Memory
RAMIS	Robotically-Assisted Minimally Invasive Surgery
RAMICS	Robotically-Assisted Minimally Invasive Cardiac Surgery
RCM	Remote Centre of Motion
STL	STereo Lithography file format
VTK	Visualization ToolKit libraries
XML	eXtensible Markup Language

Chapter 1

Introduction

This thesis presents work towards the development of a teleoperated surgical simulation environment, to assist research in preoperative planning techniques. The simulated environment can be used to simulate any serial link robot, modelled in a CAD package, that is not mounted on a mobile base. The robot can have closed-loop parallelogram linkages in the kinematic chain, such as the linkages found on the da Vinci surgical robot.

1.1 Motivation

The direction of the development of medical technology is towards safer and less invasive techniques [1], which has led to the development of procedures such as laparoscopic surgery. This type of surgery is among those that are commonly referred to as Minimally Invasive Surgery (MIS). Laparoscopic surgery is the performance of general surgery with the use of specialized instruments. These instruments are generally long thin rigid shafts that have tools, such as grippers or scalpels, located on one end and surgeon's control handles on the other. A picture of a common laparoscopic instrument can be seen in Fig. 1.1. The laparoscopic instruments are inserted into the patient through trocars.

A trocar is a medical instrument with a hollow cylinder and a seal. The trocar is inserted into a patient chest through small incisions to seal the insufflated abdomen and provide a portal, or port, for laparoscopic tools to enter the chest cavity to perform surgical procedures.

The first laparoscopic surgery was performed in 1987; the procedure was a cholecystectomy, the surgical removal of the gall bladder [2]. MIS offers an alternative to open surgery where the surgeon must make openings large enough to fit his/her hands inside the patient to perform the procedure.

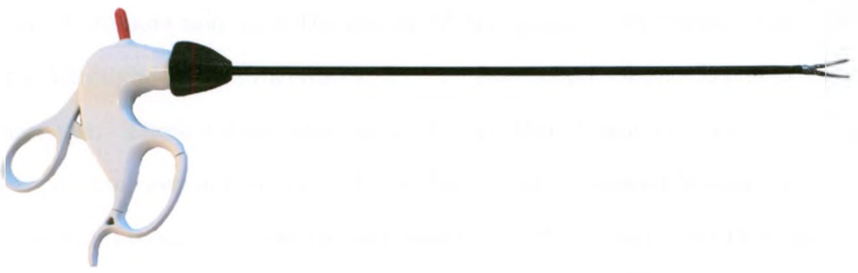


Figure 1.1: Laparoscopic grasper, as an example of a typical laparoscopic instrument.

MIS reduces the recovery time and trauma for patients undergoing surgery [2, 3]. Along with all of the successes of MIS, it has introduced new challenges to the surgeon [2, 4]:

- The laparoscopic instruments operating inside the patient are constrained by the ports, which reduces the number of degrees of freedom (DOF) at the tool end from 6 DOF to 4 DOF and makes procedures such as suturing difficult.
- The only visual feedback available to the surgeon during MIS is provided by an image from an endoscopic camera that is displayed on a two-dimensional video monitor. This removes depth perception and makes it challenging for the surgeon to successfully place the tool end of the instrument at the desired position. Only having a camera for visual feedback can lead to other problems such as a limited field of view, fog and debris collecting on the lens and so on.
- The haptic feedback of the laparoscopic tool is hampered by the friction generated between the rubber seal in the trocar and the laparoscopic instrument [5].
- The instrument's tool end moves in a nonintuitive manner that is reversed motion from the surgeons hand motion [2].

To overcome the challenges of performing MIS, robots were developed to assist the surgeon in manoeuvring the laparoscopic tools. Newer versions of these robots introduced an increase in dexterity by designing additional degrees of freedom into the wrist of the tools and stereoscopic cameras with a display that gives the surgeon depth perception [4]. These robots also reduce the surgeon's hand tremors and with the design of ergonomic workstations they also reduce the surgeon's fatigue throughout the procedures [2]. The endoscopic camera has greater magnification than the human eye. These robots also afford better instrument control by providing motion scaling and natural (nonreversed) motion. Thus, Robotically Assisted Minimally Invasive Surgery (RAMIS) has made minimally invasive surgery easier to perform and brought laparoscopic surgery to procedures that were previously difficult to perform laparoscopically.

The technology in RAMIS is relatively new and it is being used for long standing procedures that were not designed to make optimal use of the robots. These robotic systems are expensive to obtain and maintain, they are bulky and take up a large amount of space in the operating theatre [2]. Surgeons face a long and steep learning curve to be able to master the use of the robots [6]. A surgeon is firstly trained in the field in which he would perform RAMIS and thereafter trained to perform the procedures with the assistance of the robot. This adds extra time and cost for the surgeon to become qualified.

The robots developed for RAMIS use the same type of ports as MIS would. For these ports to be effective in reducing trauma to a patient, the robots cannot place any loads on the ports. Therefore, surgical robots were designed in such a way that there is a point on the tool shaft that is constrained from any motion except pivoting in the port and moving in and out of the port. There are two mechanical techniques to achieve this constrained point. One is to have a robot with redundant links and the other is to have a kinematic chain with a remote centre of motion (RCM) [4].

- Creating a constrained point using redundant links is achieved by adding two passive revolute joints to the end of a kinematic chain in the configuration shown in Fig. 1.2(a). A passive joint is defined as a joint that is not actuated by a servo mechanism, and is allowed to rotate freely without any form of constraint. By using this method, the long thin shaft of the surgical instrument will form the axis on which the constrained point lies.

- The second option of creating a remote centre of motion is shown in Fig. 1.2(b). The long thin shaft of the surgical instrument will form the link that has the RCM on it. Then as the first link in the chain is rotated, as shown in the figure, the shaft of the surgical instrument will pivot about the RCM. This method is described in more detail in Appendix B.

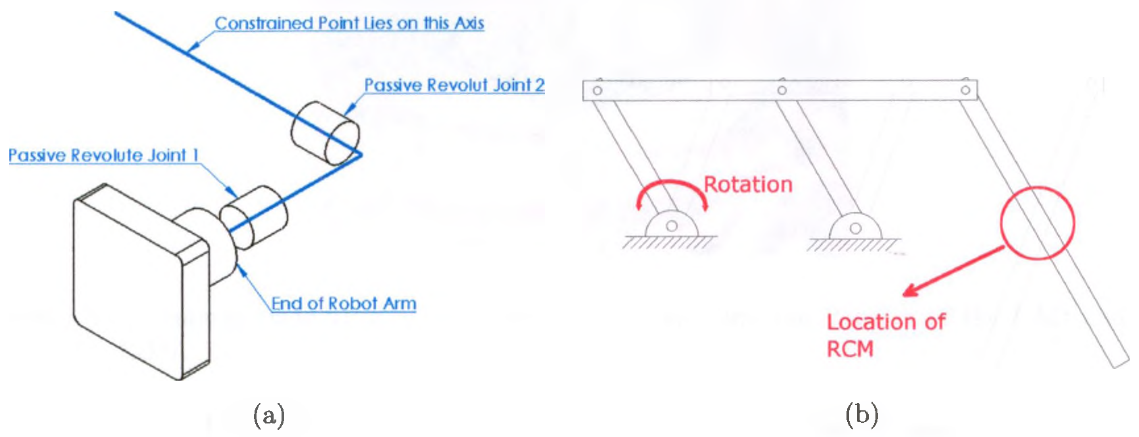


Figure 1.2: Schematics showing the two main methods of creating an RCM with (a) redundant links and (b) a mechanical mechanism.

The two main robots that were created for RAMIS are the da Vinci surgical system and the Zeus surgical system. The da Vinci surgical system is manufactured and sold by Intuitive Surgical, Inc. The Zeus system was created and sold by Computer Motion Inc. The Zeus is no longer available as Intuitive Surgical acquired Computer Motion and no longer markets the device [7]. The Zeus system was made using two passive linkages to create a constrained point on the tool's shaft, whereas the da Vinci uses a double parallelogram mechanism to create a remote centre of motion.

These robots are used for many different surgeries ranging from prostate procedures to cardiac procedures. At the hospital where this research is conducted, a da Vinci robot is used primarily for cardiac procedures; therefore this work will focus on cardiac surgeries, with particular emphasis on the Coronary Artery Bypass Graft surgical (CABG) procedure.

The CABG procedure is performed when a blockage is found in one of the coronary arteries

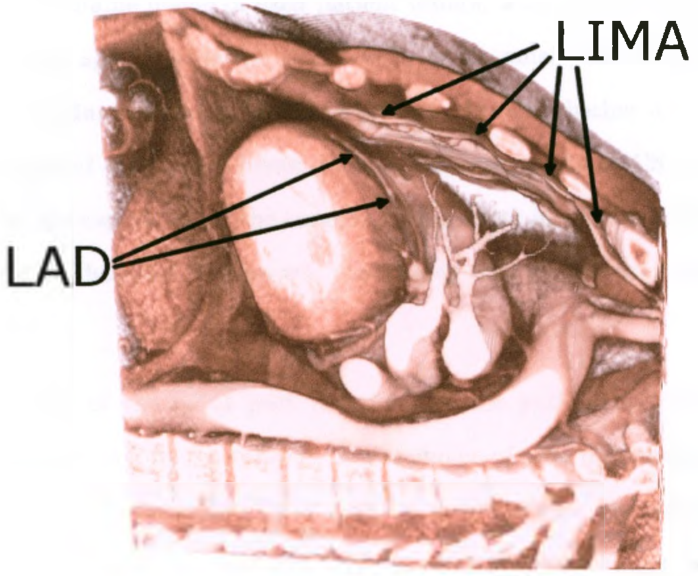


Figure 1.3: An internal side view of a human's thorax, showing the location of the LAD and LIMA.

close to the heart. In this procedure, the surgeon harvests an artery from the patient's body and then sutures it to the blocked artery giving blood an alternate path to follow around the blockage. A commonly blocked artery is the Left Anterior Descending (LAD) artery. This artery runs along the surface of the heart muscle. An artery commonly harvested for this procedure is the Left Internal Mammary Artery (LIMA), which runs along the inside of the patient's ribcage close to the sternum. The locations of the LAD and LIMA can be seen in Fig. 1.3.

For the procedure to be performed robotically, the patient is positioned on the operating table such that the port locations are exposed and the intercostal spaces are opened to their maximum. This is generally accomplished by rotating a patient's left side away from the operating table by about 20°. Then the patient's left arm is placed over his/her head such that the elbow is bent across the patient's face [8]. The robot is then positioned next to the operating table and setup. When the patient has been prepared, the left lung is collapsed, giving the surgeon room to operate the robot inside the patient's chest cavity.

As with the introduction of laparoscopic instruments, RAMIS also adds some disadvantages. The two major disadvantages are:

- The workspace available inside of each patient differs, with some patients having less space than others. If the space inside the chest cavity is too small, the robot may not be able to perform the procedure. It is up to the surgeon to decide whether a patient is suitable for Robotically Assisted Minimally Invasive Cardiac Surgery (RAMICS) or not. This is done based solely on the experience of the surgeon. If a patient is selected for the procedure but is not suitable for it, then the surgeon will most likely need to convert to Open Chest Surgery (OCS).

- The determination of the robot port locations on the patient and the positioning of the robot is at the discretion of the surgeon. If the choice of port placement is poor, the robot's arms may collide during the operation, making it difficult for the surgeon to complete the procedure. In cases such as this, the surgeon may have to change the port locations or perform continuous reorientations of the robot during the procedure. This could lead to an increased operating time and/or more trauma for the patient. In some cases, the surgeon may still have to convert the surgery to a more invasive form of surgery to complete the procedure. This could happen when the surgeon has already harvested the replacement artery and placed it on the heart. The surgeon would then make a new incision right above the heart between the ribs and make use of a retractor to open a space for him/her to reach in and suture the artery in place by hand.

In both of these cases, poor choices could lead to less than desirable results from the surgery. Solutions to reduce these problems include preoperative planning systems or better training methods for surgeons.

One of the main topics focused on in preoperative planning is the placement of ports [9]. The selected port locations are directly linked to the orientation of the robot relative to the geometry of the patient. The general criteria for planning are to reduce the chance of collisions and to maximize the dexterity of the robot during surgery [10].

The first step in preoperative planing for RAMICS is to collect information on the geometry of the patient. This generally comes from a Computed Tomography (CT) scan or from Magnetic Resonance Imaging (MRI). From this information, the space available inside the patient's chest cavity is determined. This is the first factor considered to determine if a patient is suitable for a RAMICS procedure or not.

Once a patient is found to be suitable for RAMICS, the patient's anatomy is then analyzed using medical image viewing software, such as AtamaiViewer or 3D Slicer. The analysis consists of recording the locations of the intercostal spaces along with the locations of the surgical targets. This information is then used to determine the optimal port locations along the intercostal spaces such that the robot's dexterity inside the patient's chest is maximized.

Preoperative planning systems have been designed by various research groups, for different robots and different procedures with good results [9–11].

Currently the methods available to prove results from preoperative planning systems are as follows:

- Actual procedures - If the results from the planning system are used by a surgeon in an actual procedure, the surgeon will be able to report whether the results were successful or not. The downfall to this is that only one set of results can be measured. Two or more results cannot be compared against each other on the same patient.

- Cadavers - If multiple results should be compared to each other, they can be performed on a cadaver. This method can yield more results than the mere pass or fail as with tests done in actual procedures.

- Animal labs - This method would be similar to using cadaver for research and has a drawback, in that it would be expensive to perform. Also, because of the differences in anatomy, some results may not be suitable for translation to humans.

- Phantoms - Using phantoms to prove results can be cheaper and easier than the other methods. Once a phantom is made, it can be used for multiple results and over a longer period of time than with animals or cadavers. Unfortunately phantoms can be time consuming to manufacture and only a limited number of invasive experiments can be performed on a phantom before it need to be replaced. This method is also limited to the geometry of a single patient. Each time a new geometry is introduced, the phantom needs to be modified or replaced.

- Simulations - While all of the methods above are expensive, time consuming and afford a limited number of experiments, simulations do not have these problems. Simulations can be run on multiple patient geometries, with as many different sets of results as desired. It does not take as long to set up a simulation for each patient as compared to phantoms. The drawback to simulations is that they are generally specific to a set of instruments or specific tasks to increase the surgeons proficiency.

A significant challenge in developing a simulator is modelling tissue interaction. This is necessary to provide haptic feedback to the user of the simulator. Haptic feedback enables the user of the simulator to feel when the tool tip is touching or pressing on tissue or other parts of the geometry of the patient. Simulators often use a two dimensional display as the output.

Methods have been developed to model the force response from human tissue [12]. However these techniques are still too slow for real time feedback. Meshes created from medical images for three dimensional representation are large in nature and have a large number of vertices. The larger the number of vertices, the slower the calculations are for deformations of a mesh. The challenge increases if the simulation has to take into account cutting or tearing of tissue [12]. This issue can be reduced by taking the time to smooth the models of each organ and reducing the number of vertices in each organ to an acceptable level, while maintaining the correct overall dimensions. The problem with this solution is that it is only good for a single patient, and takes a long time to prepare.

There are some computer simulation environments available to train novice surgeons to use laparoscopic tools and robots for surgery, such as the LapSim ® [13], i-Sim [14] and the 3-Dmed ® Minimally Invasive Training Systems [15]. There is also a commercial da Vinci simulator created by Mimic Technologies, Inc. These systems have carefully modelled organs and tissue to provide realistic feedback of the force applied to the tool from the tissue. However these simulators are expensive and cannot be modified to represent patient-specific anatomy. These systems are unable to assist in the research being done into preoperative planning.

The latest version of the da Vinci Si has a simulation environment built into the master console of the system, allowing surgeons to practice basic skills with the robot. If more than one master console is available, they can be joined together so that a senior surgeon can watch what a novice surgeon is doing in the connected console. This allows a novice surgeon to perform live surgeries with an experienced surgeon helping out where necessary. Once again, these master consoles are expensive and there is no need for a second console unless it is to be used for training purposes or for challenging surgeries.

Some academic research groups have developed computer-based simulators for RAMICS. These systems were either developed for the Zeus robotic system [16], or for the da Vinci specifically [11, 17]. Each of these groups used a 6 DOF haptics device to control the robot in the simulation environment. Hayashibe *et al.* [16], showed that the positioning error of the end effector in their simulator was reduced by adding in stereoscopic vision, thus making it more realistic and easier to use.

The work performed by the above mentioned research groups introduce the need for medical simulation environments. With research continuing into preoperative planning techniques at this university, the need for a simulation environment came about. Without the ability to obtain a working simulator that can sufficiently aid in current research, this simulator was created.

1.2 Significant Challenges

To simulate the motion of a robot the mass, moments of inertia and geometry of the robot needs to be known. However, most manufacturers do not make this information available and direct measurement is difficult and unfeasible.

The simulator should have real-time interaction which requires a suitable multi-DOF input device, fast kinematic computations as well as fast and accurate scene rendering for the appearance of smooth motions.

Analytical equations are generally derived for a robot based on its geometry and used for its motion calculations. Without knowing the geometry for the robot, the motion calculations needs to be computed without these analytical calculations.

Control systems are based on the robot and the desired control of the robot and are created specifically for a robot. Without knowing the specific robot to be simulated, a generic control system is needed.

Objects with simpler geometries render faster than large complex objects. When working with models of human geometries they tend to be large and complex, increasing the rendering time for a simulator.

1.3 Objectives

The objective of this work is to create a simulation environment that can accommodate any serial-link surgical robot, including the current da Vinci system, to provide a framework for evaluating preoperative planning results. The environment should also give a surgeon the opportunity to use a patient's specific CT scan and practice the surgery in the simulation environment before the actual procedure, allowing any problems that might arise during the real procedure to be identified, and to confirm the desired position of the port locations.

1.4 Contribution

This work covers the development of a framework for a teleoperated medical robotic simulation environment, to aid research in preoperative planning techniques.

The novelty of this work includes the development of a simulator in which a robotically assisted surgical procedure can be visualized to test the results of port placement algorithms. Although there are other robotic simulation environments available, this work allows for the control of general serial link surgical robots, to analyze the kinematic motions of the robot along a trajectory inside computer graphic models of human patients.

1.5 Notation

In this thesis, the following conventions are followed. A scalar is represented by an italic symbol or letter such as η or a. A vector is represented by a lowercase bold letter or symbol such as $\boldsymbol{\epsilon}$ or $\mathbf{x}$. A matrix is represented by an uppercase bold letter or symbol such as $\mathbf{\Gamma}$ or $\mathbf{T}$.

If a matrix $\mathbf{R}$ operates on a frame i, such as rotating or translating it, but is defined in frame

j, then the matrix is written as ${}^{j}_{i}\mathbf{R}$. If the superscript j is omitted then the matrix ${}_{i}\mathbf{R}$ is written for frame i and defined in the world reference frame. Similarly, if the subscript i is omitted then the matrix $\mathbf{R}$ is written for the robot end effector, but is defined in the world reference frame.

1.6 Organization of this Thesis

This chapter provides a brief introduction to the motivation and goals of this research work. In Chapter 2, the simulation software is introduced. Insight into its functionality, along with the methods of generating models for the simulation environment, is provided. In Chapter 4, the control system used for teleoperation based control of the robot in the simulator is explained. Chapter 3 outlines the steps taken to achieve teleoperation of the simulator. In Chapter 5, the experiments conducted for validation purposes are described along with all of the key results. Finally, Chapter 6 concludes this thesis and provides suggestions for future research that could be conducted based on this work.

Chapter 2

The Simulation Environment

2.1 Design Requirements

Before creating the simulation environment, some guidelines were established to make sure the simulator would be as complete as possible. These guidelines are as follows.

1. The simulator should be able to simulate a real surgical procedure as close to reality as possible. This means that as much detail should be added to the simulator as possible, such as surgical targets and also internal organs around the targets to the correct relative scale. Therefore, the surgical targets and human geometry should be obtained from real patients. This would give a surgeon a sense of the workspace they would have had during a real procedure with a patient.

2. The simulator should give the view inside of the patient as if the surgeon was busy looking at the view from the endoscope, which should also be controlled by the user. The lighting inside the patient should be the same as if a real surgical robot was used. This would add to the realism of the device. The option should also be available to alter the light intensity similar to the current surgical robot.

3. The simulator should give an external view with the ability to manually reposition the robot as desired.

4. The simulator should have the ability to import robots designed in CAD software.
5. The simulator should be able to import any generic serial kinematic chain and simulate the motion of the chain, without the user needing to enter the analytical equations to define the kinematic motion.
6. The simulator should not be limited to non-redundant robots.
7. The user should have the option to move the robot with the 7 DOF input device or switch to an external view and re-orient the base of the robot manually.
8. When using the 7 DOF input device, the user should have the option to control both the position and orientation or only the position of the end effector.
9. The user should also be able to change the scaling between the input device and the motion of the robot in the simulation. Some typical values for the scaling should be similar to the scaling found on the current surgical robot. These scaling values, for the da Vinci, are 2:1, 3:1 and 5:1 [18] where the first number represented the distance the input device is moved and the second value is the corresponding distance the end effector would move.
10. If the current surgical robot is being simulated then the simulator should have a similar motion and response to the input when compared to the real robot.
11. The motion of the simulated robot should be controlled in real-time offering the sense of telepresence.
12. The user should have the option to save the scene, including the robots imported and their orientations such that it can be reopened at a later stage.
13. The simulator should be user friendly. This included setting up kinematic chains, scenes and controlling the robot.

2.2 Introduction to the Simulator

The simulation environment was written in C++, making use of the Qt libraries to create the Graphic User Interface (GUI), and the Visualization ToolKit (VTK) libraries for the rendering functions.

The VTK libraries are wrapper libraries for the more complex OpenGL rendering functions. These libraries were created to make the process of writing rendering software simpler. The VTK libraries are compiled for the C++ computer programming language. It is possible to use these libraires in other computer programming languages, but would add needless complexity. Qt libraries were chosen for creating the GUI, as they are the simplest libraries for GUI creation in C++.

The person using the simulator is referred to as the user. Any parts from a robot or other objects added to the simulator are referred to as a body.

The simulator stores a body in one of two classifications, either as a passive body or an active body. A passive body is a body that has not been added to a kinematic chain, does not have a driven joint, is a graphic object to add information to a scene or is a surgical target. An active body is a body that has been linked to other bodies in a kinematic chain and has an active joint assigned to it.

The simulator's main display is composed of an information sidebar and the main three dimensional (3-D) rendering window, as shown in Fig. 2.1. The side bar is comprised of information trees that gives information about all the bodies in the 3-D rendering window. As seen in Fig. 2.1 the upper most information tree holds all the information of all the passive bodies in the rendering window. The following trees hold the information for all the kinematic chains created in the simulator. Multiple passive bodies can be grouped together to form a chain that retains all the relative position information between all the bodies, but has no joints in the chain. This allows easy movement of groups of bodies. In the 3-D rendering window a scene is shown with a patient laying on a table with the model created of the current surgical robot that was in use in surgeries to assist surgeons.

The creation of the scene shown in Fig. 2.1 is explained in this chapter.

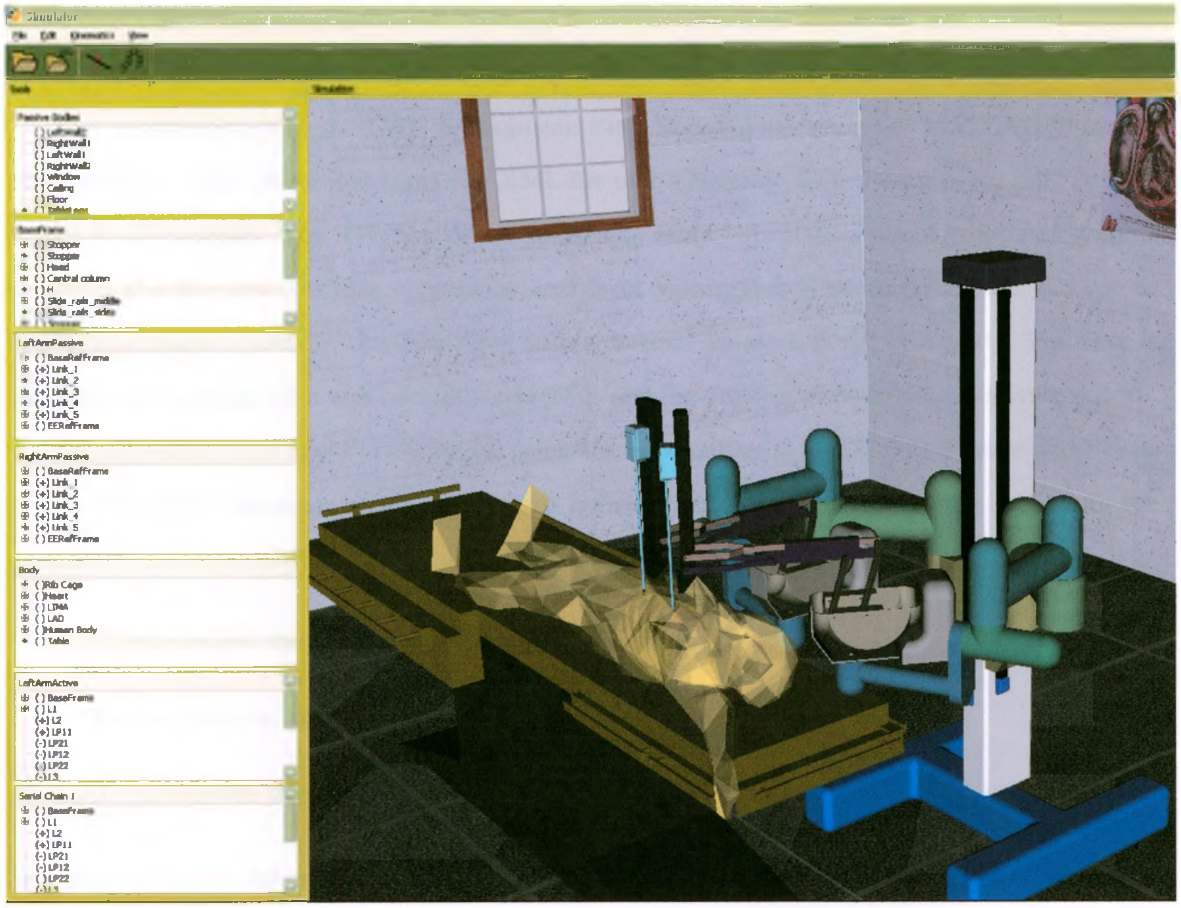


Figure 2.1: A screen capture showing the main layout of the simulator.

2.3 Creating Robots for the Simulator

The simulator does not serve as a CAD modelling tool, therefore no robot can be modelled or altered in the simulator itself. Another tool is required to perform this task and is left to SolidWorks or any other CAD modelling software that has links to MATLAB such as Pro/ENGINEER and Autodesk Inventor, for modelling. The need for the links to MATLAB is that the simulator needs the files generated by the MATLAB linker. This reduces the complexity of trying to interface directly with the CAD programs. A robot created for the simulator is created similar to any other model created in the CAD modelling software. Each part is created separately and compiled into one final file through the assembly process. The only restriction of the model created in CAD modelling software is that only simple mating features can be used in the creation of the assembly.

Joint limits and other such information would be manually added to the kinematic chain created in the simulator if desired.

The robots created in the CAD package can then be exported using the MATLAB linker for SimMechanics. The linker generates one XML file and a separate STL file for every part created in the CAD package. The STL file format is used to store information about computer graphic models. The file contains lists of position and facet values that rendering software can use to recreate a graphic model. The XML file format is used to store structured text. The text is grouped into sections that can be easily searched, making parsing of these files relatively quick.

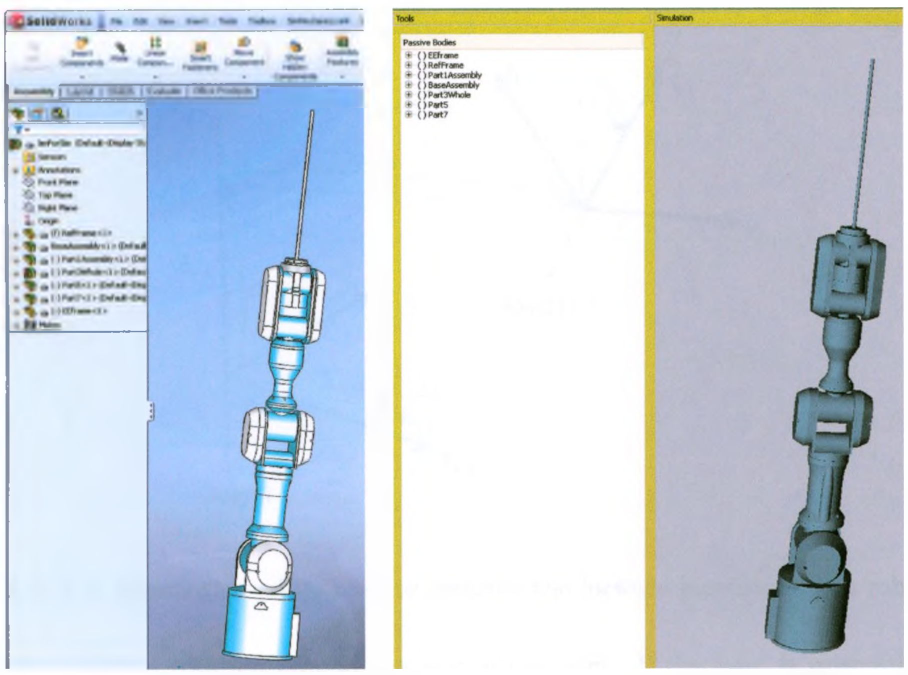
The simulator then imports these files using the information in the XML file generated by the linker. This XML file contains all the relevant information required to import the robot into the simulator. The specific information used from this file for importing is:

- STL file names for each part of the robot.
- The position and orientation of each part.
- All the relative joints between the parts.

and the rest of the information is disregarded, as it contains information such as special linking information between bodies specific for MATLAB's SimMechanics, and scene information such as colours. It should be noted that only SolidWorks was used to make robot models for this project, therefore only SolidWorks will be referred to instead of any other CAD packages.

The simulator then reads and stores all this information and proceeds to import all the STL files associated with the robot and position them accordingly in the scene. It then creates indicators for each available joint in the robot for use in constructing a kinematic chain. Using only the three items listed above the robot can be recreated in the simulator. An example is shown in Fig. 2.2.

If a closer look is taken at the parts imported into the simulator, it can be seen that the surfaces are not as smooth as the ones seen in SolidWorks. The reason for this is that SolidWorks is a parametric modelling tool which exports the parts into an STL format. This conversion can cause problems as SolidWorks attempts to make the STL files as smooth as possible which can result in too many facets created for a part. The effects of the number of facets is shown in



(a) (b)

Figure 2.2: A screen shots of a robot (Mitsubishi PA10-7C) which was (a) created in SolidWorks and then (b) imported into the simulator.

Chapter 5.

In order to have a simulation of the current surgical robot, a model of it was created in SolidWorks. The process of the creation of this model is shown in Appendix B and the resulting robot was used to make Fig. 2.1.

2.4 Robot Joints

Before covering the process by which the simulator calculates the motion of a robot, the basic principle behind robot kinematics is covered first.

2.4.1 Defining Robot Kinematics

A joint is required between two parts of a robot to allow relative motion between them. A joint can be either a revolute or prismatic. If the joint is revolute then the relative motion between

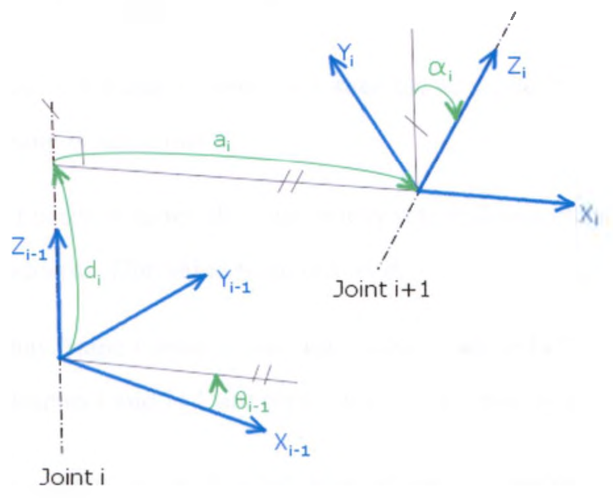


Figure 2.3: The kinematics values used to describe the forward kinematics of a robot [19].

the two parts will be a rotation around the axis of the joint. If the joint is prismatic then the relative motion between the two parts will be a rectilinear translation on the plane defining the joint, along the axis of motion.

A coordinate frame is defined for the joint such that the Z-axis of the frame is coincident with the axis of the joint and the X-axis points along a line that is perpendicular to the axis of the joint and the axis of the following joint in the robot [19].

If a coordinate frame is attached to each joint in a robot, then vectors defined in one frame can be transformed to another frame with a transformation matrix, as shown in the following equation:

$$_{i}\mathbf{v} = {}^{i}_{i+1}\mathbf{T}_{i+1}\mathbf{v}, \tag{2.1}$$

where the subscripts i and $i+1$ refer to the coordinate frames assigned to Joints i and $i+1$. The vector is defined as $\mathbf{v}$ and the transformation matrix is defined as $\mathbf{T}$.

The transformation matrix that can transform a vector between any two joints can be created using only the following four measurable quantities [19], as shown in Fig. 2.3:

- The distance between the two coordinate frames assigned joints i and $i+1$, measured along the X-axis of frame i. This value is known as a_i.

- The angle that Z-axis, of frame i needs to rotate by, to make it parallel to the Z-axis of frame $i+1$. This value is known as α_i.
- The distance frame i needs to move along the newly rotate Z-axis of frame i such that frames i and $i+1$ are coincident. This value is known as d_i.
- Finally, the angle that frame i must rotate, about the newly rotated Z-axis of frame i, such that the X-axes of frames i and i+1 are concentric. This value is known as θ_i.

These four values are known as the Denavit-Hartenberg parameters. When defining a the kinematics of a serial link robot, only these four values are noted for each joint and are generally displayed together in a table. Using these parameters the basic kinematics of the robot is known.

If these four quantities are defined correctly then the transformation matrix that will transform vectors from frame $i+1$ into frame i can be defined as follows [19]:

$$
{}^{i}_{i+1}\mathbf{T} = \begin{pmatrix} \cos\theta_{i+1} & -\sin\theta_{i+1} & 0 & a_i \\ \sin\theta_{i+1}\cos\alpha_i & \cos\theta_{i+1}\cos\alpha_i & -\sin\alpha_i & d_{i+1}\left(-\sin\alpha_i\right) \\ \sin\theta_{i+1}\sin\alpha_i & \cos\theta_{i+1}\sin\alpha_i & \cos\alpha_i & d_{i+1}\left(\cos\alpha_i\right) \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.2}
$$

Using this matrix, the position of the robots end effector and joints can easily be mapped into one coordinate frame and forms the basis of robot kinematics.

It should be noted that the frame assigned to each, is done so with its Z-axis concentric to the axis on which the joint moves. All positive directions are in the direction of the Z-axis or around the Z-axis using the right hand rule method. As this frame is used to calculate the motion of a body, it will be referred to as the kinematic frame in this thesis.

2.4.2 Assigning Kinematic Joints

Once the files has been imported, the user can then assign joints to each body based on the information read in from the XML file. This is accomplished by scrolling through all the joints available from the information file and assigning them to the necessary bodies. Each available joint has a cone representing its location and direction, which is shown in Fig. 2.4. If the joint is a revolute joint the direction of the cone shows the axis about which rotation occurs. Using the right-hand rule a positive rotation of the joint can be determined. If the joint is a prismatic joint then the direction indicates the axis along which motion occurs. It does not matter whether the a joint is assigned in the wrong direction, it is merely an indication of a direction for anyone who wants to use it. The simulator will work correctly whether the user changes the orientation of the joint or not.

The different colours of the cone indicate whether the joint is revolute or prismatic. If the cone is green, then the joint is prismatic. If the colour is red, then the joint is revolute. If the simulator does not understand which type of joint is available, then it will be coloured blue, such as when the XML file lists joints as welds. These joints were displayed as they could come in useful when dealing with many passive bodies in a design.

Manually assigning joints to a robot gives the user the ability to create a robot with different kinematic chains, using the same CAD model. If one joint is left out, then it would be similar to locking a joint in a real robot. It is not expected of the person creating the chain in the simulator to know about kinematic algorithms, merely to know where the robot should have a joint.

A joint can be assigned as either an active joint or as a passive joint in the simulator, which are described in the paragraphs below. Once joints have been assigned to a body, a Cartesian coordinate frame is assigned to the body and is used to calculate any motion of the body. This Cartesian coordinate frame will be referred to as the body's kinematic frame from this point on.

Active joint An active joint is a joint that allows a body to be driven around this joint. It would simulate a joint that has an actuator connected to it, if it was a real robot. If an active joint is assigned to a body, then a kinematic frame is created with the origin of the frame at the same location as the joint position given by the CAD software. The frame is then rotated such that the Z-axis is pointing in the same direction as the joint direction. Therefore, if any rotation is made about the joint, it will then rotate about the kinematic frame's Z-axis. The kinematic frame is then rotated once again such that the frame's X-axis is pointing

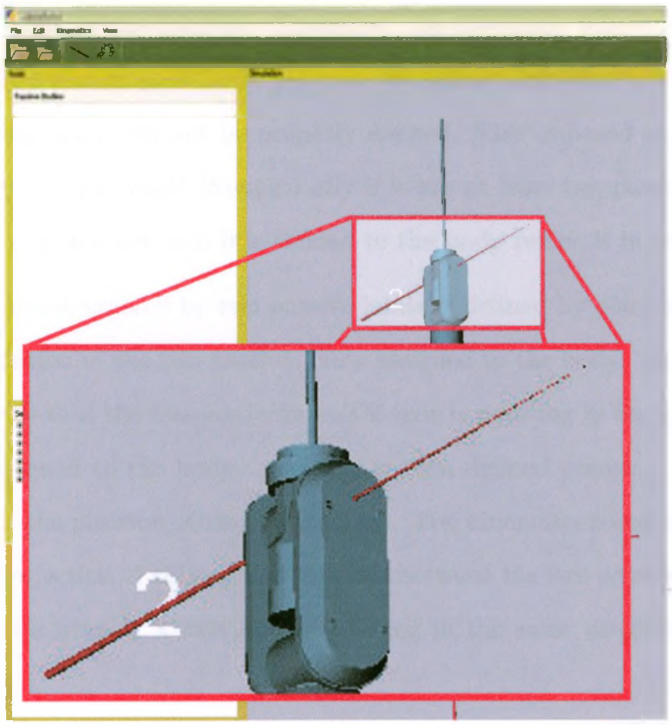


Figure 2.4: A screen capture showing the representation of a joint to be assigned to a body.

towards the body's original coordinate reference frame from its original STL file. If the original coordinate frame is coincident with the joint location, then the kinematic frame is rotated such that the two X-axes are aligned. This gives the frame a unique orientation and will follow the body as it is moved. The frame assignment is done in this manner so that if a rotation or translation is applied to the joint then the rotation or translation will always be around or along the Z-axis of the kinematic frame making motion calculations easier to handle, than if the frames were randomly placed around the body.

Passive joint A passive joint is one in which no actuation can be applied to the joint. The joint is actuated through connecting bodies that have an active joint, or if the passive joint is connected to a body that also has an active joint.

If two passive joints are assigned to a body, a kinematic frame is created to be used for calculating the motion of the body. If either of the passive joints or both of them are moved, then the kinematic frame is repositioned according to the new position of the joint(s), showing the motion of the joint(s). Therefore, the kinematic frame needs to be uniquely defined, if it is randomly positioned between the joints each time a joint was moved then the motion of the body will not be properly defined. This imposed a restriction, in that a body could only be positioned kinematically if it has at least two passive joints assigned to it. If not, the body will act as if it is welded to the body before it in the chain.

The kinematic frame created by two passive joints is defined by placing it at the mid point between the position of the two passive joints assigned to the body. The kinematic frame is then rotated such that the kinematic frame's Z-axis is pointing in the same direction as the second joint assigned to the body. A vector is then defined pointing from the position of the first joint to the position of the second joint. The kinematic frame is then rotated again such that the projection of this vector, that lies between the two passive joints position, lies on the kinematics frame's X-axis and is pointing in the same direction as the kinematics frame's X-axis.

If a passive joint is added to a body that already has a kinematic frame assigned to it, then the relative position and orientation of the joint between the kinematic frame and the joint is stored for calculations when the kinematic frame is shifted.

Any number of passive joints can be assigned to any body, as only the first two passive joints or the body's active joint, fully define the motion of the body. Any subsequent passive joints do not play any part on the motion of the body.

The reason a kinematic frame is defined, is that the coordinate frame of the body from the STL file does not always lie on the same axis of motion. The coordinate frame for the STL file is created in the CAD modelling software and can lie anywhere relative to the body, based on how the part is created in the software. The body's coordinate frame will lie on the axis of motion only if it is specifically created with that purpose. This makes the calculations for motion of a body a difficult task if no new frame is assigned.

Once a kinematic frame is defined, relative transformation matrices between the kinematic frame and body's reference frame is created using the following equation:

$$ {}^{\mathrm{k}}_{\mathrm{b}}\mathbf{T} = \left({}^{\mathrm{W}}_{\mathrm{k}}\mathbf{T}\right)^{-1} {}^{\mathrm{W}}_{\mathrm{b}}\mathbf{T}, \tag{2.3} $$

where k refers to the kinematic frame, b refers to the body's reference frame and W refers to the world coordinate frame.

A transformation matrix is a matrix composed of a rotation matrix and position vector that can transform a vector or another transformation matrix from one Cartesian reference frame to another. Therefore, if the kinematic frame is transformed in any way, the body's new position can be transformed by using the following equation:

$$ {}^{\mathrm{W}}_{\mathrm{b}}\mathbf{T} = {}^{\mathrm{W}}_{\mathrm{k}}\mathbf{T}\, {}^{\mathrm{k}}_{\mathrm{b}}\mathbf{T}, \tag{2.4} $$

where the subscripts and superscripts have the same definition as in (2.3).

These equations can be applied to kinematic frames assigned by either passive or active joints and shows the need for a kinematic frame to be uniquely defined for each body.

2.5 Creating Kinematic Chains

The user can create a kinematic chain by selecting a passive body's names currently in the 3-D display and assigning them to a chain. An example of this can be seen in Fig. 2.5. All the bodies added to the desired kinematic chain have to be added in the correct order from the base body to the effector frame, which is the order they would be connected together in reality. If a group of bodies form a closed-loop chain, they can be added in any order as long as the first body of the closed-loop is the driving body of the loop, which is explained later on in this section.

Once the user has completed the list of the bodies desired in the kinematic chain, the program then reads through all the bodies position information in the list and calculates the relative transformation matrices between the kinematic frames in the chain. If a body is added to the chain without a kinematic frame assigned to it yet, its reference coordinate frame from its STL file is used instead. The relative transformation matrix is calculated using (2.5).

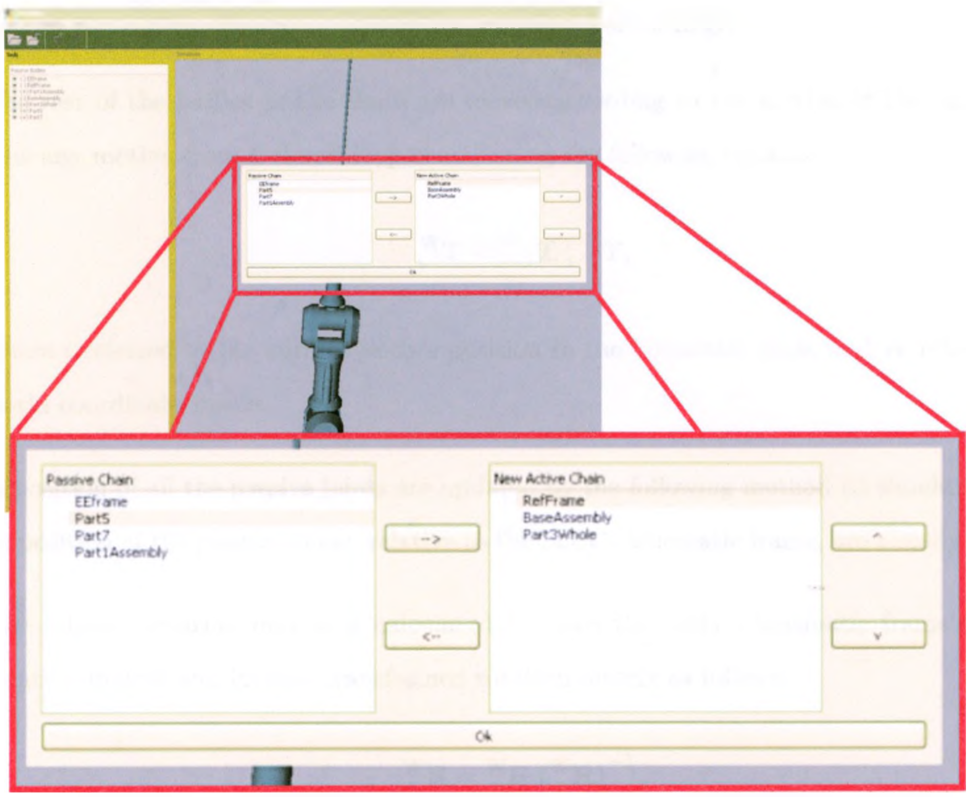


Figure 2.5: A figure showing how the user creates a kinematic chain.

$$ {}^{i-1}_{\ i}\mathbf{T} = \left({}^{\mathrm{W}}_{i-1}\mathbf{T}\right)^{-1} {}^{\mathrm{W}}_{\ i}\mathbf{T}, \tag{2.5}$$

where i refers to the body's position in the kinematic chain and W refers to the world coordinate frame.

If a body is moved in the chain, its kinematic frame would be transformed prompting the repositioning of the rest of the bodies in the chain, using the following steps:

- The relative matrix between the body that is moved and the previous body is recalculated and stored, using (2.5).
- All the positions of the passive joints connected to the body are updated, if there were any, as described below.
- If the body is the driving part of a closed-loop chain, then the closed-loop solver is called

and all the bodies affected by this solver are moved accordingly.

- The rest of the bodies in the chain are moved according to the motion of the initial shift plus any motion from a closed-loop chain, using the following equation:

$$ {}^{\mathrm{W}}_{i}\mathbf{T} = {}^{\mathrm{W}}_{i-1}\mathbf{T}\, {}^{i-1}_{i}\mathbf{T}, \tag{2.6}$$

  where i referred to the current body's position in the kinematic chain and W refers to the world coordinate frame.

The position of all the passive joints are updated by the following method (it should be noted that the position of the passive joints, relative to the body's kinematic frame, are already known):

- The relative rotation matrix is calculated between the body's kinematic frame's current rotation matrix and its new transformed rotation matrix as follows:

$$ {}^{\mathrm{W}}_{\mathrm{r}}\mathbf{R} = {}^{\mathrm{W}}_{\mathrm{n}}\mathbf{R} \left({}^{\mathrm{W}}_{\mathrm{p}}\mathbf{R}\right)^{-1}, \tag{2.7}$$

  where $\mathbf{R}$ is the rotation matrix, W is the world coordinate reference frame, n is the new rotation matrix of the kinematic frame after it had been transformed and p is the previous rotation matrix of the kinematic frame.

- The relative position vector between the positon of the passive joint and the kinematic frame is then rotated using the new relative rotation matrix calculated above.
- The direction of the joint is then rotated using the same rotation matrix as above.
- The joint is repositioned by using the following equation:

$$ \mathbf{P}_{\mathrm{r}} = \mathbf{P}_{\mathrm{s}} + \mathbf{P}_{\mathrm{r}}, \tag{2.8}$$

  where r is the relative position and s is the shift applied to the kinematic frame. All of these vectors are defined in the world coordinate reference frame.

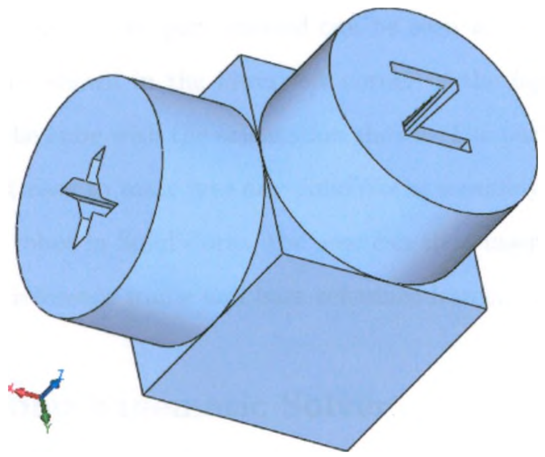


Figure 2.6: A diagram showing the reference frame that would be created from the SolidWorks part designed to give a desired reference frame at the end effector and base frame.

However, if the chain is not a pure serial chain, such as the active section of the da Vinci, more information about the chain is required, otherwise the kinematic chain has been established at this point and the chain is ready for motion calculations.

2.6 Reference Frames

When a part for a CAD model is created, it is created with coordinates relative to a reference frame. If the part is then exported to another file type, such as exporting from a SolidWorks part to an STL file type, all the geometrical relationships are made relative to the parts reference frame. This reference frame does not always coincide with the position of the desired references frame. If special attention is paid to the creation of the part then the part's reference frame could be located at the desired point. However, this should not be expected. So an extra part was designed in SolidWorks such that it can be added to the final assemblies of a robot in SolidWorks to define the desired location and orientation of the frame for the part. This is essential for the base frame and end effector frame for a robot.

The reference frame model was designed such that it is small in size, relative to the current medical robots. It was made small so that it will not have any visual impact on any simulated scenes. It was also designed in so that it had sides showing the positive direction of the Z-axis and the X-axis of its reference frame. The part created can be seen in Fig. 2.6 with the orientation of the parts reference frame shown in the lower left corner of the figure. The parts reference frame lies in the centre of the cube with the orientation shown. The part has both cylindrical and rectangular faces making it easy to mate into any SolidWorks assembly.

If a user creates a new robot in SolidWorks, the user can then insert this part at the location of the desired end effector reference frame and base reference frame.

2.7 The Closed-loop Kinematic Solver

Solving for generic closed-loop chains adds unwanted complexity to the system, so it was decided that smaller solving algorithms would be written and added on to the serial kinematic chain solver already in place. Therefore as new types of closed-loop chains are needed, new solvers could easily be written and added on.

One such solver was written and added on to the serial kinematic chain solver. This algorithm solves the closed-loop chain that makes up the active section of the da Vinci, and is known as the four-bar mechanism solver, which is explained in detail in Appendix A. This algorithm is also used on the endoscopic arm as it has the same basic structure.

If the da Vinci's active section is observed from the side, as shown in Fig. 2.7, it can be seen that the closed-loop chain is composed of two parallelograms connected to each other, with seven joints. The joints can be grouped together in two groups as follows:

- Joints 1, 2, 3 and 4
- Joints 3, 5, 6 and 7

Observing the lower parallelogram in Fig. 2.7, composed of Joints 1-4, it can be seen that the lower two joints (numbered 1 and 2) are fully defined in position and orientation by the motion of the body before the closed-loop chain. It should be noted that Joint 1 is the driving joint of the closed-loop and sets the position of the body it is joined to, therefore the position of Joint 3 is defined in position and orientation as Joint 1 is rotated. That leaves Joint 4, the only undefined joint in the group.

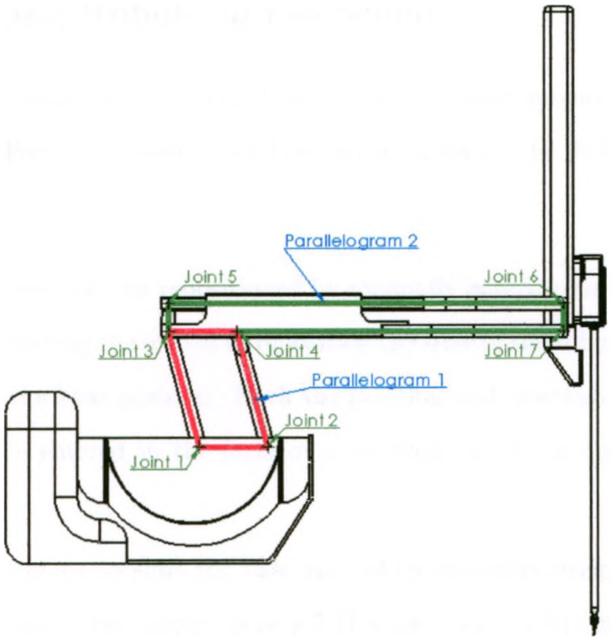


Figure 2.7: A schematic showing the double four-bar mechanism of the da Vinci tooled active section.

As Joint 1 in Fig. 2.7 is rotated then only the position of Joint 4 needs to be calculated and the parallelogram is fully defined and all the positions and orientations of the bodies can be calculated.

Once the first group of joints has been calculated, the second group can be solved for as follows. Once again looking at Fig. 2.7, it should be noted that the body with Joints 3, 4 and 7 connected to it, has already been positioned by the first group of joints, therefore Joint 7 has also been fully defined. Joint 5 is also connected to the body driven by Joint 1, so it too had been fully defined. Therefore, the only joint that needs to be calculated is Joint 6. Once again the position for this joint can be calculated using the four-bar mechanism equations described in Appendix A.

This method is chosen for solving for the closed-loop chains in the da Vinci as the positions and orientations of all the joints in the loop can be solved for by knowing the angle of only the first joint in the loop, and reduces the complexity in defining the closed-loop chain for the user. As mentioned previously, if other closed-loop solvers are needed then they can easily be created and added on to the program.

2.8 Manipulating Robots in the Scene

Once a robot has been successfully imported into the scene and combined into a kinematic chain it can then be manipulated or moved around the scene. This can be done in the following three methods.

- The base of the robot can be repositioned by manually entering new values for its position. This is done by selecting the option to reposition the base of the robot and manually entering in the values for the base position. Both the position and orientation can be entered. The orientation can be entered in the form of a rotation matrix, a quaternion or Euler XYZ angles.

  The option is not given to shift the base around by manually dragging it around with the mouse. This is because the display is as a 2-D screen, with a 2-D input from the mouse, of a 3-D rendering. Which means that the software cannot be sure of the desired location and will place it in unwanted positions.

- Each link can be individually moved by clicking on it and dragging the mouse from side to side. This rotates/translates the body in the robot along/about its axis if it has an active joint assigned to it. If no active joint is assigned to the body, no motion will occur.

- The end effector can be moved around using an external multi-DOF input device. Which is covered later in this thesis. While this option is selected, no user interaction can take place except through the multi-DOF input device.

2.9 Creating Surgical Targets and Physical Features

To increase the realism of the simulator, an option was created for the user to import surgical targets and physical features into the scene. Physical features are defined as parts of the human body that will be taken into account when a surgery is performed, such as the rib cage or heart. Surgical targets are defined as the targets of the simulated surgical procedure, such as the artery harvested to replace a blockage.

Any number of targets or features can be imported into the simulator. Once a complete set of targets and features are imported into the simulator, they can be combined together into a chain without any active joints. This welds all the bodies together such that if one body is shifted, all the other bodies in the chain are shifted accordingly.

2.9.1 Physical Features

Physical features can be created in any manner, as long as they are stored in an STL file format. One method of generating a physical feature is to make use of a medical imaging software such as 3D Slicer [20].

The segmentation process begins by opening a patient's CT scan in 3D Slicer. Segmentation is the process by which pixels are labelled according to their intensity in the scan and recorded. This can be done because matter of different densities have different pixel intensities in the CT scan. The range of notable differences in densities is from air to bone, with air being the darkest pixel intensity and bone being the brightest. Therefore, if only the bone structure is required in the segmentation, all the pixels with an intensity above a given threshold can be segmented out. This process can be performed automatically but CT scans are noisy and objects such as the table the patient is lying on will appear as bone. Therefore, not only would the bone be segmented out but also some other random spots will be included. Also, if only the ribcage is desired and the CT scan includes the arm and other bones the software will not know the difference. This means that if a nice and clean segmentation is sought, manual segmentation must be done. Fig. 2.8 compares the result from a manual segmentation and an automatic segmentation.

Manual segmentation of a CT scan is performed by looking at each slice in the CT scan and physically marking off the areas that are desired in the segmentation. The process of manual segmentation is shown in Fig. 2.9. Manual segmentation is a time consuming process, which is useful in the end to generate a complete and clean model. The higher the quality of the scan, the more slices there are and the more time it takes to segment it. Some scans of the chest can have as many as 300 slices in it. The lower the quality of the scan, the fewer slices are available, but less information can be found in the scans too. It is difficult to determine the location of the arteries in a low quality scan, therefore high quality scans are generally used.

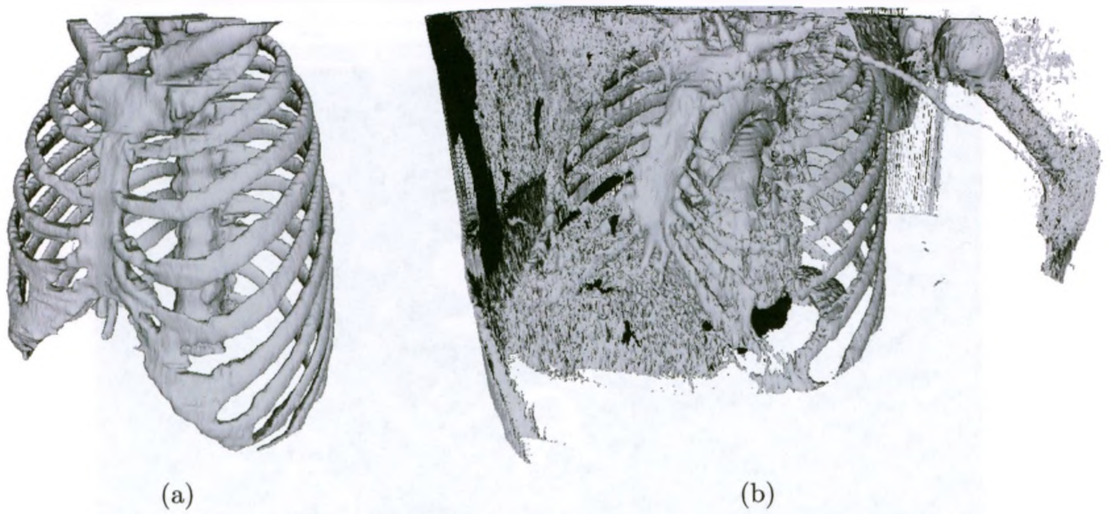


Figure 2.8: An image showing the difference in the resulting 3-D model created by a clean manual segmentation (a) and an automatic segmentation (b) of a human ribcage made from a CT scan.

Once all the slices in the CT scan are correctly segmented, then the segmented pixels can be used to generate a 3-D model. The model is created using the marching cubes algorithm built into 3D Slicer. The output of the algorithm is a set of polydata that can be saved in the format of an STL file.

These STL files have a large number of facets, which is bad for rendering and memory management. The large number of facets from a clean segmentation cannot be prevented. To reduce the number of facets in the STL file, the following series of cleaning steps can be performed on it.

The number of facets and vertices are first reduced using the algorithms built into the software package MeshLabs. These algorithms merely subsample the amount of points in the file while keeping the overall geometric shape constant. The STL file is then opened in the software package Blender and any unwanted vertices, created by noise or other problems, are manually deleted, further reducing the number of facets in the file.

2.9.2 Surgical Targets

Surgical targets for RAMICS are generally arteries or veins. To create models of the surgical targets in the simulator the software AtamaiViewer [21] is used. A module was found in the

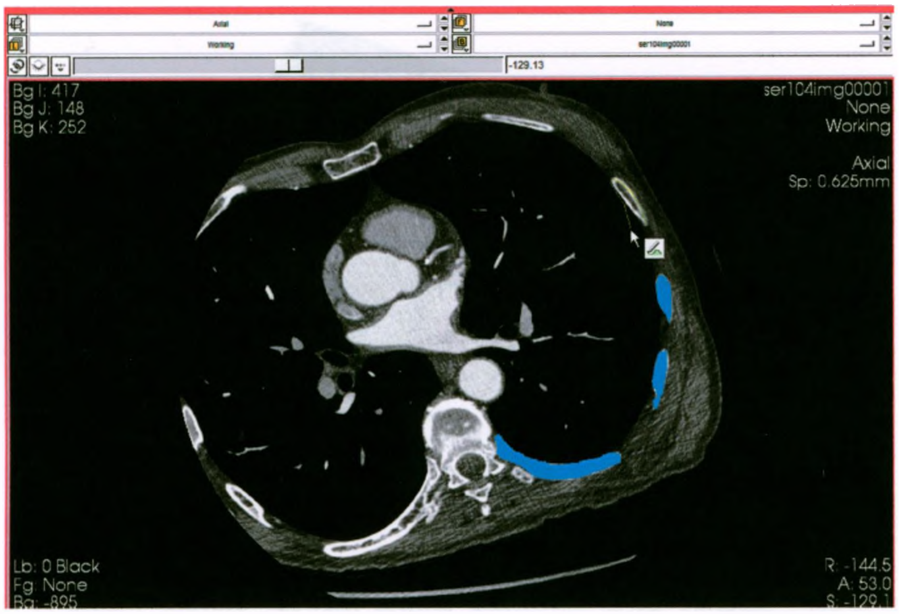


Figure 2.9: A screen capture showing the process of manual segmentation on a CT scan in 3D Slicer.

software that allows a person to pick out intercostal spaces in a patients CT scan and generate splines from the chosen points. This module was modified to allow a person to chose more inter costal spaces and also allow the user to pick points along desired surgical targets and specifying the surface on which the targets lay. The result of this module is shown in Fig. 2.10. The cones in the figure above the surgical targets are surface normals to indicate the approach vector needed to avoid the surface it is lying on, to assist preoperative planning research.

The targets are stored in a file as a set of points along each spline. These points can be read in by the simulator, which recreates the splines to represent the surgical target in the simulation environment.

2.10 Customizing a Scene

The user has the option to modify most of the graphical representations of the bodies in the simulator.

The user has the option to change the colour of each body in the scene. One body cannot have

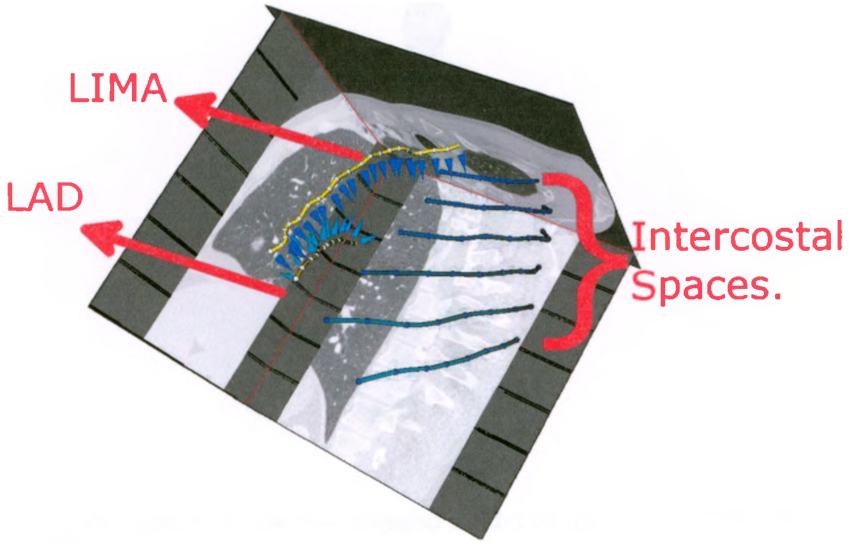


Figure 2.10: A screen capture showing splines generated along surgical targets in the AtamaiViewer. The surgical targets selected were the intercostal spaces, LAD and LIMA. The LAD and LIMA had normals along their splines to help the preoperative planning algorithm.

multiple colours assigned to it as the bodies are not defined parametrically. This means that an individual face cannot be selected to assign a specific colour to it. The scene's background colour can also be changed as desired.

The view from the camera can be shifted by clicking on the scene's background or a passive body and dragging it around. If the left mouse button is used, the camera rotates around its focal point. If the right mouse button is used the camera pans in the dragged direction. The camera can zoom on the focal point by clicking the middle mouse button and moving the mouse up and down relative to the display.

A new camera can be placed in the scene by attaching it to the end effector of a kinematic chain. When this is done the simulator creates a second 3-D display and opens it up with the display from the new camera. If the given kinematic chain is then moved in the original rendering window the camera is moved accordingly.

To complete the scene, models of a surgical table and human body were also created. If only the internal structure of the patient is observed through a simulation, problems can arise with the external structure of the human body interfering with the motion of the robot. So a general

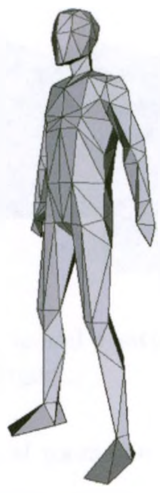


Figure 2.11: A screen capture of the low polygon model of the human body created, showing the vertices that make up the model.

model of a human body was created to represent the general area that is be taken up by a patient through a surgery. The surgical table was also measured to show its geometry and any problems it can create during the procedure.

As the human body is relatively large and a complex object, it could consume necessary computer memory to rendering it, so a low polygon model was created. A low polygon model means the body represented will appear rough and blocky, devoid of any natural contours. The advantage of using a low polygon model is that the rendering time is increased. An image of the low polygon model created for this project is shown in Fig. 2.11.

The body model was initially created using photographs of a person from the front and the side. This process was done in Blender, where one half of the body can be modelled and then mirrored to created the other half. The pose of the model created is not similar to a person lying on the surgical table, so the legs and arms were shifted to represent the pose of a patient on the table ready for a surgical procedure. This is shown in Fig. 2.12.

The human body model is scaled relative to the surgical markers from 3D Slicer and AtamaiViewer; therefore if the model is imported together with the surgical targets into a single program, they are all done so relative to the same frame of reference and will automatically be placed in the correct position and orientation.

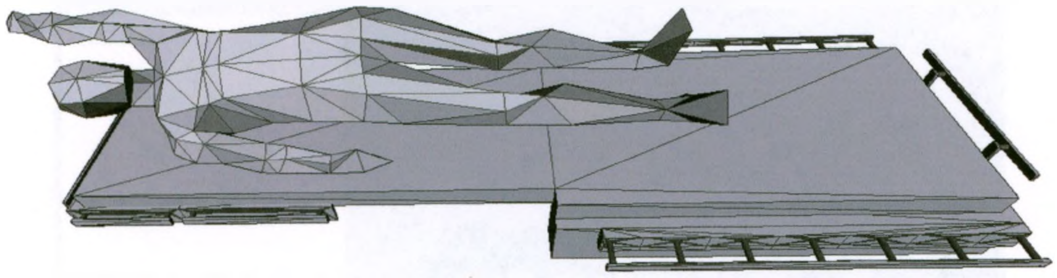


Figure 2.12: A screen capture of the body model positioned on the surgical table in a pose similar to a real patient ready for surgery.

Finally, to give the illusion of a surgical room the option is also given to add in planes with pictures attached to them which can be seen as walls, such as the walls shown in Fig. 2.1.

2.11 Saving and Loading

The option to save the current setup was created. This means that at every stage in setting up a scene it can be saved and reloaded if a mistake is made. The undo functionality has not been created yet and is listed as tasks to still be completed.

Two options are available for saving. Firstly, the user can save a single kinematic chain, which would record the position and orientation of each body as well as any joints assigned to a body. Secondly, the user also has the choice of saving a scene.

If the scene is saved, all the kinematic chains in the scene will be saved into the file, along with all the passive bodies and other scene information. A single kinematic chain cannot be extracted from this file. Each body of a chain is written to an STL file and a new information file is created along with the STL files with all the information required to recreated the chain or scene. This means that the initial files created from SolidWorks are no longer needed. All other information about the scene is also recorded, such as the camera position and orientation and the colours and images attached to bodies.

By having the option of saving a chain, a scene can be created quickly if a scene is composed of more than one set of the same robot. They can be imported individually into a scene and used in the scene as necessary. A scene is shown in Fig. 2.13 that was created using the method of

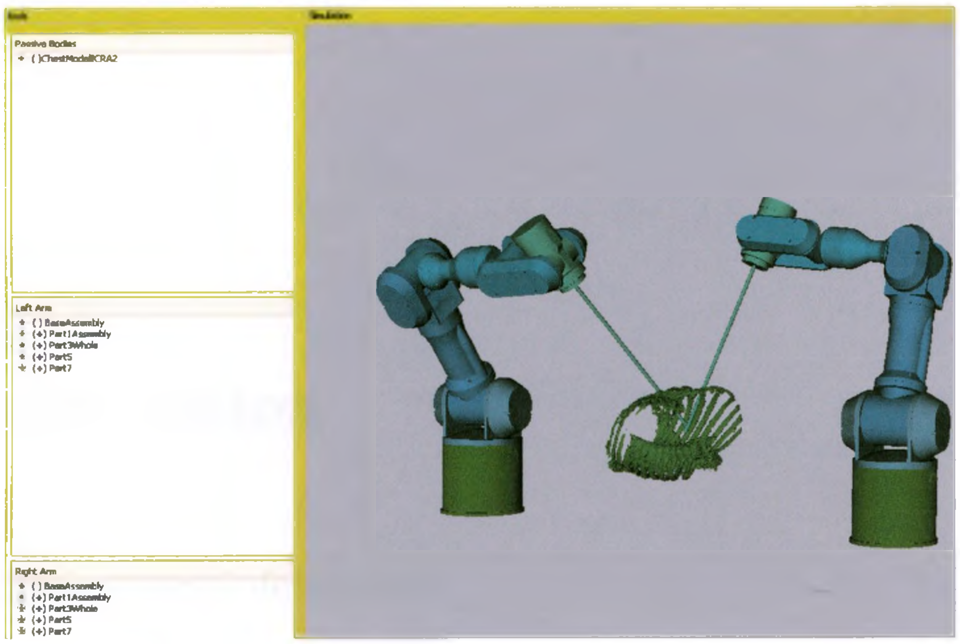


Figure 2.13: A screen shot showing a scene created from importing multiple chains.

importing multiple chains.

2.12 Conclusion

This chapter outlined the framework for the simulator created for this thesis. It listed the design requirements and showed how the simulator was created to match them.

The simulator created for this thesis transforms the complex task of performing motion calculations, of a serial link manipulator, into the simple task of assigning joints to a CAD model and sorting them into the correct order. Even though a method was not created to solve generic closed-loop chains, a method is shown where simple closed-loop chains can be solved by adding algorithms on as needed.

The process of creating models of a patients geometry and surgical targets is explained along with methods for reducing the memory size of these models. A generic model of a human body and surgical table was also created to represent boundaries for the positioning of a robot in the simulator.

Chapter 3

Teleoperation

3.1 Background Information

Many robot simulators have been created to date and they can be grouped into two main categories, the first being an offline and the other being a real-time simulator. The difference between the two categories, is when a robot's motion is calculated and rendered to the display. The offline simulator does all the calculations before any rendering is done, whereas real-time simulators do the calculations and rendering concurrently and in real-time.

Both of these simulators can perform simulations for predefined trajectories, but only the real-time simulator allows for continuously changing trajectories while the simulation is in progress.

For this work, both predefined and continuously changing trajectories need to be handled. The continuously changing trajectories are from inputs from surgeons running through surgical procedures with a multi-DOF input device. This method of simulation is referred to as teleoperation. The predefined trajectory can come from the results of a preoperative planning algorithm for testing.

3.2 Requirements

In making the simulator teleoperated, a list of guidelines needs to be established beforehand to make sure the resulting system is user friendly and complete. The following set of requirements are shown here, which outline a successful teleoperated system.

- A teleoperated simulator should have a simple and intuitive method of entering the desired position and orientation of the robot. The simulator should work in such a way that the user would have the perception of directly controlling the end effector of the robot. This is known as telepresence.
- The most challenging part of teleoperated simulators is to give the user a sense of telepresence, while maintaining the stability of the system [22]. For stability of the system, the robot should be sufficiently controlled regardless of the motion required from the user.
- It should be possible to control the simulated robot from a distance.
- The robot should follow the given trajectory as closely as possible.

3.3 User Input

For teleoperation of the simulator software, the user of the software needs a form of input that can control all the DOFs of the robot, rather than a simple 2 DOF input such as the computer mouse. One such device available for this project is the PHANToM Premium. The PHANToM Premium is a haptic input device produced by the company Sensable Technologies, Inc. The device gives the user the ability to intuitively give a desired position and orientation for the end effector of the robot. It is similar to what a surgeon could use in a RAMIS procedure. It does differ from the inputs found on the master controller of the da Vinci in both dimensions and in the number of DOFs. Although the PHANToM Premium device has 7 DOF, it only has 6 DOF for position control and a switch on the handle to give a seventh that can be used for simulating the closing of jaws on the end effector.

The controllers on the da Vinci have a much larger workspace, allowing the surgeon almost full use of his/her arms. The da Vinci's controllers also have an extra DOF, to prevent any kinematic singularities. The implications of these differences is that the PHANToM Premium would have to be readjusted more often in a simulation compared to the readjustments required by the da Vinci's controllers.

Although the device used for this project has haptic feedback, no force was reflected to the device from the simulator for this project due to time constraints. This is listed as one of the possible future projects based on this work. The PHANToM Premium only has force feedback along the Cartesian axes, but has no method for applying any moments about the Cartesian axes. A picture of the haptic device is shown in Fig. 3.1.

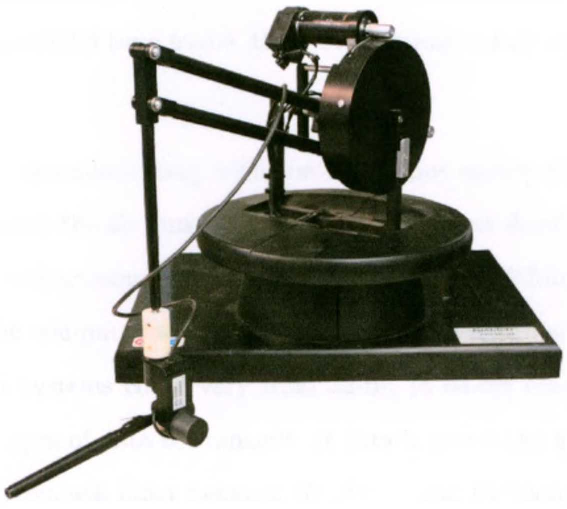


Figure 3.1: The PHANToM Premium input device used to control the simulator.

Rather than having this device as the only possible input for the simulator, the simulator was created to accept inputs from devices on a network. Therefore, if multiple inputs are needed, they can simply be connected to a computer and joined to the network to send the desired trajectory to the simulator. If a surgeon would like to test the results of a preoperative planning algorithm in the simulator, the surgeon would want multiple inputs, similar to the real surgical robots. Also, by connecting the device to the network it frees up resources on the main computer.

By placing the input on a network, the performance of the simulator is not reduced due to latency, as long as there is a small number of devices on the network. For this work, there was only one network switch and two computers connected together. Also, the amount of information sent between the computers is small and at widely spaced intervals. Therefore, this is currently not a performance issue. However, if this system is setup on a busy network with many devices connected to the network, it could cause lag between the motion of the input device and the main computer receiving the updated position.

The network protocol that is used for the simulator is a simple ad-hoc communication protocol. The protocol was as follows:

- The main computer would ask for the position information from the first haptic device. It would wait for six pieces of data in response. If the main controller does not receive all six pieces of data in a specified time frame, the system would ignore any information from that device.

- Once the computer communicating with the first input device received the request for a response, it would send the six current values from the input device. The six pieces of data are composed of three Cartesian coordinates and three Euler XYZ angles. Once the six pieces of data are sent, the computer would not send any more information until more positions were requested. As systems could vary from 32-bit to 64-bit machines, it is necessary to choose the correct types of data to transmit. If data is sent in an integer format, the values for the same set of data will differ between the 32-bit and 64-bit machines. That is why all the data is sent in a double format as both types of operating system use the same type of memory allocation for a double variable.

- Once the main computer has successfully received all six pieces of data, it converts the Euler XYZ angles into a rotation matrix and uses these values to define the error measure for the control system of the specified robot.

- This process is then repeated with however many input devices are said to be connected to the system.

A diagram showing a network structure with multiple PHANToM Premiums is shown in Fig. 3.2.

The maximum possible number of concurrent control systems on the main computer would be the limiting factor to the number of input devices connected to the system. With the update rate of 1 kHz from each PHANToM Premium device and a network communication speed of 10 Mbps, more input devices could be put on the system than the amount that could be used by

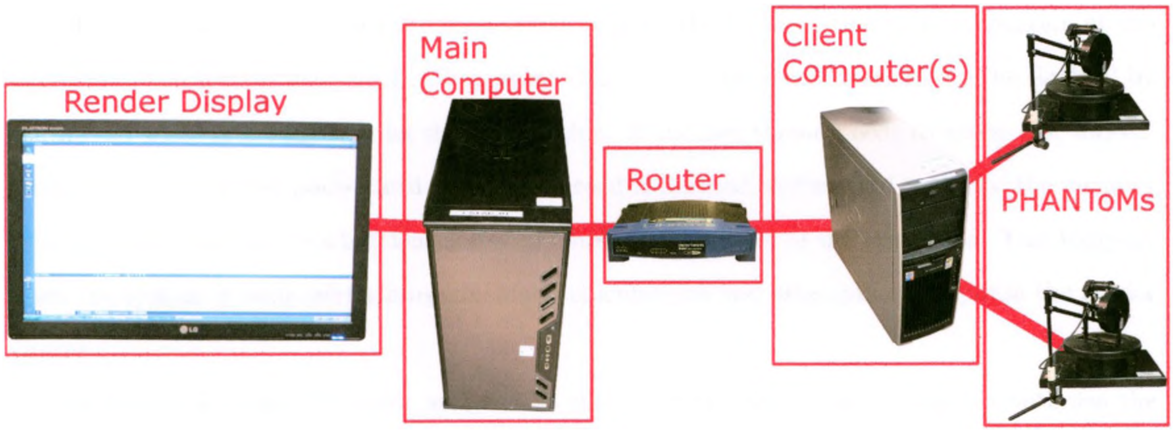


Figure 3.2: A diagram outlining the network structure used in communicating with the PHANToM Premium input devices.

the main controller. Every time the input device is sampled, the resulting desired position for the control loop is updated. Although the update rate of 1 kHz seems fast, the control loop was able to complete 2 to 3 loops between each position update. This was the case for all the robots tested for this work.

3.4 Multi-threading

To make the system perform faster without changing the hardware of the main computer, the simulator was changed to run in multiple threads. This means that sections of the simulators code could be run concurrently by the computer, speeding up the computations as compared to performing calculations sequentially.

By dedicating an important task to run in its own thread, the computer will have more resources to perform other necessary calculations. Therefore, it is best to add as many threads to the system as possible, allocating a thread to every critical task. However, this is not possible. Only algorithms that can be calculated independently of each other can be assigned to their own threads. If one algorithm depends on the result of another, then both those algorithms should be run in the same thread. Most operating systems allow for more threads than the simulator would need.

One problem that arises with multi-threading is sharing of variables between threads. If one thread tries to access a memory location while another thread is writing to that location at the same time, the system may crash. To prevent this, certain memory locations can be 'locked' by one thread while it is busy writing the information. If another thread needs to access the 'locked' memory location it will pause until the other thread is finished writing and 'unlocks' the memory location. This can lead to a bottleneck if some threads share a lot of the same data. This happens when the system is busy performing kinematic calculations and attempting to update the render display.

The following tasks were each assigned to their own threads in an attempt to optimize the motion calculations during a simulation routine.

GUI This is the main thread for the system. This thread is the initial thread that is created when the simulator is started, and all the other threads are created from this one. All flags are stored in this thread and shared to all the subsequent threads, making it easy to share the flags between all threads.

This thread also shares memory allocations with the control algorithm thread. This is because, the control algorithm will manipulate the position values for all the bodies in the render window.

Network Communication This thread establishes communication with the client computers and handles all the information sent between the computers. This thread also reads in all the positions and angles, sent from the client computer, and converts them into a position vector and rotation matrix for the control algorithm.

This thread shares only the memory locations of the desired position and orientation with the control algorithm thread.

Control Algorithm The control algorithm thread reads the values from the position vector and rotation matrix collected by the network communication thread to create the error measures. It then runs through the loop of the control system. In the loop, all of the kinematic calculations are performed. New positions and orientations for all of the bodies in the render window are also calculated and updated. This is the thread that is the most critical and generally takes the longest to repeat. This thread is looped continuously or until the user ends the control algorithm manually.

If a new position is not available from the user input at the time the loop repeated itself, the previous values are reused for the following iteration of the loop. The faster this thread loops, the better it will perform, as it will continuously reduce the error between the desired position of the robot's end effector and the desired position.

This thread shares memory with the trajectory recorder thread as it sends the desired and current end effector positions, for analysis. It also shares memory with the GUI thread and network communications thread, as mentioned before.

Trajectory and Error Recording To measure the accuracy and response of the control system, a thread was dedicated to recording desired and current position of the end effector every time the control algorithm completed a loop.

When a simulation is ended, this thread would store the results in a text file for later analysis. If the option is selected, then this thread would also graph all of the results upon completion.

This thread shares memory with the control algorithm thread, as mentioned before.

Render Timer A thread was created with a dedicated timer, that sets the render window to update itself at a specified rate. The render window is run in the GUI thread, which is continually updated by the operating system. As the update function for this thread could not be changed, the rendering refresh function was rewritten for the simulator. This allows the simulator to manually update the render window whenever it is needed, and prevents the operating system from updating it. This ensures that the simulator will not crash, if the system tries to update the render window while the control system is busy updating the positions of the bodies in the render display.

3.5 Rendering

The time it takes to render a scene can vary in time based on the bodies in the scene, and could sometimes take longer than desired, which will be quantified in the following chapter. Therefore, the updating process is run while the control algorithm is performing kinematic calculations.

To keep the process flowing smoothly and to prevent the appearance of any jittering, the render window is only updated at a rate of 30 Hz. While humans can still discern motion between frames or blurring at 30 Hz, this effect is negligible for small motions and found to be suitable for the simulator. If the rendering rate is increased it can affect the control algorithm timing and slow the whole system down, sometimes making it unstable. If the rendering time for a scene is too large making the 30 Hz update rate impossible, the simulation is stopped so bodies can be removed from the render window before another simulation is attempted. When a simulation is started, a visual reference is given to the user to show the maximum update rate of the render window, based on the scene's rendering times. This will allow the user to remove bodies from the simulator if the rendering rate is too low.

3.6 Conclusion

This chapter outlined the basic features and some of the difficulties in creating a teleoperated simulator. It then showed how the simulator, created for this work, attempted to best address these features and overcome the problems.

By creating a network to allow for multiple multi-DOF input devices also allows for the prospect of control from a distance. If the simulator software is installed on a server, and a high enough bandwidth was available, a user could run the simulator from a distance.

The network communication protocol is a simple ad-hoc protocol. This method is sufficient for the proof of concept, but will not suffice in the long run. A more stable communication protocol with error checking, such as a check sums, should be instituted.

By using the multi-threading, the system can run and be stable on the hardware used for this work. However, it did reach the limits of the computer. If a more complex simulation, or more robots are needed in a simulation, a more powerful machine is needed.

Chapter 4

Control System

4.1 Background Information

For the implementation of teleoperation, the user of the simulator has some means of entering a desired position and orientation for the end effector of the robot, and the control system will then move the robot to match the input from the user. The means of entering this information, for this work, is a multi-DOF input device. It is important to have the end effector match the motion of the user's input device as closely as possible. If this is accomplished, the perception of the user will be that they are directly controlling the end effector.

For this to work, a fast control algorithm is needed. The control system will take the user's input and calculate the angles of each joint for the robot's end effector to match the motion of the user's input in realtime.

Most control systems strive to control either the position and orientation of the robot's end effector or the force or torque the end effector exerts on a surface. These control systems use various algorithms to achieve the desired result. The control system is chosen based on the type of robot being controlled and the work the robot is made to do. The choice of control system will have an impact on the performance of the robot [23]. This work is focused on the position and not the force exerted by the end effector. Therefore, it focuses on position control systems.

There are many different control systems available for controlling robots. These can range from independent joint control [23] used for preplanned motions, to adaptive control systems [19], that can continuously update system variables to increase the performance of the system. These control systems will be narrowed down by the type of trajectory the robot will follow. For this work, a continuous path tracking system is needed as opposed to a trajectory that is straight lines between discrete points.

Control systems generally make use of the robot dynamics equation:

$$\boldsymbol{\tau} = \mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \mathbf{V}(\dot{\boldsymbol{\theta}}, \boldsymbol{\theta}) + \mathbf{g}(\boldsymbol{\theta}), \tag{4.1}$$

where $\boldsymbol{\tau}$ is a vector of torque applied to each joint of the robot to achieve the desired input trajectory. $\mathbf{M}(\boldsymbol{\theta})$ is the robot's mass matrix, $\mathbf{V}(\dot{\boldsymbol{\theta}}, \boldsymbol{\theta})$ is a matrix of inertial and Coriolis forces and $\mathbf{g}(\boldsymbol{\theta})$ is the gravitational force based on the pose of the robot. $\boldsymbol{\theta}$ is a vector of the robot's joint angles, along with $\dot{\boldsymbol{\theta}}$ the joint velocities and $\ddot{\boldsymbol{\theta}}$ the joint accelerations.

The control systems derived from (4.1) are based on extensive knowledge of the robot, including the robot's dynamic equations and the mass of the robot. This conflicts with the design of the simulator, where the simulator does not require any analytical equations or the mass of the robot for the robots being simulated.

As noted before, a single control system cannot control all robots correctly. Therefore, the control system selected for this work was designed in a modular manner, allowing for other control systems to be implemented as desired. The control system chosen for this work was done so for initial testing purposes.

4.2 Control System Selection

A simple [24] and intuitive [23] type of control system was found for the simulator. The type of control system found is referred to as kinematic control systems. These control systems only take into account the current pose of the robot and the linkage dimensions.

Kinematic control differs from dynamic control in that the system dynamics are not modelled. Therefore, all inertia and link masses are not described and are ignored. The system will not have damping or overshoot in the conventional terms. The effects of forces such as gravity are also ignored. For this simulator, the dynamics can be ignored as only the kinematic reach and pose of the robot is needed to check results from preoperative planning algorithms. The simulator is not focused on the dynamic response of a robot, but merely its kinematic behaviour.

Firstly, a matrix called the robot's Jacobian needs to be defined. The Jacobian is a matrix used to map a robot's joint velocities into its end effectors Cartesian velocities and rotations. This means that if the robot's joint velocities are known, by using the Jacobian, the velocities and rotations of the end effector can be calculated.

The Jacobian is a first-order temporal derivative of the forward kinematics of the robot [19] and is defined as follows:

$$\dot{\mathbf{X}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}, \tag{4.2}$$

where $\dot{\mathbf{X}}$ is velocities and rotations of the robot's end effector (EE), $\dot{\mathbf{q}}$ is a vector defining all the first time derivatives of the joints in the kinematic chain and $\mathbf{J}(\mathbf{q})$ is the Jacobian.

It should be noted from (4.1) that the Jacobian is a function of the joint values, $\mathbf{q}$. Therefore, the Jacobian needs to be recalculated as the pose of the robot is adjusted.

If both sides of (4.2) are premultiplied by the inverse of the Jacobian the following equation is obtained:

$$\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{-1}\dot{\mathbf{X}}. \tag{4.3}$$

Given (4.3), it can be seen that it should be a simple process to determine the desired joint velocities, if given a vector of desired end effector velocities and the pose of the robot is known. Their are control systems based on this equation and they are known as the Jacobian inverse control system [24].

The challenge with this type of control system is when the pose of the robot changes, the Jacobian can lose rank and the inverse of the Jacobian will not exist, making it impossible to calculate the desired joint velocities. This is known as a kinematic singularity. Even if the robot is in the vicinity of a singularity the calculated joint velocities will be amplified, leading to erroneous and unstable control. It is not known when a singularity might occur. To prevent kinematic singularities the rank of the Jacobian is monitored.

These kinematic singularities can be handled with different methods. Some of the more common ways is to map out the workspace of the robot and identify all singularities. Then the control system can only allow motions in a subset of the workspace where there are no singularities, or if the robot has redundant links, a redundant link can be moved as the singularity is approached to prevent the Jacobian from loosing rank.

Another method of solving this equation is to use a *pseudoinverse* of the Jacobian instead of the inverse, which is useful is the Jacobian is not square. This method is referred to as the Moore-Penrose *pseudoinverse* method [25] and is shown here:

$$\dot{\mathbf{q}} = \mathbf{J}^{*}(\mathbf{q})\dot{\mathbf{X}}. \tag{4.4}$$

The *pseudoinverse* $\mathbf{J}^{*}$ is defined as $\mathbf{J}^{*} = \mathbf{J}^{\mathrm{T}}(\mathbf{J}\mathbf{J}^{\mathrm{T}})^{-1}$ which results in a square matrix and gives the minimum norm joint velocities [25]. However, the *pseudoinverse* can still have singularities.

The problem with this approach is that it can be computationally expensive and that the kinematic singularities are not completely avoided, as it is still prone to calculating amplified velocities near singularities [25].

Another method that builds on the *pseudoinverse* adds constraints to the system. This leads to the calculation of an extended Jacobian [26] which increases the complexity and the computations required to obtain a result. This method also leads to algorithmic singularities from the constraints.

A simpler solution was posed by Wolovich and Elliot [27], to use the Jacobian transpose rather than using any matrix inversion. This control system is used for this work, as it does not have workspace singularities. This means that the workspace of the robot does not need to be mapped out beforehand to test for singularities. This control system is derived and discussed in the following section.

4.3 Control Law Derivation

By using the principle of virtual work the following statement for each joint in the robot is true [19]:

$$\mathbf{F}^{\mathrm{T}}\delta\mathbf{x} = \boldsymbol{\tau}^{\mathrm{T}}\delta\boldsymbol{\Theta}, \tag{4.5}$$

where **F** is a Cartesian force or moment applied at the end effector. $\delta\mathbf{x}$ is an infinitesimal displacement vector of the end effector. $\boldsymbol{\tau}$ is a vector of the torques applied at each joint and $\delta\boldsymbol{\Theta}$ is a vector of infinitesimal displacements of each joint.

Given the definition of a Jacobian:

$$\delta\mathbf{x} = \mathbf{J}\delta\boldsymbol{\Theta}, \tag{4.6}$$

where **J** is the Jacobian, and substituting it into (4.5) the following is obtained:

$$\mathbf{F}^{\mathrm{T}}\mathbf{J}\delta\boldsymbol{\Theta} = \boldsymbol{\tau}^{\mathrm{T}}\delta\boldsymbol{\Theta}, \tag{4.7}$$

therefore,

$$\mathbf{F}^{\mathrm{T}}\mathbf{J} = \boldsymbol{\tau}^{\mathrm{T}}. \tag{4.8}$$

by transposing both sides of (4.8) the following is obtained:

$$\boldsymbol{\tau} = \mathbf{J}^{\mathrm{T}}\mathbf{F}. \tag{4.9}$$

If the generalized spring observation is used, the following formula applies:

$$\mathbf{F} = \mathbf{K}(\mathbf{X}_{\mathrm{d}} - \mathbf{X}), \tag{4.10}$$

where **K** is diagonal scalar matrix, $\mathbf{X}_d$ is the desired position of the end effector and **X** is the current position of the end effector.

If the tracking error is defined as follows:

$$\mathbf{e} = \mathbf{X}_{\mathrm{d}} - \mathbf{X}, \tag{4.11}$$

and then inserting (4.10) and (4.11) into (4.9) the following equation is derived

$$\boldsymbol{\tau} = \mathbf{K}\mathbf{J}^{\mathrm{T}}\mathbf{e}, \tag{4.12}$$

where $\boldsymbol{\tau}$ is the torque required at each joint to obtain the desired velocities of the end effector. It

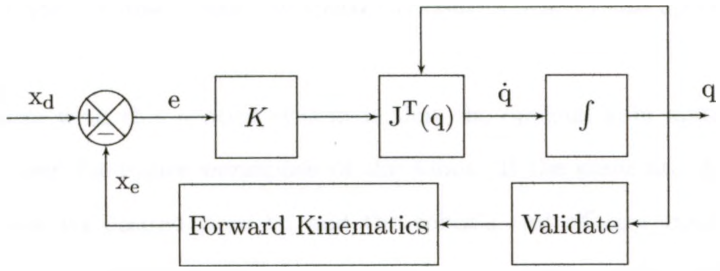


Figure 4.1: Block diagram of the Jacobian transpose control system.

can be said that the $\boldsymbol{\tau}$ is a direct indication of the desired velocities of the joints $\dot{\mathbf{q}}$ [27], and can be used as a direct substitution. Therefore, the control law is defined as follows [19,24,27]:

$$\dot{\mathbf{q}} = \mathbf{K}\mathbf{J}^{\mathrm{T}}\mathbf{e}, \tag{4.13}$$

where $\dot{\mathbf{q}}$ is the vector defining all the joint velocities of the robot. The block diagram for the (4.13) is shown in Fig. 4.1.

It should be noted that the matrix $\mathbf{K}$ in (4.13) is a diagonal matrix where the scalars along the diagonal are referred to as the gains of the system.

This control algorithm is simple in that it only requires the current Jacobian of the robot as well as the desired trajectory of the end effector to obtain the desired joint velocities. It also avoids any singularities in its workspace as it does not perform any matrix inversion. The simplicity of the equation means that the amount of computation to obtain the desired velocities is minimal compared to most other kinematic control systems.

Some advantages of this method are that it has a zero steady state error for a unit step input and it has a bounded tracking error for a given trajectory [26]. These bounds can been reduced by increasing the scalar gain matrix $\mathbf{K}$ found in (4.13). It should also be noted that the system does not have a zero steady state error for a ramp input. The steady state error for a ramp input is bounded and may be reduced by increasing the scalar gain matrix $\mathbf{K}$.

The draw back of this control system is that it is not as accurate as other kinematic control systems, such as the Jacobian inverse control system and is only suitable for slower trajectories. This is not a problem for this work as the sample rate for the user's input is 1 kHz, which is set by the 7 DOF input device, is lower than the speed the control system can update its desired joint velocities.

Another challenge with this control system is that the optimal gain matrix is not constant between robots or over the entire workspace of the robot. If the gains are chosen too large the system can overshoot its desired position and the robot's end effector might oscillate around the desired position. If the gains are too low the robot will move at an undesirably slow rate towards the desired position, making it difficult to control. This is overcome by noting that when working with most robots, only a small subset of their entire workspace is used in a single surgical simulation, therefore once a suitable set of gains is defined they can be used for the simulation without the need for recalculation.

4.4 System Gains

Each robot imported into the system needs its own set of system gains. The gains are entered manually, and should be chosen in such a way that the robot is responsive to user inputs, but is not unstable.

The most important factor in the choice of gains is the physical structure of the robot. If a link is added, removed or even changed the response of the system will change accordingly and new gains should be entered.

If optimal gains are chosen for a robot in one pose, it does not mean that the gains are optimal throughout the rest of the robot's workspace [19]. This is not a problem as the response of the system does not change noticeably over the workspace for all the robots tested.

Because the integration for the control system is performed numerically, its results are time dependant, as the integration is a multiplication of the time it took the loop to execute and the result of the loop. This means that if the control system takes a long time to complete a loop the integration results will vary from a loop that was performed much faster, if all other factors are constant. Therefore, if the gains are chosen on one computer, they might not be sufficient on another. However, for noticeable differences in the performance of the control system, between two computers, the calculation speeds of the two computers would need to vary widely.

The means of choosing the system gains for a robot is a simple process. The simulator is run with the robot being controlled by the user input device, then stopped a short time later. While the simulator is running, the user moves the input device randomly. When the simulator ends, graphs are plotted with the desired trajectory and the motion of the robot. The gains are then increased in a direction to make the response faster in that direction, or made smaller if the robot would have sudden motions in the given direction. This process is repeated until the response of the robot is satisfactory.

4.5 Joint Limitations

In the control loop, each time a new set of joint values are calculated they are checked against constraints before being applied to the robot. For example, the robot can have larger than desired velocities making the simulation move faster than an actual robot could, making it unrealistic.

The reason the joint limitations are left for the user to enter, is that different makes of robots have different architectures, and therefore different joint limits.

Therefore, the following constraints can be entered to the simulator:

- Acceleration limits

  The change in each joint's velocity is measured against the time it takes for the control system to define the new velocity of the joint. If the acceleration is found to be higher than the allowable maximum limit, the change in joint velocity is reduced to the maximum allowable acceleration.

  The maximum allowable acceleration is a single value entered, for each joint, by the user and is used throughout the joint's complete motion. It is also used for both positive and negative accelerations. This value does not take into account any inertial effects of the system, but only a numerical change in the joint velocity.

- Velocity limits

  Once the results from the control algorithm have been tested against the maximum accelerations, they are then tested to see if the resulting velocity is above the maximum velocity as specified by the user.

Once again, if the velocity is above the specified maximum the velocity is reduced to the maximum value. The value entered by the user is once again a single value, for each joint, that is used for both positive and negative velocities.

- Angle or displacement limits

  The final test of the control system results is to test to see if the resulting angle or displacement would be larger than the maximum allowable value. If this is the case, the joint will only be set to its maximum value. This will be repeated every time the control system attempts to set an value larger or smaller than allowable.

  Both maximum and minimum joint limits can be set for each joint.

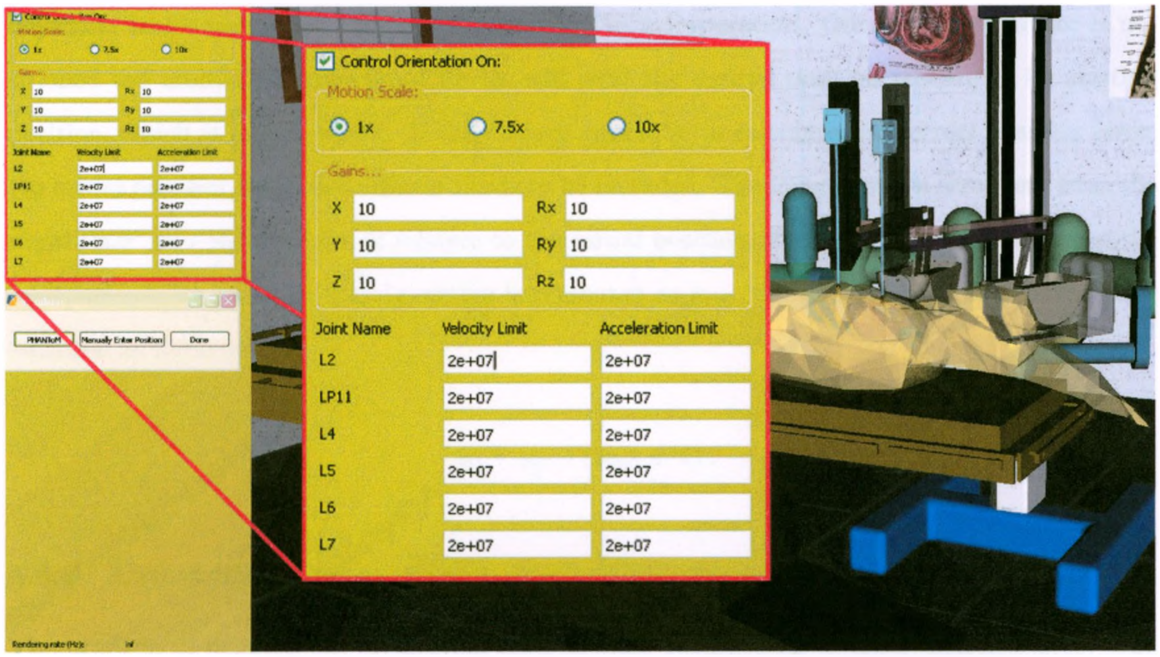
All of these constraints are set to large numbers initially, and are up to the user to modify as desired. The initial values are shown in Fig. 4.2. If these values are left to their initial large numbers, they will have no effect on the system and the robot will respond to motions based on the gains of the control system. Therefore, if a user is not specifically concerned about these limitations, they can be ignored.

4.6 Defining the Errors

A control system can be defined as an algorithm that continually attempts to reduce a given error to zero. Therefore, if the control system is to perform correctly, the correct method of obtaining the error, between the desired input and actual output of the system, should be obtained correctly. The process of calculating these errors are described here.

As the desired position is generally defined in 6 DOF, to perform complete control of the end effector, the input values for the control algorithm are specified in a $[6 \times 1]$ vector with the first 3 values being the position error and the final three being the orientation error. This error vector can be written as follows:

$$\mathbf{e} = \begin{pmatrix} \chi_e \\ \phi_e \end{pmatrix}, \tag{4.14}$$



[✓] Control Orientation On:

Motion Scale: [✓ 1x] 7.5x 10x

Gains...

X	10	Rx	10
Y	10	Ry	10
Z	10	Rz	10

Joint Name	Velocity Limit	Acceleration Limit
L2	2e+07	2e+07
LP11	2e+07	2e+07
L4	2e+07	2e+07
L5	2e+07	2e+07
L6	2e+07	2e+07
L7	2e+07	2e+07

Figure 4.2: A screen capture showing where the user has the option of adjusting the gains for the system, along with any limitations for the robot's joints.

where χ_e is the positional error and ϕ_e is the orientation error.

4.6.1 Position Error

The position error is the simpler of the two errors to obtain. This error is obtained by vector subtraction between the desired position and the current position of the end effector. This is defined as follows:

$$\chi_e = \mathbf{x}_d - \mathbf{x}, \tag{4.15}$$

where $\mathbf{x}_d$ is the desired position and $\mathbf{x}$ is the current position of the end effector.

The vectors χ_e, $\mathbf{x}_d$, and $\mathbf{x}$ are all $[3 \times 1]$ vectors and are always defined in the same Cartesian reference frame. A digram illustrating the definition of the positional error is shown in Fig. 4.3.

If the velocity error is desired, it can simply be defined in the same manner as the positional error, as long as both the desired velocity and actual velocity are defined in the same Cartesian reference frame.

Another point that is taken into account, is that the end effector can be manually positioned anywhere in its workspace before it is controlled by teleoperation. Once the end effector is positioned the multi-DOF user input device will send the desired position relative to its starting position, which is a zero vector. If the control system is to calculate the correct position error, the initial position of the end effector needs to be recorded. Then any position error sent from the multi-DOF user input device is relative to the initial position of the end effector from when the teleoperation began. The initial position is presented as $\mathbf{x}_{\mathrm{init}}$, and is added to the position error equation as follows:

$$\chi_{\mathrm{e}} = \mathbf{x}_{\mathrm{d}} - (\mathbf{x}_{\mathrm{init}} + \mathbf{x})\,. \tag{4.16}$$

4.6.2 Orientation Error

The problem of defining the orientation error is more complicated than the position error. The orientation error can be thought of as the difference between two Cartesian reference frames sharing the same origin, only rotated into different orientations. The orientation error between the two frames can be defined as the amount one frame needs to be rotated about a given axis, such that the two frames are then aligned. This error is needed in the form of a [3 × 1] vector for

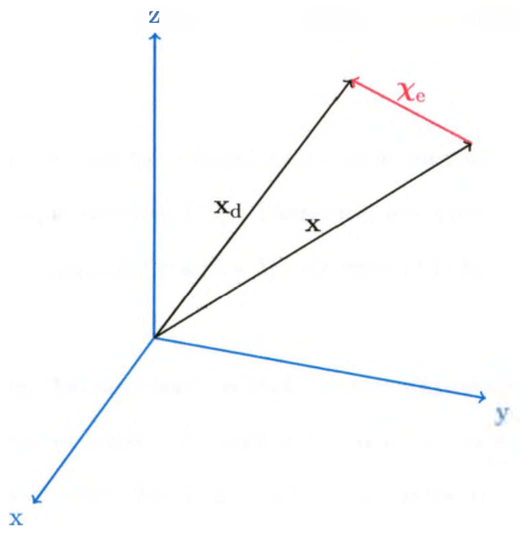


Figure 4.3: Diagram illustrating the positional error vector for the control system.

the control system, and can be calculated using one of the methods described below [28].

4.6.2.1 Methods of Representation

- Euler Angles

  Euler angles are a set of three angles that can be used to rotate a reference frame, in a specified order, to obtain a desired orientation. Euler angles are minimal representations of a specific orientation. By using the Euler angle approach the orientation error can be determined by subtracting the current Euler angles from the desired angles.

  However, it is known that Euler angles have inherent singularity points [28–30], making the use of these angles unstable and prone to errors. Therefore, if desired orientations are defined in Euler angles along a smooth trajectory, the output error angles between the previous orientation and the following orientation will not be a continuously smooth change but can have discontinuities.

  Another drawback of this approach is that unique Euler angles can not be extracted from a given rotation matrix at representation singularities, making conversions between Euler angles and rotation matrices unreliable. These representation singularities occur when the rotations of the first and last Euler angle lie along the same direction [29].

- Angle-Axis Pairs

  The orientation of a body can be defined as a single rotation around a given axis, which is known as the angle-axis theorem [19]. Therefore, an orientation can be defined by four values: θ – the rotation angle, and $\mathbf{u}$ a $[3 \times 1]$ unit vector of the axis about which the rotation occurs.

  The draw back of using this approach, is that the orientation error is not uniquely defined, for example the rotation of angle $-\theta$ about axis $-\mathbf{u}$ is the same as the rotation of θ about axis $\mathbf{u}$. This is a problem when deciding which three values, out of the four possible values, to be use for defining the error.

Another problem is when $\theta = 0$. For this angle the axis $\mathbf{u}$ is arbitrary [29] making it difficult to define three unique values to use as the error measure.

- Unit Quaternions, or Euler Parameters

  The nonuniqueness of the angle-axis values can be overcome by the unit quaternion values defined as $\mathcal{Q} = \{\eta, \boldsymbol{\epsilon}\}$, where η is the scalar part of the unit quaternion and $\boldsymbol{\epsilon} = (\boldsymbol{\epsilon}_{\mathrm{x}}, \boldsymbol{\epsilon}_{\mathrm{y}}, \boldsymbol{\epsilon}_{\mathrm{z}})^{\mathrm{T}}$ is the vector part of the unit quaternion. Unit quaternion's are always constrained by the following equation [24]:

$$\eta^2 + \boldsymbol{\epsilon}_{\mathrm{x}}^2 + \boldsymbol{\epsilon}_{\mathrm{y}}^2 + \boldsymbol{\epsilon}_{\mathrm{z}}^2 = 1. \tag{4.17}$$

  When selecting three unique values for the error measure, for the unit quaternion, the following should be noted. The unit quaternion will be $\mathcal{Q}_{\mathrm{c}} = [1, \mathbf{0}]$ if and only if the orientation error between two reference frames is zero. Therefore, the three values to be used as a measure of error from this approach is the vector part of the unit quaternion, $\boldsymbol{\epsilon}$.

  This approach seems to be the desirable approach, however it only works if the relative rotations about a given axis do not exceed 180°. This is an undesirable quality for an error measure, as motions performed by a surgeon can exceed 180° in maneuvers such as suturing.

- Rotation Matrix

  A rotation matrix is a [3 × 3] matrix that is used to rotate a reference frame from one orientation to another. A rotation matrix is an orthogonal matrix that always has a determinant of one. The entries of the matrix are always real numbers [19].

  Rotation matrices can uniquely define a 3 DOF orientation in space. This means that, if given a smooth trajectory of rotations to follow, the rotation matrix will have continuous transitions between the points on the trajectory, unlike the Euler angles approach listed above. This approach also accommodates rotations greater than 180° about a given axis. Therefore, this method is chosen to calculate the error measure.

  One restriction with this method is that the relative rotation between two successive desired points, from the input, should be small. This restriction holds for inputs from a human hand performing surgical maneuvers. If the control system is far slower than the sample rate from the input device, successive orientation errors will become relatively large making this method unstable. This is not a problem as the control system on the computer is faster than the 1 kHz sample rate of the multi-DOF input device.

4.6.2.2 Defining the Error Measure

Given two reference frames with rotations relative to the world frame as $\mathbf{R}_d$ and $\mathbf{R}_c$, then the relative rotation between the two reference frames is defined as:

$$\mathbf{R}_e = \mathbf{R}_d \mathbf{R}_c^T, \tag{4.18}$$

where $\mathbf{R}_e$ is the orientation error between the two frames.

This will hold true for all rotations defined in the same coordinate system. Therefore, if $\mathbf{R}_d$ is defined as the desired rotation matrix of the end effector, and $\mathbf{R}_c$ as the current rotation matrix of the end effector, then $\mathbf{R}_e$ is the orientation error between the current orientation and the desired orientation for the control system.

Given that a rotation matrix is composed of nine elements and only three are required for the orientation error, three unique values need to be chosen from the orientation error rotation matrix, to be used as an error measure. The three elements to be used are proposed by Luh *et al.* [31] and are described here.

Given an orientation error rotation matrix:

$$\mathbf{R}_e = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}, \tag{4.19}$$

Luh *et al.* state that the orientation error between two frames can be reduced to zero, if the following terms of $\mathbf{R}_e$ are reduced to zero:

$$\phi_e = \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} . \tag{4.20}$$

These three values are used to define the orientation error for the control algorithm.

However, there is an initial offset between the orientation of the multi-DOF user input device and the simulated robot's end effector, which means that desired orientation obtained from the user input device is not relative to the initial end effector position, but it is relative to the orientation of the world reference frame.

To account for this initial offset, all the desired rotation matrices obtained from the multi-DOF input device is rotated to void the initial offset using the following formula:

$$\mathbf{R}_d = \mathbf{R}_{init}\mathbf{R}_{input} , \tag{4.21}$$

where $\mathbf{R}_{input}$ is the rotation matrix from the user input device, and $\mathbf{R}_{init}$ is the initial orientation of the end effector.

This gives the final error rotation matrix as:

$$\mathbf{R}_e = \mathbf{R}_{init}\mathbf{R}_{input}\mathbf{R}_c^T \tag{4.22}$$

4.7 Deriving the Jacobian

Having no analytical algorithms for the kinematic chain in the simulator makes defining a Jacobian numerically a problem. This is overcome using the method defined by Angeles [32]. He stated that a Jacobian can be written as follows:

$$\mathbf{J} = \begin{pmatrix} \mathbf{j}_1 & \mathbf{j}_2 & \cdots & \mathbf{j}_n \end{pmatrix} , \tag{4.23}$$

where $\mathbf{j}_i$ is the i–th column vector of the Jacobian. Each column vector can be defined as:

$$\mathbf{j}_i = \begin{pmatrix} \mathbf{e}_i \\ \mathbf{e}_i \times \mathbf{r}_i \end{pmatrix}, \tag{4.24}$$

if the joint is revolute, or

$$\mathbf{j}_i = \begin{pmatrix} \underline{\mathbf{0}} \\ \mathbf{e}_i \end{pmatrix}, \tag{4.25}$$

if the joint is prismatic.

In (4.24) and (4.25) $\mathbf{e}_i$ is the unit vector of the axis of Joint i. $\underline{\mathbf{0}}$ is a $[3 \times 1]$ zero vector. $\mathbf{r}_i$ is the vector that points from the origin of the kinematic frame of Joint i to the origin of the reference frame of the end effector.

The resulting Jacobian is a $[6 \times n]$ matrix, with n being the number of active joints in the robot and 6 is the number of degrees of freedom that the end effector reference frame has.

4.8 Conclusion

This chapter briefly introduced control systems and named a few types that are available for controlling modern robots. The main control systems were narrowed down so that a control system could be chosen for this work. Some of the viable control systems are listed along with the reason they were not chosen.

The control system for this work was then explained, and the strengths and weaknesses of the algorithm were discussed. The methods for defining the error measure for the control system were then presented, along with the method chosen for this work.

The Jacobian transpose control system chosen for this work is able to control the robots imported into the simulator. By adjusting the gains of the control system the response and accuracy of the robot can be enhanced. The performance of this control system is tested with the results presented in Chapter 5.

Chapter 5

Validation

5.1 Introduction

When creating a teleoperated simulation environment for robots, the following key factors need to be tested to determine if the simulator operates satisfactorily. These factors are: (a) the rendering times for the display, and (b) the response of the control system. Inherent in these two factors are other factors, such as the kinematic computation times, control loop times and the times taken to transform all the bodies in the rendered scene.

The rendering time is important, in that if it takes too long, the simulator will use more time rendering the scene than performing calculations for the control system. This would make the control system at worst unstable and at best sluggish. The rendering time will be affected by many factors, such as the amount and complexity of objects being rendered or the computations used to perform the rendering. Using the VTK libraries for rendering means that the rendering method has already been defined, and it only needs to be tested to determine if the rendering speed is fast enough for this work or if it needs to be modified.

Without having the ability to directly control which objects will be added to a rendering scene, it is vital to know what limitations the rendering method will have, based on the number and type of object(s) being added to the rendering scene. If the specific numbers are known, these can be set as limitations in the simulator to keep it functioning correctly.

The response of the control system is altered by the type of robot being simulated as well as the speed of the system performing the calculations. If the robot is comprised of complicated linkage systems, it will slow down the kinematic calculations.

5.2 Computer System

To better define the performance of the simulator, the specifications of the computer system on which it is run is listed in Table 5.1. The computer used to obtain information from the multi-DOF input device is not listed here, as it does not need to be a powerful machine and does not affect the performance of the simulator. Its only task is to request positions from the input device and relay the response over the network, which can be accomplished by most current PC's.

5.3 Rendering Times

It is known that the main factors to influence the rendering times of a scene is the number of objects in the scene or the number of facets to be rendered. Increasing the number of objects in a scene simply increases the memory required to render a scene as well as increase the number of facets in the scene.

The number of facets in a scene increases the rendering times, as the rendering algorithm needs to compute the colour and the intensity of light reflected from each facet in the scene. The rendering algorithm also needs to determine which facets should be visible and which ones are hidden behind other facets. If a complex shading technique is used, then the rendering algorithm also needs to compute the colour gradient along each facet, based on its orientation in the scene.

Table 5.1: The specifications of the computer system on which the simulator was run.

Operating System	Windows XP SP2 x64
CPU	Intel Core 2 quad Q6600
Processor Speed	2.4 GHz
Memory	8 GB
Memory Type	DDR2
GPU	NVIDIA GeForce GTX 260

5.3.1 Number of Objects in a Scene

To test the effect of the number of objects in a rendering scene, nine different objects were created with the same shape but with a different number of facets for each object. The lowest number of facets for one of the objects was 2,000. This number increased by 1,000 for each consecutive object, and continued like this up to 10,000 facets on the final object.

A blank rendering scene was created, with a simple single coloured background, for the test. One of the objects was then imported into the scene. The scene was then rendered 1,000 times and the average rendering time was recorded. A second version of the same object was added to the scene and rendered 1,000 times again. This was repeated until the scene had 100 of the same object.

The rendering times were calculated using one of the VTK libraries built-in functions. As a scene is rendered, the time taken to perform the rendering is automatically calculated. For this test, these render times were merely recorded.

This process was repeated for each of the nine objects created. These rendering times are shown in Fig. 5.1. Some spikes can be seen in the figure, but these were caused by other processes running in the background on the computer at the time of the test.

The test only went up to 100 objects as it is not likely that more than 100 objects would be used in the simulator at one time. The complete scene shown in Fig. 2.1 is comprised of 70 objects and is on the limit of the rendering time.

Fig. 5.1 shows that the number of objects in the scene does affect the rendering times, but not as much as the collective number of facets in the scene, as shown below.

5.3.2 Number of Facets in a Scene

When rendering a scene, the light reflected from every facet to the camera needs to be calculated as each facet can have a different surface normal from the surrounding facets. This means that the more facets a scene has, the more calculations are needed to render it in the scene. To test the effect of the number of facets on rendering times the following test was performed.

Two separate objects were created, one with a model of a human rib cage and the other with

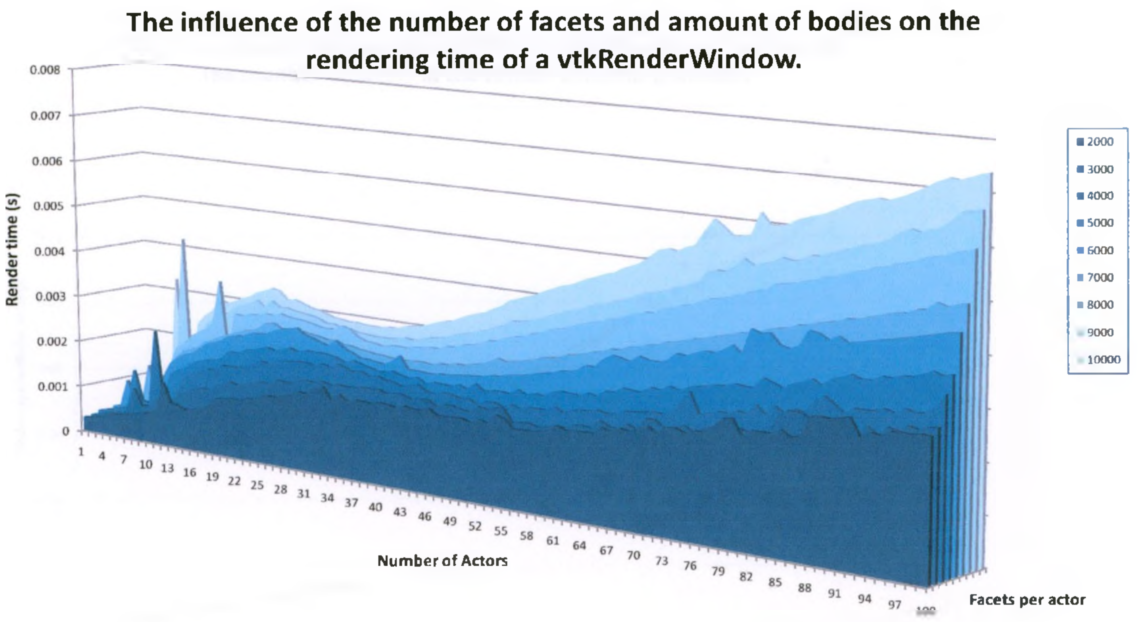


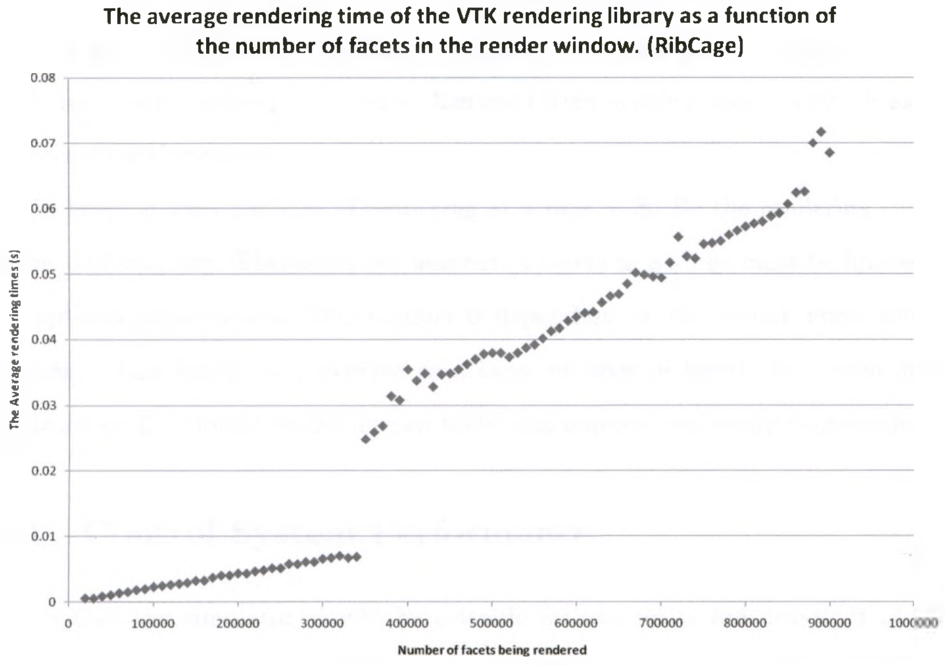
Figure 5.1: A graph representing the change in the rendering time with the number of bodies added to a scene and the number of facets on each body.

a simple sphere primitive. The rib cage model initially had 950,000 facets. This model was then subsampled down by 5,000 facets creating a new object. This process was repeated until the final model of the rib cage had 20,000 facets. This process created 186 models with a different number of facets in each.

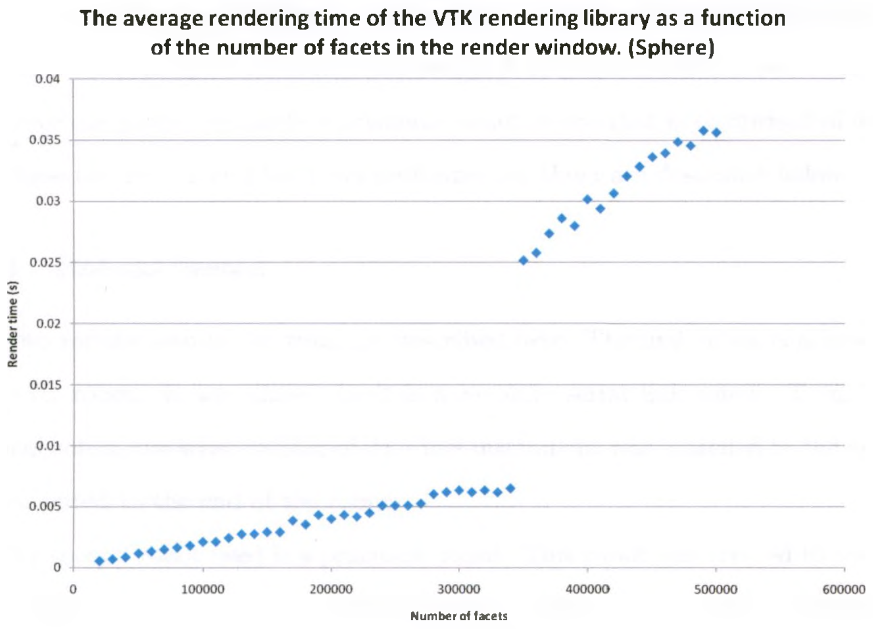
The model of the sphere primitive initially had 500,000 facets and was also subsampled by 5,000 facets, creating new models, until the final model only had 20,000 facets.

A rendering scene was setup with a blank background, a single camera and a single light source in the scene. One of the objects of the rib cage was then imported into the scene. The scene was rendered 1,000 times and the average of these times was recorded. This process was then repeated with every object created of the rib cage and the sphere primitive, each being rendered in its own scene. The results of this test are shown in Fig. 5.2.

As can be seen in Fig. 5.2 the rendering times are relatively fast until the scene has 350,000 facets or more. From that point on the rendering time increases rapidly. These graphs also show that it does not matter how complex a shape is, the rendering time is affected by the number of facets and not the shape of the object.



(a)



(b)

Figure 5.2: Two graphs showing the comparison between the number of facets in a scene and the rendering time for the scene using (a) a rib cage model and (b) a sphere primitive.

The reason there is a large jump in the rendering times in Fig. 5.2 is due to the limited amount of memory available on the GPU for rendering. As soon as all the memory is consumed on the GPU the excess memory required is then used from another source which is slower than using the on-board GPU memory.

In keeping with the idea of rendering at a rate of 30 Hz the rendering times need to be kept below 0.03 seconds. Therefore, the number of facets in a scene must be limited to 350,000 facets, for optimal performance. This number is dependant on the system upon which the simulator is running. This might be perceived as a large number of facets, but when dealing with natural contours such as found in the human body this number can easily be exceeded.

5.4 Control System Performance

Given that the simulator should be suitable for any serial link robot, it is difficult to test the control system accurately. Therefore, two different robots were created, on which to run tests. These tests were done to determine the general performance of the control system. The two robots created are a prismatic and a revolute robot. A revolute robot is a serial link robot comprised of only revolute joints. Similarly a prismatic robot is one that is comprised of only prismatic joints.

These two robots and the tests performed on them are described below.

5.4.1 Robots Tested

The two robots used in the tests are described here. The first robot is a model of the Mitsubishi PA10-7C robot. It was chosen as it is a revolute serial link robot. To make it perform like a surgical robot, the wrist section of da Vinci instrument was attached to the end of a long thin rod and mounted to the end of the robot.

The second robot used is a prismatic robot. This robot was created to test the control system on prismatic robots and is not based on any real robot. The robot is made up of 10 prismatic joints and a revolute joint for the instrument shaft with a revolute wrist attached to it. The wrist partition of a da Vinci instrument was also used as the wrist for this robot.

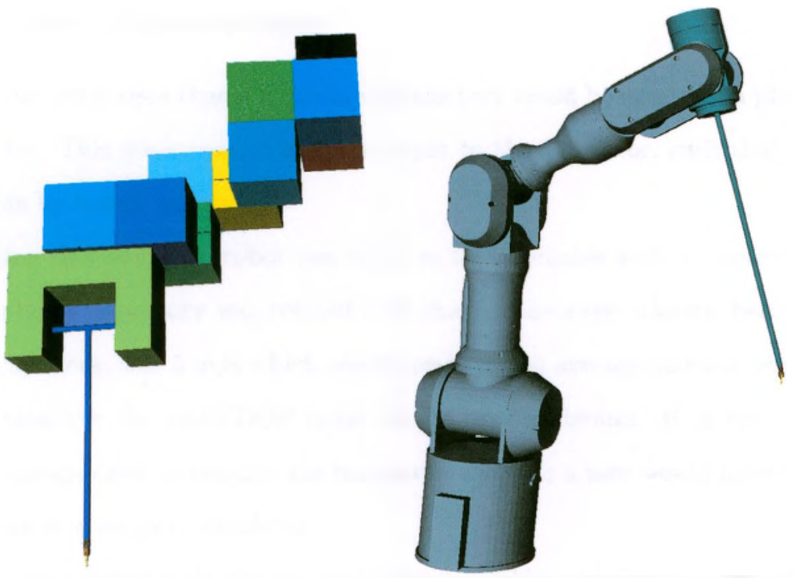

(a) (b)

Figure 5.3: A picture of the two robots used to test the control system with (a) being the prismatic robot and (b) the revolute robot.

5.4.2 The Tests Performed

Three tests were chosen to measure the general performance of the control system, and they are described as follows:

- Measuring the response of the robot with a predefined smooth sinusoidal input.
- Measure the response of the robot with a random smooth input, such as the input from the multi-DOF input device.
- The final test is only for the model created of the da Vinci robot. A preset trajectory was given to the real da Vinci robot and the pose of the robot was measured along these points. The same trajectory was given to the simulated robot to check that the simulated motion was similar to the motion of the real robot. It should be noted that the model of the da Vinci was created from measurements taken of the real robot for this work. This is covered in Appendix B.

5.4.2.1 Controlled Trajectory Input

The simulator was setup such that a predefined trajectory could be entered, in place of the multi-DOF input device. This gives control over the input to the simulator, such that the response of the simulator can be tested accurately.

Before running this test, each robot was setup in the simulator with its optimal gains already calculated. The input trajectory was created such that the average velocity, between consecutive points on the trajectory, was 5 m/s which corresponds to the average speed a user would be able to accurately maneuver the multi-DOF input device over a distance of 16 cm [33]. This speed was used as it corresponded to roughly the maximum distance a user would have to move the end effector of a robot in a surgical simulator.

A trajectory was created such that the end effector would move about a point with a sinusoidal input for all three axes. The orientation of the end effector tip was kept stationary throughout the trajectory. The reason for choosing a sinusoidal trajectory is that it engages motion in all direction with continuously changing velocities.

5.4.2.2 Manual User Inputs

For the manual user input test, the multi-DOF input device was moved by hand in a circular fashion, such that the end effector would move in all three axes of the Cartesian coordinate system. In this test the end effector orientation constantly changed, based on the orientation of the multi-DOF input device. The user did not follow an exact path, as this tested whether the system could track an undefined trajectory.

The speed of the multi-DOF input device in this test is much faster than the controlled motion of a surgical simulation.

5.4.3 Results

The following results were obtained from the tests described above. In each of the results the multi-DOF input device would read zero while the user had already started moving it around. After a delay it would send its adjusted position, causing a rapid change in the desired location. This procedure was not corrected to illustrate how the control system would converge to the desired location, rather than follow the trajectory from begin to end. The results from the revolute robot with controlled trajectory input can be seen in Fig. 5.6. The results for the same robot, but with the trajectory from the multi-DOF user input, can be seen in Fig. 5.7. The corresponding trajectory and motions for these errors are shown in Fig. 5.4 and Fig. 5.5 respectively.

These results show that the control system does follow the desired trajectory but does have a lag between the desired input and position of the robot. For the controlled input trajectory test, the error stayed below 5 mm in all three Cartesian directions. The hand input results has higher magnitudes of error, which were created by the erratic motion of the human hand. The control system did bring these errors down to an acceptable level within a time frame of about 30 ms.

The results for the controlled trajectory input for the prismatic robot are shown in Fig. 5.10, and the results for manual input, for the same robot, are shown in Fig. 5.11. The corresponding trajectory and motions for these errors are shown in Fig. 5.8 and Fig. 5.9 respectively.

Once again for this robot, there are noticeable errors in the two figures. These errors are larger than the errors for the revolute robot. The increase in magnitude for these errors can be attributed to the extra number of links in the robot, which slowed down the kinematic calculations and the control system. During the test there were noticeable errors, but these errors were not large enough to make the user think the robot was lagging and that it could not be controlled properly.

The hand input test was repeated on a da Vinci active section with a 1:1 ratio between the user input and the desired position for the robot. These results are shown in Fig. 5.14. With the erratic hand motions, the da Vinci did not track the input motion as desired, but did follow the trajectory with a noticeable lag. The scaled ratio was changed to 5:1, similar to the scaling used on the real da Vinci robot, and the test was run again. These results are shown in Fig. 5.15. The da Vinci model followed this input closely. The error in these simulations are attributed to the complex kinematic of the active section. The corresponding trajectory and motions for these errors are shown in Fig. 5.12 and Fig. 5.13 respectively.

5.5 Real World Comparison

To compare the way the simulator moves a robot to the motion of the real robot, the following test was run. The real robot was moved through a given set of points, and the joint angles of the robot were measured at each point. The simulated robot was moved through the same set of points, recording the joint angles for comparison.

To make sure the simulator and the robot started in the same configuration a guide was built to position the real robot at the start of the test. This guide is shown in Fig. 5.16 along with the same setup in the simulator. The base of the guide has a pattern of holes. The da Vinci was moved around placing the end effector into the holes. The holes were numbered so that the motions could be recorded.

Absolute angles from each joint could not be obtained from the da Vinci, so the joint angles at the beginning of the test were taken as zero and all the subsequent angles were relative to the first set of angles. The joint angle of the shaft of the surgical tool could not be measured and was ignored. For the end effector of the simulated robot and the real robot to be in the same position, with 5 of the 6 joints having the same angle, then the angle of the instruments shaft does not matter. If the angle between the real and the simulated robot differs by 180°, it will not matter as this will not cause collisions.

The results from the simulated and real experiment are shown in Table 5.2. This test was repeated with several different patterns and it was found that the motion of the simulated robot match up to the motion of the da Vinci.

The motions did not compare when the distances between two given points were too large. If

Table 5.2: Comparison between the real da Vinci and a simulated version.

	θ_1		θ_2		d_3		θ_5		θ_6	
pos	real	simulated	real	simulated	real	simulated	real	simulated	real	simulated
1	0	0	0	0	0	0	0	0	0	0
3	0	1.6	40	39.04	33.61	36.17	30	32.57	20	23.78
11	-35	-38.64	0	-1.14	35.72	36.54	-20	-21.24	30	33.35
15	0	0.5	-35	-35.86	39.91	36.36	-15	-16.05	-10	-10.57
23	35	38.35	0	1.07	35.7	35.37	20	21.77	-25	-32.42
1	0	0	0	0	0	0	0	0	0	0

the simulated da Vinci had a joint reach its limit, then it would behave differently from the real da Vinci.

5.6 Conclusion

In order to have smooth motions in the simulator, the number of facets in the render scene should be kept as low as possible. This can be accomplished by making models for the simulator with sharp edges rather than round edges, which would require less facets in the STL file. Also, models of human body parts should be subsampled to as low a resolution as possible, while trying to keep it as realistic as possible.

Even if a large group of facets fall on the same plane and have the same surface normals, the rendering algorithm will calculate the light being reflected off of each facet. Therefore, the number of facets on an object should be reduced and made as large as possible. Reducing the number of bodies in a scene will also speed up he rendering time.

The values tested to determine a suitable rendering time are not the only factors to influence the rendering time of a scene, but are the major contributors to slowing the process down.

It can also be seen in Fig. 5.10 that the response of the prismatic robot is not as accurate as with the PA10 robot. This is because the prismatic robot has more joints in its kinematic chain making the control loop calculations slower.

The control system did exhibit noticeable errors, but did manage to reduce these errors in times that were generally faster than the human eye could discern. As this project is for aiding in preoperative planning research, and only the kinematic reach is what is desired, if the control system can control the robot faster than the human eye can discern, it will be sufficient.

Also, the control system is not a finely tuned control system for a specific robot. Therefore, it will not perform as well as some other control systems. The control system did control the tested robots to an acceptable level, for the purpose of this simulator. If a faster response is desired for a specific robot a different control system should be implemented. This control system is only suitable for slower trajectories and should suffice for the general controlled motions a surgeon would use in a surgical procedure.

The results of these tests are dependant on the hardware used for running the simulator. If a different system is used, these results would change and would need to be checked again to make sure that the system will perform optimally.

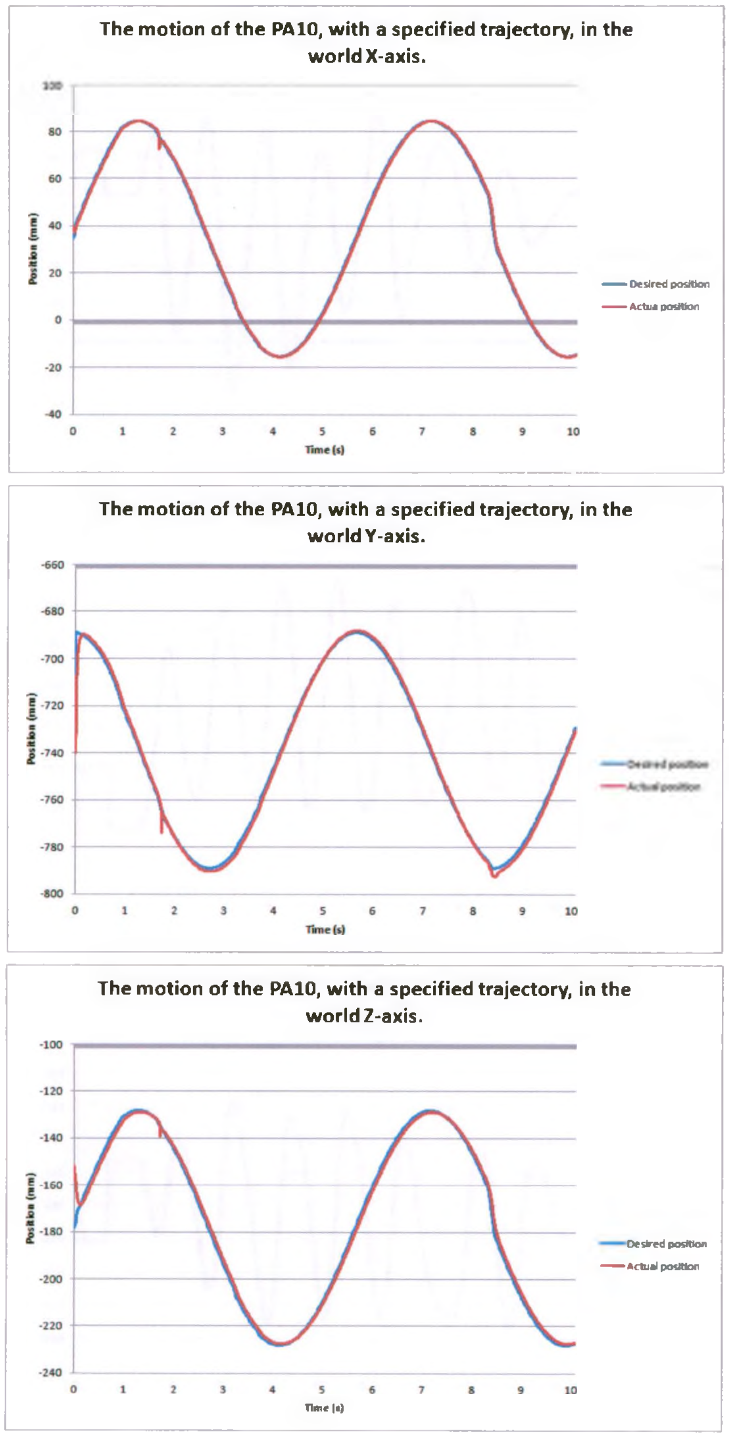


Figure 5.4: Results from the controlled trajectory input for the PA10.

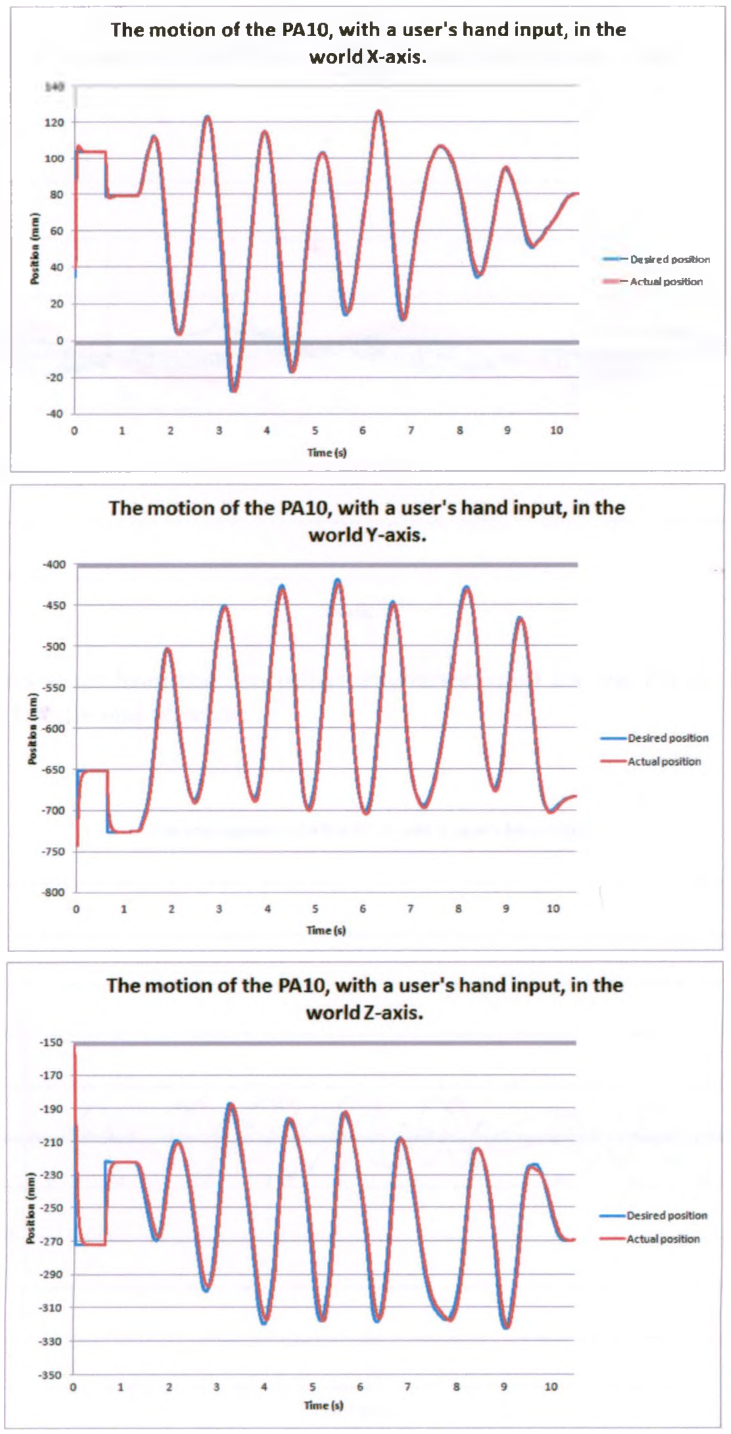


Figure 5.5: Results from the hand input for the PA10.

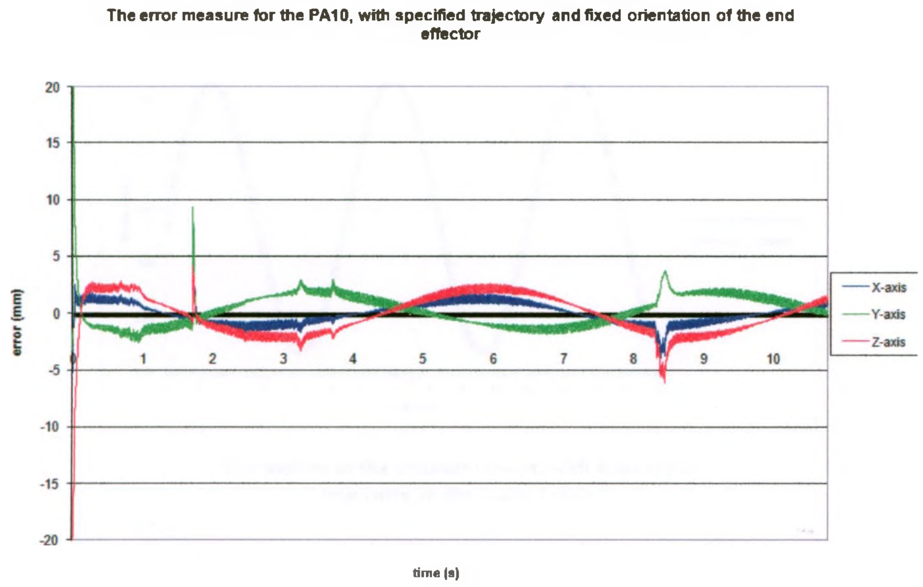


Figure 5.6: Error measure from the controlled trajectory input for the PA10, with a fixed orientation for the end effector.

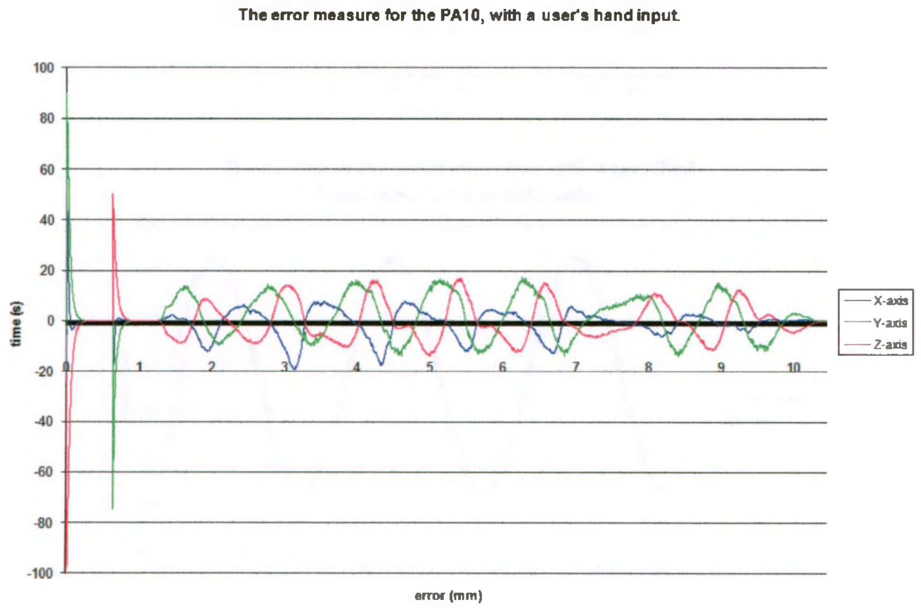


Figure 5.7: Error measure from the hand input experiment for the PA10.

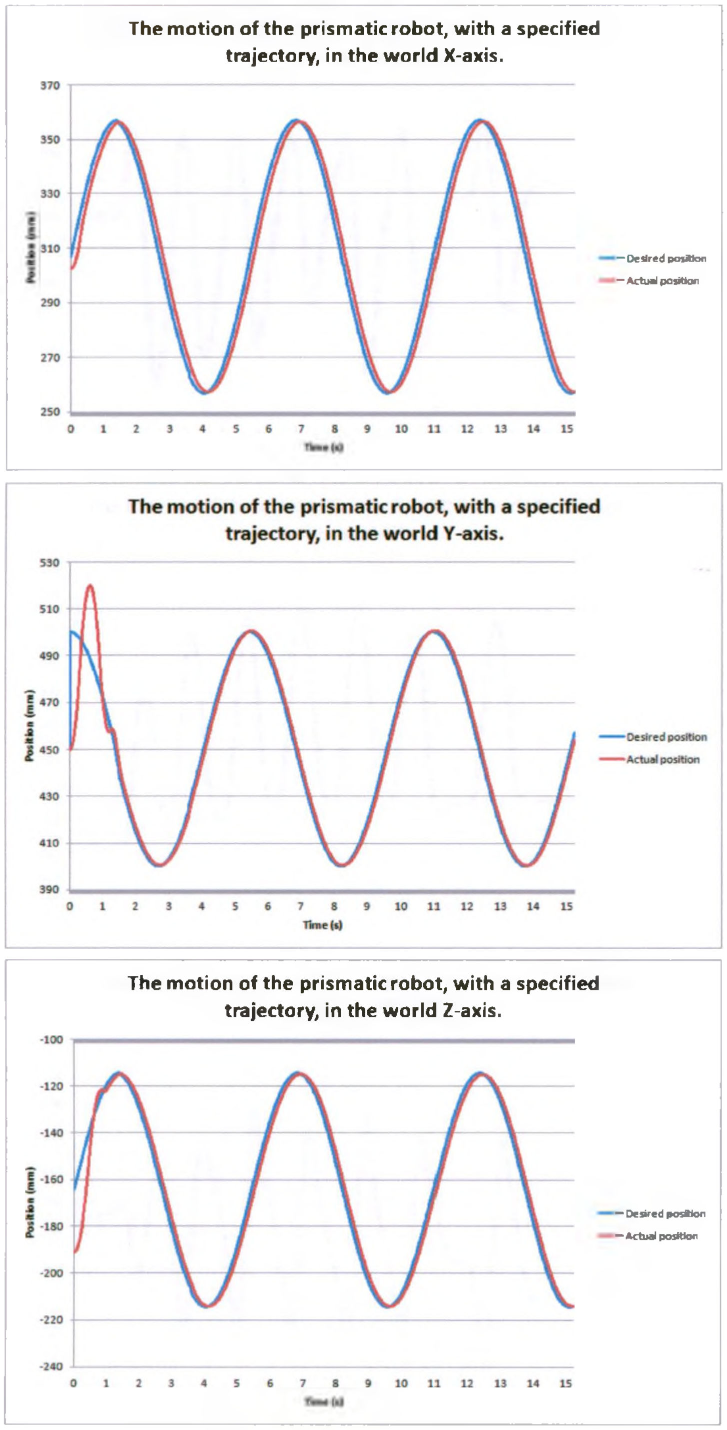
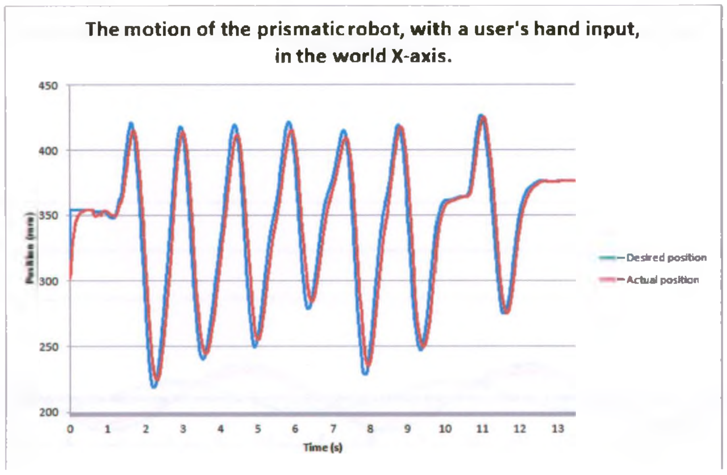
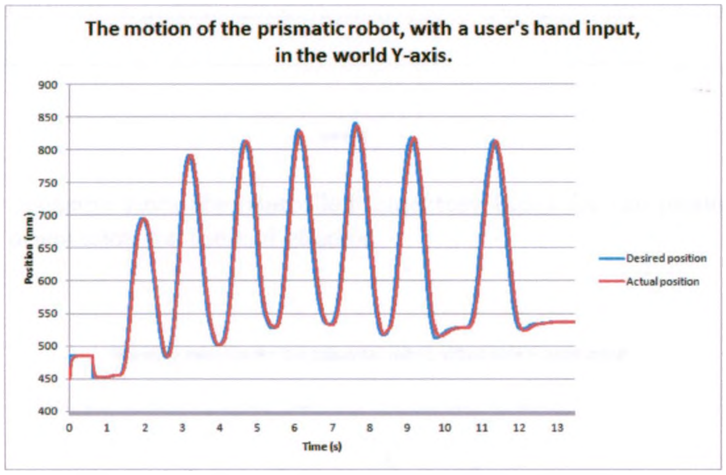


Figure 5.8: Results from the controlled trajectory input for the prismatic robot.





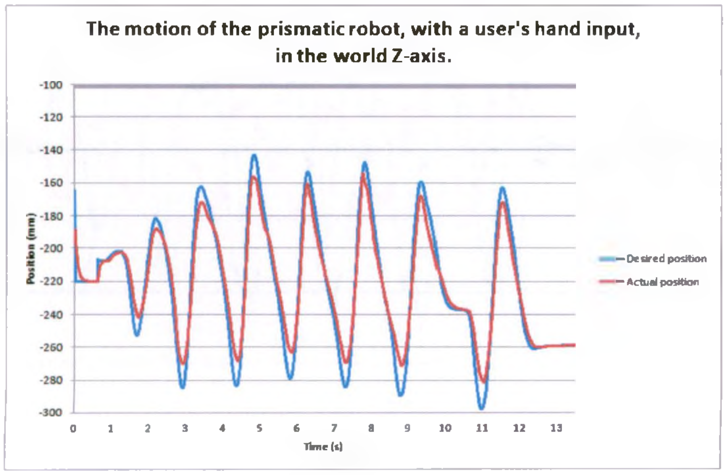


Figure 5.9: Results from the hand input for the prismatic robot.

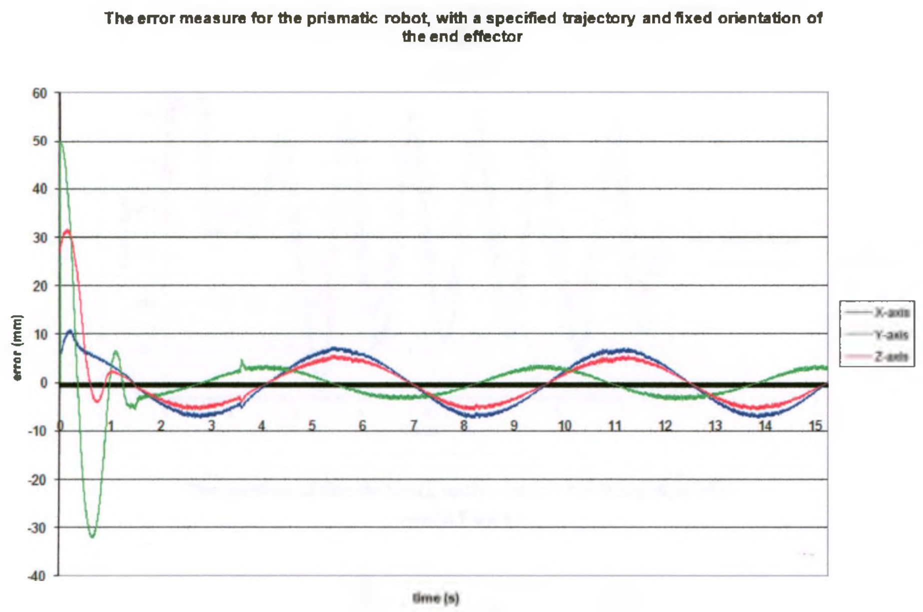


Figure 5.10: Error measure from the controlled trajectory input for the prismatic robot, with a fixed orientation for the end effector.

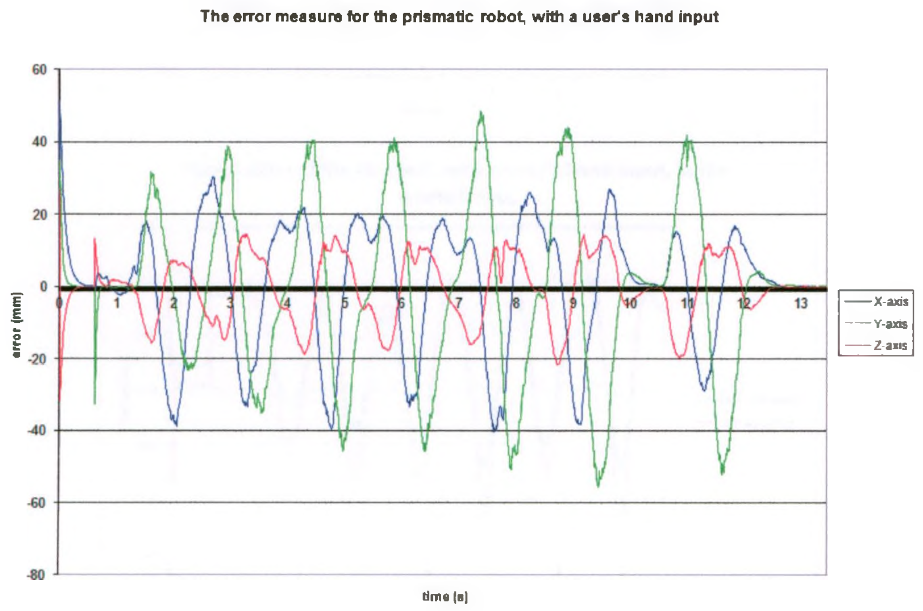
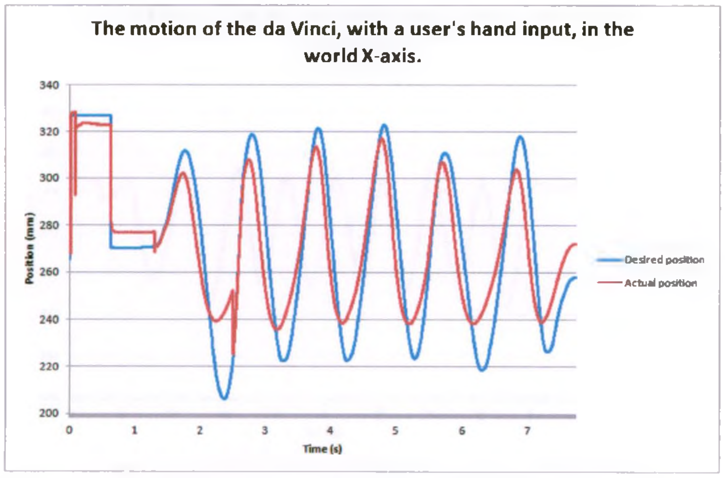
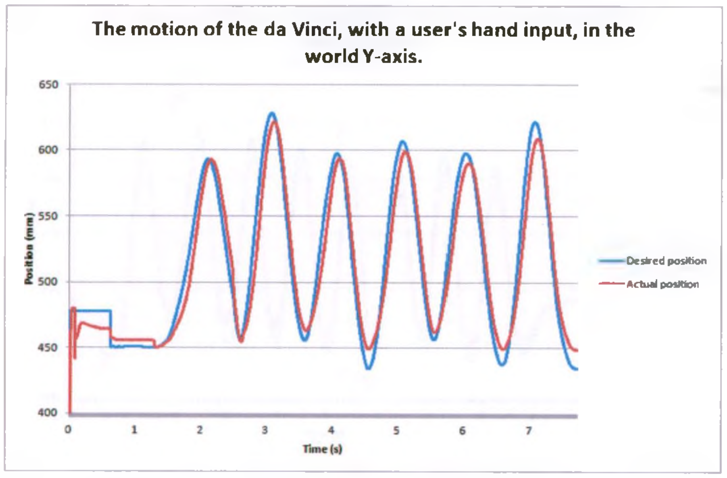


Figure 5.11: Error measure from the hand input experiment for the prismatic robot.





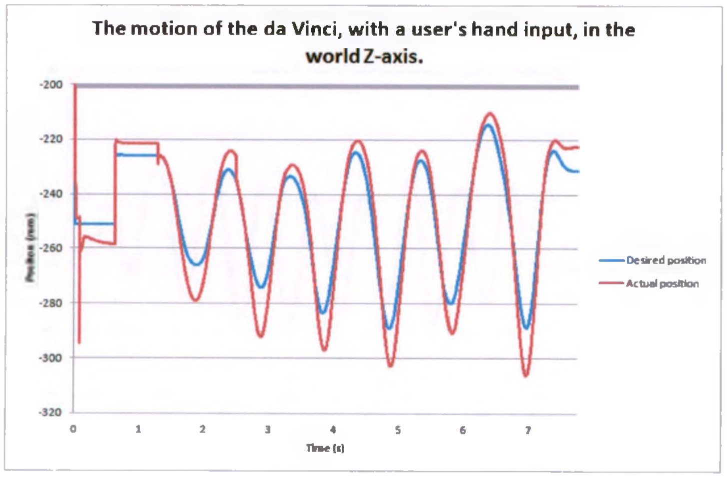


Figure 5.12: Results from the hand input for the da Vinci robot.

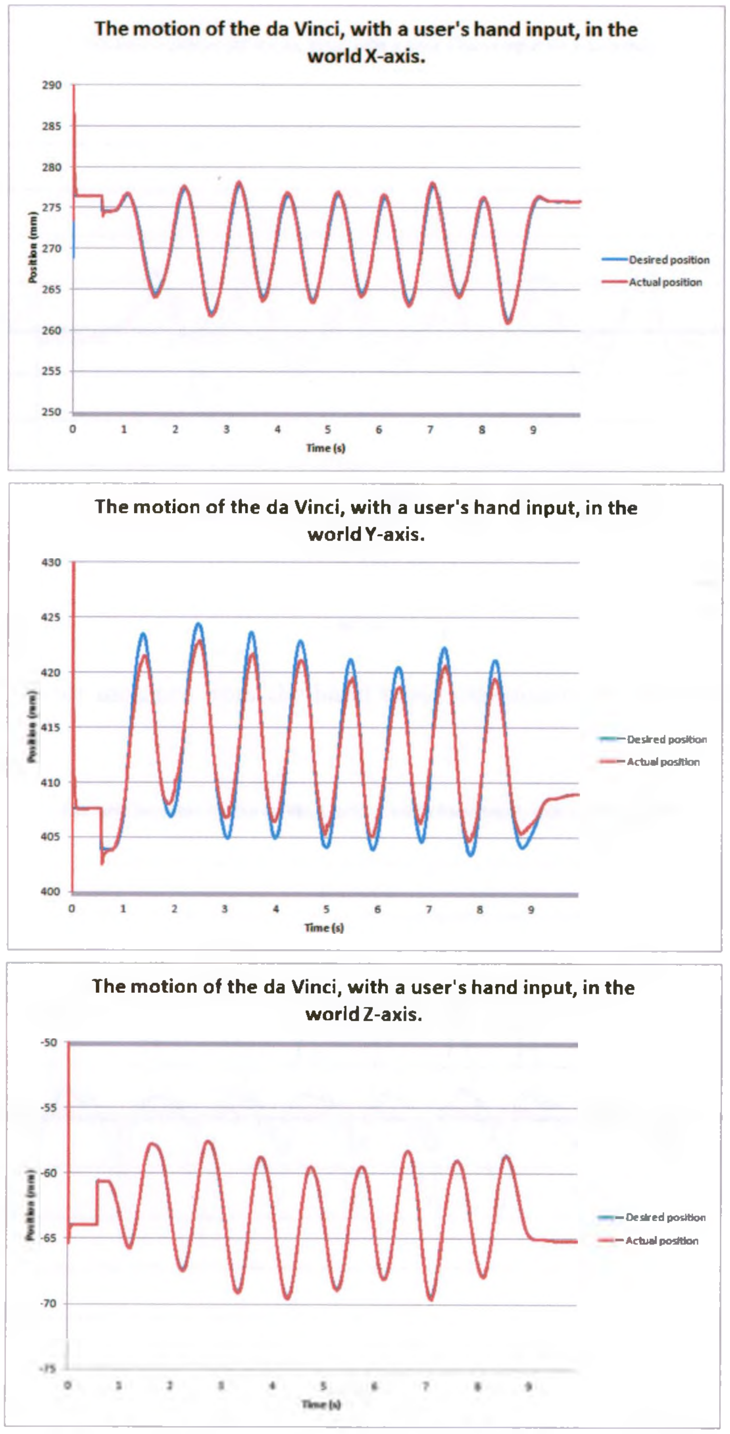


Figure 5.13: Results from the hand input for the da Vinci robot, where the input is scaled similar to the real robot..

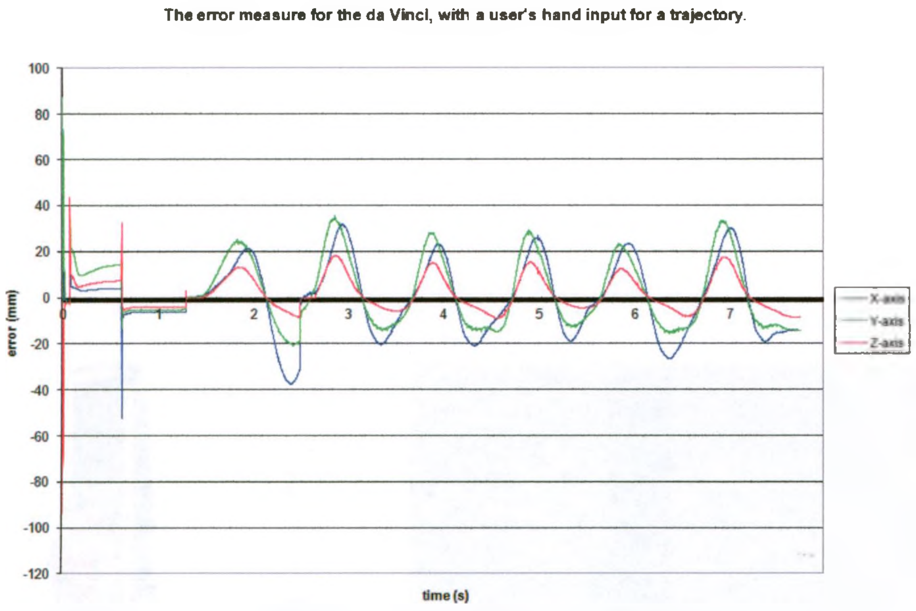


Figure 5.14: Error measure from the hand input experiment for the da Vinci robot.

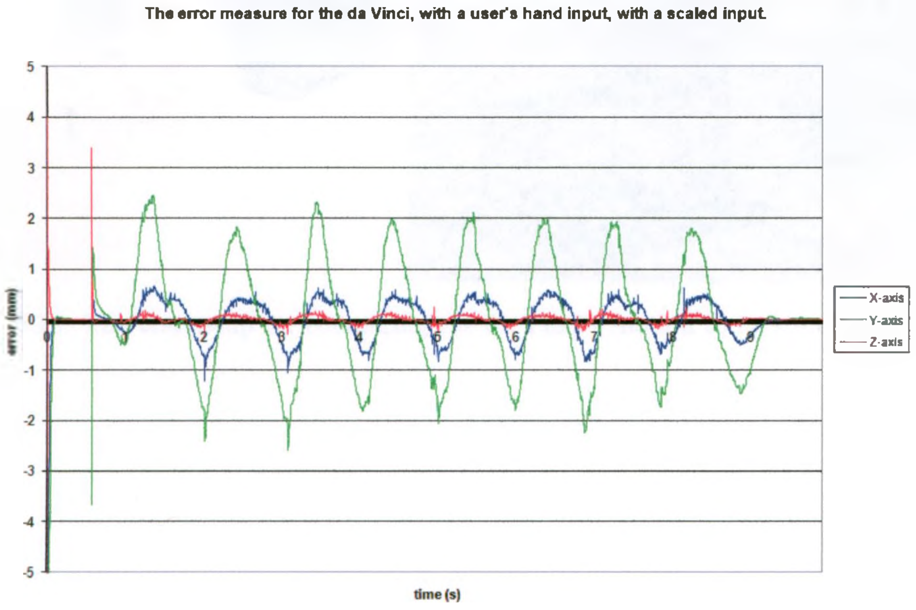


Figure 5.15: Error measure from the hand input experiment for the da Vinci robot, where the input is scaled similar to the real robot.

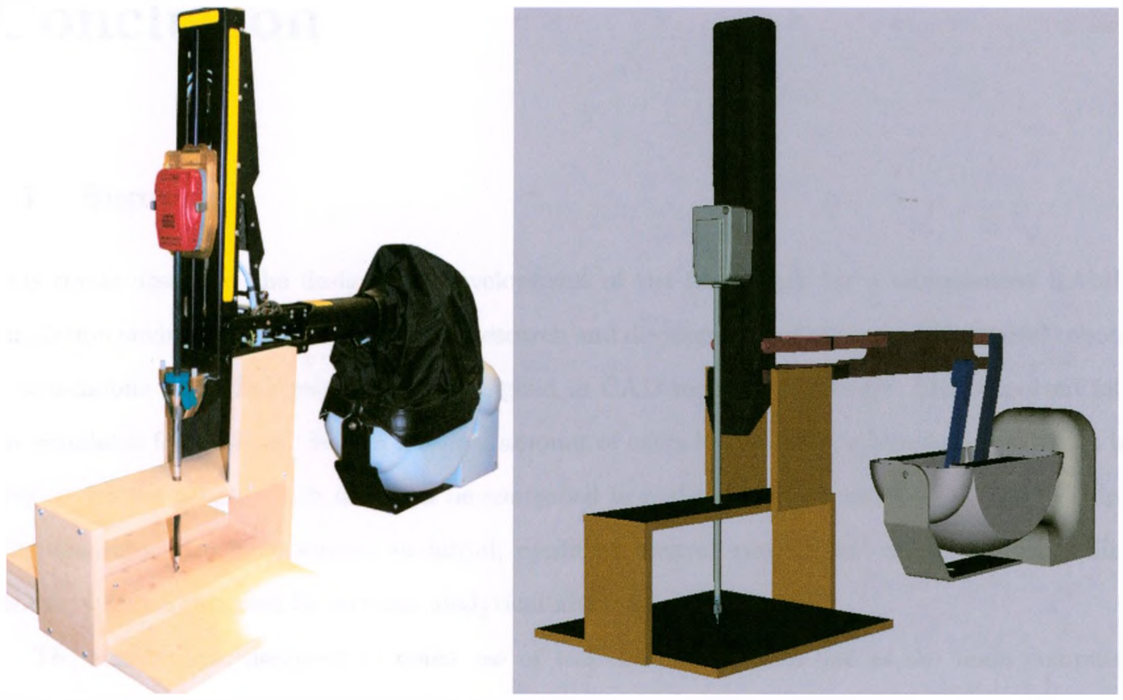


Figure 5.16: The test setup for comparing the da Vinci's motion to its simulated motion.

Chapter 6

Conclusion

6.1 Summary

This thesis describes the design and development of the framework for a teleoperated RAMIS simulation environment, for assistance in research and development of teleoperated surgical robots. A non-mobile serial link robot can be designed in CAD modelling software, then exported into the simulator for analysis. With a minimal amount of extra information, a kinematic chain can be created for the robot, which can then be controlled in real-time from a multi-DOF input device. The simulator also incorporates an initial, modular, control system that allows for controlling robots, without the need for explicit analytical kinematic algorithms.

The simulator is designed to make use of two computers, with one as the main computer and the second as a client computer, which performs the task of the master console. The main computer performs all of the simulators calculations and performs all the rendering. The client computer connects to the multi-DOF user input device, sending the information from the user input device to the main computer over a network connection. The network communication protocol is a simple ad-hoc protocol created for the simulator. The ad-hoc protocol was created to speed up the development of the simulator. The multi-DOF user input device is a PHANToM premium haptic input device. Although the PHANToM premium did have the option for haptic feedback, haptics was not used.

This work shows that the Jacobian transpose control system can successfully track a person's hand motion from the user input device, for various robots. With a scaled input, similar to the scale found on the da Vinci, the simulated da Vinci model can track a persons hand motions. This work also provides some constraints for models to be imported into the simulator, that should be taken into account for real-time rendering.

6.2 Concluding Remarks

Some RAMIS simulators have already been created by research groups for simulating the da Vinci or the Zeus system, but these simulators were each created specifically for a single robot. These simulators would need to be modified if a newer version of the robot is released, or if a different robot needs to be simulated.

Although this work only shows the connection of a single user input device, a second one could be added to the system by connecting it to the client computer, with a slight modification to the network protocol.

The software written for this work provides a base for performing the kinematic calculations for a serial link robot, without any analytical kinematic equations. If any further work is required, it can be created on top of the work already completed, without the need to modify any of the low-level kinematic algorithms.

Although the simulator performs well, it is not a finished product. As with the development of all software, there are many more features that can be added on to make the system function better and provide a more complete package. Some of the recommended features that could not be added on, due to time constraints, are listed below.

6.3 Suggestions for Future Work

The suggestions below will complement the work already completed. These suggestions are classified into categories, with each category adding to a different section of the simulator.

6.3.1 Overall Simulator Performance Suggestions

The first suggestion for the simulator would be, to create multiple control systems. This would allow the user to decide which control system to use for a robot in a scene. This will give the option, to more knowledgable users, to define analytical equations for the robot if desired, allowing for a more robust control system, with possibly faster control calculations.

Another suggestion for the simulator would be to incorporate a form of hand tremor reduction, as found on the da Vinci. This could be done by looking at sudden erratic motion from the input device and filtering it down to reduce the tremors. Another option could be to take the weighted average of all the desired positions from the input device to smooth out erratic motions.

6.3.2 Robot Motion Suggestions

The first recommendation for the control system, created for this work, would be to add more closed-loop mechanism solvers to complement the four-bar mechanism solver. Closed-loop mechanism solvers were not the focus of this work, so a simple method dealing with specific closed-loop mechanisms was developed. This method enabled specific mechanism solvers to be created and added to the code as needed. To allow for the simulation of the da Vinci, a four-bar mechanism solver was created and added to the software.

With the Jacobian transpose control system not explicitly avoiding joint limits, the joint can get 'stuck' on its limit during a simulation. Then as the simulation continues and the joint needs to move away from its limit, it would remain 'stuck' for a number of the control loop iterations, and would then rapidly move away from the limit, causing errors in the tracking systems. This is a known side-effect of the Jacobian transpose control system, which is generally overcome by imposing extra constraints to the system [26]. This would add extra complexity to the user of the simulator, which is undesirable. A method of avoiding these limits without the need for input from the user can be developed to prevent this from happening.

Collision detection and obstacle avoidance can be added to the simulator. This would allow for automatic monitoring of any collisions between objects in a simulation. This would be a vital addition for when the simulator is being used to test preoperative planning algorithms. Currently the simulator does not check for collisions. If two bodies occupy the same space in the scene, the two bodies merely move into each other and there is no notification of the collision, and it is up to the user to visually monitor for any collisions. Obstacle avoidance can be used as a constraint in the control system if the robot has redundant links.

For robots that do not have a set of mechanical linkages to create an RCM for surgery, software based control can be used to create the RCM. A robot that uses software control for an RCM has been developed and is called the MicroSurge [34]. This leads to the need for adding a method for creating an RCM on a robot using a software approach rather than a mechanical design.

Currently, one of the hardest parts of importing a robot into the simulator is choosing the optimal gains for the control system. To speed this process up, an auto-tuning algorithm can be developed to find the optimal gains for any robot added to the simulator. This algorithm could be run in the background, intermittently, while a simulation is performed to make sure the robot always has optimal gains throughout its workspace.

Another point to consider is adding an algorithm that can optimally solve any given closed loop chain. This is a complex task to perform, without any analytical equations for the kinematic chains to be imported into the simulator, while still maintaining real-time motion calculations. This will eliminate the need to create multiple closed loop algorithms for various linkage mechanisms, and remove the need for the user to define a closed loop chain and to select which solver to use.

6.3.3 Teleoperation Suggestions

To add more value to the teleoperation of the simulator, tissue interaction with haptic feedback could be added to the simulator. This is a complicated task. Not only in the calculations of force feedback, but also maintaining real-time teleoperation.

If a third haptic device is added to the system, this device could control a visual cue in the render scene and therefore be used as a training tool. In such a case, a less experienced surgeon can be using two multi-DOF input devices to practice a procedure, while a senior surgeon could manipulate the third input device to point out features in the scene. This would be similar to the training techniques available in the newest da Vinci Si robots.

Another suggestion for the simulator would be to add a 3-D output display to the simulator.

This could add to the realism of the simulator, and make it easier to accurately position the end effector of a robot as compared to operations performed using a 2-D output display.

The final suggestion would be to add an extra arm for an endoscope to add an internal view to the simulation.

References

[1] M. Lum, D. Trimble, J. Rosen, K. Fodero, H. King, G. Sankaranarayanan, J. Dosher, R. Leuschke, B. Martin-Anderson, M. Sinanan, and B. Hannaford, “Multidisciplinary approach for developing a new minimally invasive surgical robotic system,” in *The First IEEE / RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, Tuscany, Italy*, pp. 841–846, February 20-22, 2006.

[2] A. Lanfranco, A. E. Castellanos, and J. Desai, “Robotic surgery: a current perspective,” *Annals of Surgery*, vol. 239, pp. 14–23, 2004.

[3] G. Guthart and J. Salisbury, “The intuitive telesurgery system: Overview and application,” in *Proceedings of the IEEE International Conference on Robotics and Automation, California, USA*, pp. 618–621, April 24-28, 2000.

[4] A. J. Mahdhani, *Design of Teleoperated Surgical Instruments.* PhD thesis, MIT, 1997. pp. 20-25.

[5] M. Zhou, J. Perreault, S. D. Schwaitzberg, and C. G. L. Cao, “Force perception threshold varies with experience in minimally invasive surgery,” in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Quebec, Canada*, pp. 2227–2232, October 7-10, 2007.

[6] F. Corcione, C. Esposito, D. Cuccurull, A. Settembre, N. Miranda, F. Amat, F. Pirozzi, and P. Caiazzo, “Advantages and limits of robot-assisted laparoscopic surgery,” *Surgical Endoscopy*, vol. 19, pp. 117–119, January 2005.

[7] Intuitive Surgical Inc., “Intuitive surgical and computer motion close merger.” `http://investor.intuitivesurgical.com`, November 2005.

[8] T. Vassiliades, “Technical aids to performing thoracoscopic robotically-assisted internal mammary artery harvesting,” *Heart Surgery Forum*, vol. 5, pp. 119–124, June 27, 2001.

[9] A. L. Trejos, R. Patel, I. Ross, and B. Kiaii, “Optimizing port placement for robot-assisted minimally invasive cardiac surgery,” *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 3, pp. 355–364, November 14, 2007.

[10] H. Azimian, J. Breetzke, A. Trejos, R. Patel, M. Naish, B. Kiaii, T. Peters, J. Moore, and C. Wedlake, “Preoperative planning of robotics-assisted minimally invasive coronary artery bypass grafting,” in *Proceedings of IEEE International Conference on Robotics and Automation, Alaska, USA*, pp. 1548–1553, May 3-7, 2010.

[11] M. Hayashibe, N. Suzuki, M. Hashizume, Y. Kakeji, K. Konishi, S. Suzuki, and A. Hattori, "Preoperative planning system for surgical robotics setup with kinematics and haptics," *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 1, pp. 76–85, November 17, 2005.

[12] S. Cotin, H. Delingette, and N. Ayache, "A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation," *The Visual Computer*, vol. 16, pp. 437–452, December 2000.

[13] Surgical Science Inc., "Surgical science medical training simulator." `http://www.surgical-science.com/`, November 2010.

[14] iSurgicals, "Laparoscopy simulator." `http://www.isurgicals.com/`, March 2011.

[15] 3-Dmed, "Laparoscopic - minimally invasive training system." `http://www.3-dmed.com/`, March 2011.

[16] M. Hayashibe, N. Suzuki, and M. Hashimuze, "Robotic surgery setup simulation with the integration of inverse-kinematics computation and medical imaging," *Computer Methods and Programs in Biomedicine*, vol. 83, pp. 63–72, July 2006.

[17] L. wah Sun, F. V. Meer, Y. Bailly, and C. K. Yeung, "Design and development of a da vinci surgical system simulator," in *Proceedings of IEEE International Conference on Mechatronics and Automation, Heilongjiang, China*, pp. 1050–1055, August 5-7, 2007.

[18] Intuitive Surgical, Inc., 950 Kifer Road, Sunnyvale, CA, USA, *da Vinci surgical system user manual*, 4th ed., April 2004.

[19] J. J. Craig, *Introduction to Robotics: Mechanics and Control.* Prentice Hall, Upper Saddle River, New Jersey, USA., 3rd ed., August 6, 2004.

[20] BWH - Harvard, "3D Slicer." `http://www.Slicer.org/`, February 2010.

[21] D. Gobbi, "AtamaiViewer software.," May 11 2004.

[22] P. F. Hokayem and M. W. Spong, "Bilateral teleoperation: An historical survey," *Automatica*, vol. 42, pp. 2035–2057, December 2006.

[23] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modelling and Control.* John Wiley and Sons, Inc., Hoboken, NJ, USA, 1st ed., November 1989.

[24] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control.* Springer-Verlag London Limited, 1st ed., 2009.

[25] B. Siciliano, "Kinematic control of redundant robot manipulators: A tutorial," *Journal of Intelligent and Robotic Systems*, vol. 3, pp. 201–212, September 01, 1990.

[26] P. Chiacchio, S. Chiaverini, L. Sciavicco, and B. Siciliano, "Closed-loop inverse kinematics scheme for constrained redundant manipulators with task space augmentation and task priority strategy," *International Journal of Robotics Research*, vol. 10, pp. 410–425, August 10, 1991.

[27] W. A. Wolovich and H. Elliot, "A computational technique for inverse kinematics," in *Proceedings of IEEE International Conference on Decision and Control, Las Vegas, USA*, pp. 1359–1363, December 12-14, 1984.

[28] R. Campa and H. de la Torre, "Pose control of robot manipulators using different orientation representations: A comparative review," in *Proceedings of IEEE American Control Conference, St. Louis, MO, USA*, pp. 2855–2860, June 10-12, 2009.

[29] F. Caccavale, B. Sicilliano, and V. Luigi, "The role of Euler parameters in robot control," *Asian Journal of Control*, vol. 1, pp. 25–34, March 2, 1999.

[30] B. Xian, S. de Queiroz, D. Dawson, and I. Walker, "Task-space tracking control of robot manipulators via quaternion feedback," *IEEE Transactions on Robotics and Automation*, vol. 20, pp. 160–167, February 2004.

[31] J. Luh, M. Walker, and R. Paul, "Resolved-acceleration control of mechanical manipulators," *IEEE Transactions on Automatic Control*, vol. 25, pp. 468–474, June 1980.

[32] J. Angeles, *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms.* Springer, 5th Ave., New York, NY, USA, 2nd ed., October 16, 2002.

[33] L. Stocco, S. Salcudean, and F. Sassani, "Optimal kinematic design of a haptic pen," *IEEE/ASME Transactions on Mechatronics*, vol. 6, pp. 210–220, September 2001.

[34] A. Tobergte, R. Konietschke, and G. Hirzinger, "Planning and control of a teleoperation system for research in minimally invasive robotic surgery," in *Proceedings of IEEE International Conference on Robotics and Automation, Kobe, Japan*, pp. 4225–4232, May 12-17, 2009.

[35] H. H. Mabie and C. F. Reinholtz, *Mechanisms and Dynamics of Machinery.* John Wiley and Sons, Inc., Hoboken, NJ, USA, 4th ed., January 2, 1987.

Appendix A

Special Consideration of the Four-Bar Mechanism Solver.

The four-bar mechanism is a well known kinematic mechanism. The equations to solve for the motions of this mechanism can readily be found in textbooks. The schematic for this mechanism is shown in Fig. A.1. Where, Joints 1 and 4 are connected to ground and Joint 1 is rotated, moving the rest of the links. When Joint 1 is actuated, the equations for this mechanism solve for the position of Joint 3. Once the position of Joint 3 is known, the position and orientation of all the links are defined [35].

The equations for this mechanism are generally defined for 2-D space. The following steps show how the simulator performed the calculations for the four-bar mechanism in 3-D space.

- Before any calculations are performed, the four joints assigned to the four-bar solver are checked, making sure they are parallel to each other.
- If the four joints used to create the mechanism are found to be parallel, the following values are then recorded.
  - The initial distance between Joint 1 and 3.
  - The vector Z from Fig. A.1.
  - The lengths between each of the joints.

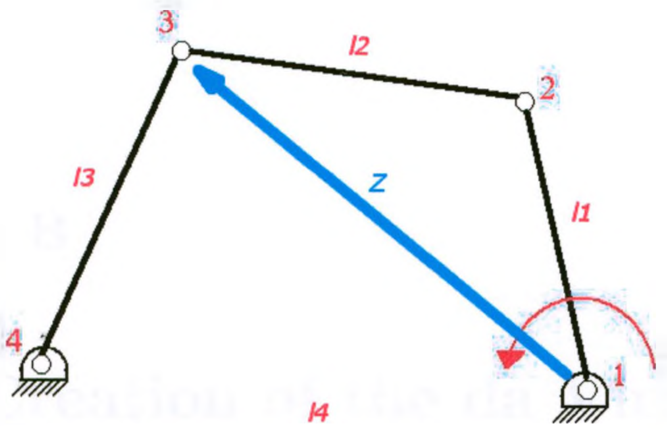


Figure A.1: A Schematic of links that make up a four-bar mechanism.

- When Joint 1 is rotated, the following steps are followed to shift the joints such that the four-bar solver equation are valid.
  - The joints are all rotated such that the direction of Joint 1 is aligned with the world Z-axis.
  - The joints are then translated to make the position of Joint 1 coincided with the origin of the world frame.
  - The four-bar mechanism equations are then used to recalculate the position of Joint 3.
  - Calculations are performed to make sure that the original link lengths are maintained and that a valid solution has been obtained. The length of Z is also compared to it's previous value to make sure that it did not change by a large amount, to prevent the inversion of the mechanism. If a valid solution is not found, all the motions are reversed and failure flag is set.
  - All the joints are then rotated and shifted back to their original positions, in the 3-D environment, using the inverse of the matrices that were used to shift the joints for the mechanism solver.

Appendix B

Model Creation of the da Vinci Surgical Robot for Simulation.

To allow for the simulation of the current robot used in RAMIS, the da Vinci was measured to create a CAD model. The da Vinci is produced by a private corporation for profit. Therefore, the company will not easily disclose any information about its product that they think should be kept a secret, such as the dimensions of the robot. Thus, to create a computer model of the robot, a da Vinci robot was manually measured. This process is described here.

The da Vinci is a teleoperated robot, where the surgeon sits at a master console and controls the slave section, which performs the procedure inside the patient. These two sections can be seen in Fig. B.1. Only the slave section of the robot was modelled for this work.

Most of the shafts, for the joints of the slave section, are covered by a protective casing. This makes it difficult to get the exact dimensions of the robot correct. These protective casings could not be removed as the robot is currently being used for surgical procedures.

To ease the process of measuring the da Vinci slave section, the linkages were categorized into the following sections: the base frame, the passive linkages, the tooled active sections, the endoscopic active section and the removable instruments. These categories are illustrated in Fig. B.2.

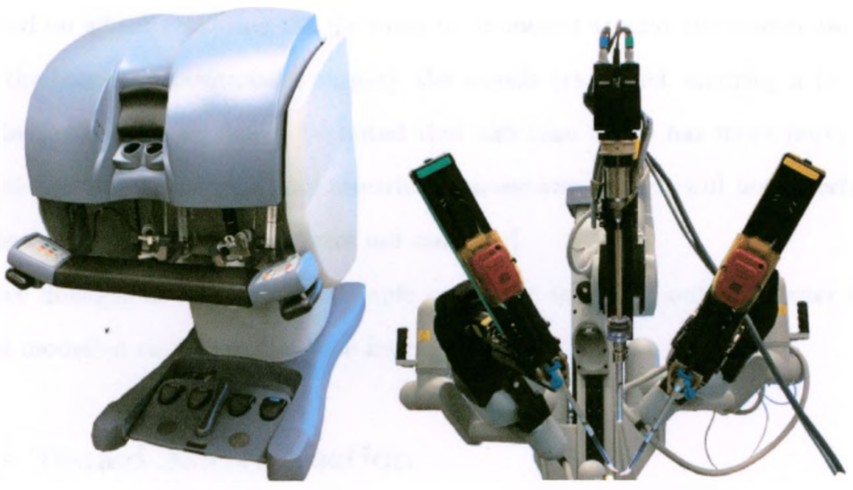

(a) (b)

Figure B.1: Pictures showing the (a) master and (b) slave sections of the da Vinci.

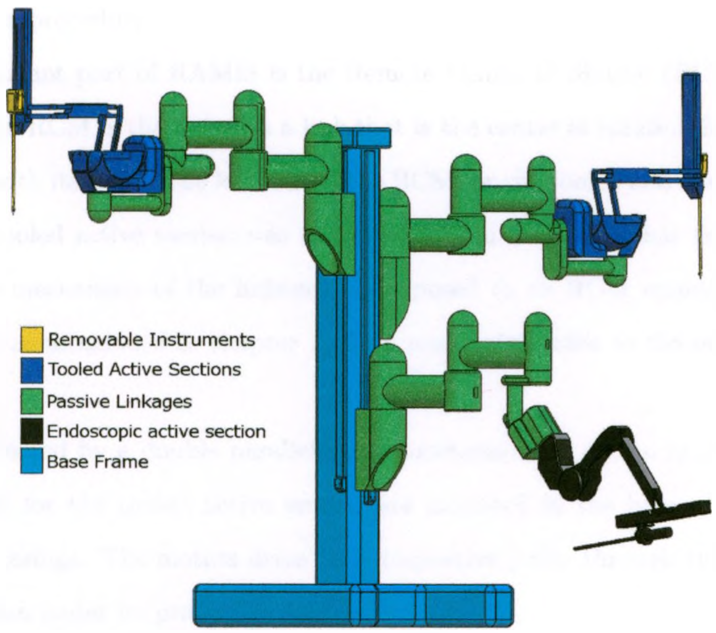


Figure B.2: A screen capture showing the different categories of the slave section of the da Vinci.

B.1 The Base Frame and Passive Links

The base frame of the robot forms the main support structure on which everything else is mounted. The da Vinci is positioned in the operating theatre using the base as the reference point. The base is mounted on wheels, allowing the da Vinci to be moved around and stored away, when not in use. Once the base is in position for surgery, the wheels are locked, securing it in place for the duration of the procedure. It should be noted that the base frame has more parts than shown in Fig. B.2, such as the electronics and electrical connections, which will not interfere with the motions of the robot. Therefore, they were not modelled.

The passive linkages of each arm are simple geometric links and only the outer casings were measured and modelled similar to the base frame.

B.2 The Tooled Active Section

The tooled active sections is the section of the robot that moves the laparoscopic instruments during a procedure. These are the only sections of the robot, along with the endoscopic section, that moves during a procedure.

The most important part of RAMIS is the Remote Center of Motion (RCM) created by the robot's linkages. An RCM is the point on a link that is the centre of rotation for that link, which does not coincide with its joint. The location of the RCM for the tooled active section can be seen in Fig. B.3. The tooled active section was designed in such a manner that the RCM is created by the mechanical mechanism of the linkages, as apposed to an RCM maintained by software, or passive linkages as mentioned in chapter 1. This mechanism adds to the overall safety of the manipulator.

The RCM is created by a double parallelogram mechanism, as shown in Fig. B.4. The electronics and motors for the tooled active section are mounted in the base of the tooled active section, inside its casings. The motors drive their respective joints through tungsten cables that run along the section under its protective casings.

The first teo joints of the active section are covered by casings and other objects. The locations of the hidden shafts are shown in Fig. B.5.

In an attempt to accurately measure the active sections mechanism, the size of the linkages were measured where possible and the location of the covered joints were then estimated. A MATLAB function was then written to simulate the motion of the tooled active section based

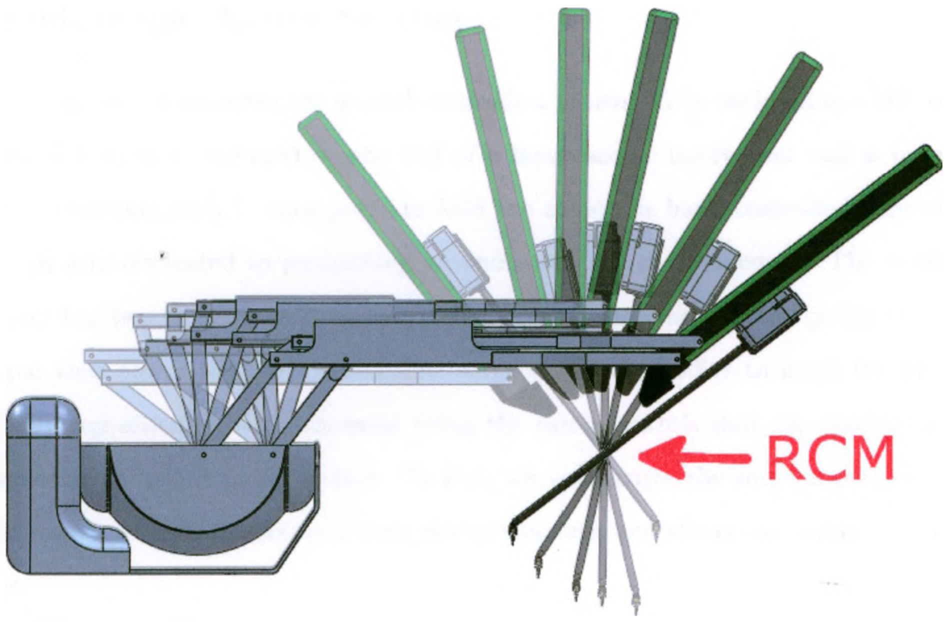


Figure B.3: An image showing the motion of the tooled active section and the RCM created by the its linkages.

on these measurements. The MATLAB function simulates the motion of the mechanism, while recording the motion of the expected position of the RCM. To reduce the motion of this point, an optimization algorithm was applied, where the cost function was the sum of the euclidean distances between the initial position in a motion and every subsequent position of the expected RCM. The initial position is defined when the first link in the chain is perpendicular to the base linkage that it is attached to. This initial position is the 0° position.

The simulated motion of the mechanism is created by rotating this link from -45° to 60°. The motion of the mechanism can be seen in Fig. B.3. It was found that the only measurements to affect the motion of the expected RCM are the variables ϕ and the distance between joints 1 and 3 (l_1) as shown in Fig. B.6. The length l_1 can be measured relatively accurately and is assumed to be correct. Therefore, the code was changed such that it only optimizes for the value of ϕ.

The optimal angle was found to be 15.07°. If the length l_1 is changed, a new optimal angle would need to be recalculated.

B.3 Endoscopic Active Section

To assist a surgeon in visualizing the procedure inside a patient while performing a MIS or RAMIS procedures, a camera is mounted on the end of a laparoscopic instrument and is inserted with the other instruments with its own port. In MIS the camera is hand controlled, however the da Vinci has an arm dedicated to positioning an endoscope for the procedure. The endoscope for the da Vinci has two cameras next to each other on the same instrument, giving the surgeon a stereoscopic view and a sense of depth. The surgeon has the ability to move the view around by physically repositioning the endoscope using the same controls that are used to control the instruments on the tooled active section. To shift the endoscope, the surgeon presses down on a foot pedal, which releases the controls from the instruments and allows the surgeon to control the endoscope.

To allow the endoscope to physically move inside the patient, it also needs an RCM. This RCM is generated in the same method as the one on the tooled active section with a minor modification in the linkage mechanism.

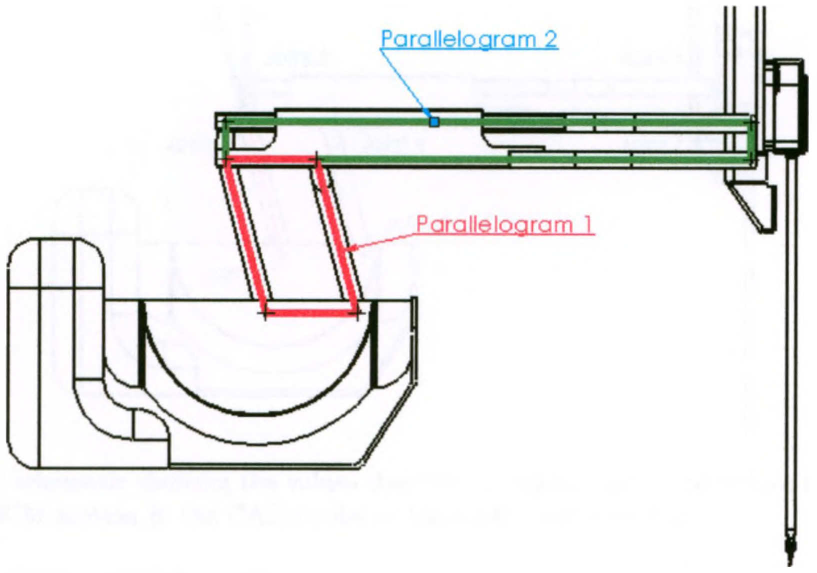


Figure B.4: Schematic of the tooled active section, showing the double parallelogram linkage mechanism that creates the RCM.

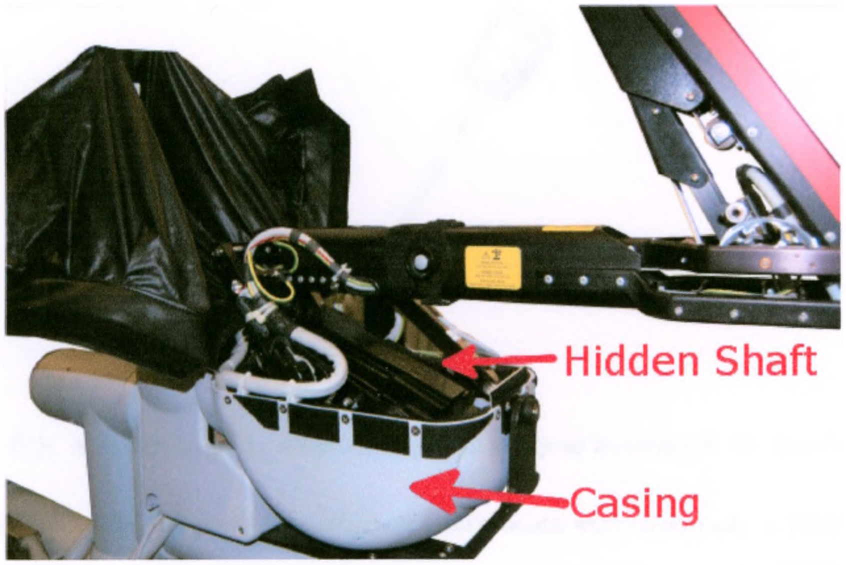


Figure B.5: An image showing how casings around shafts hinder the measurement of the tooled active section.

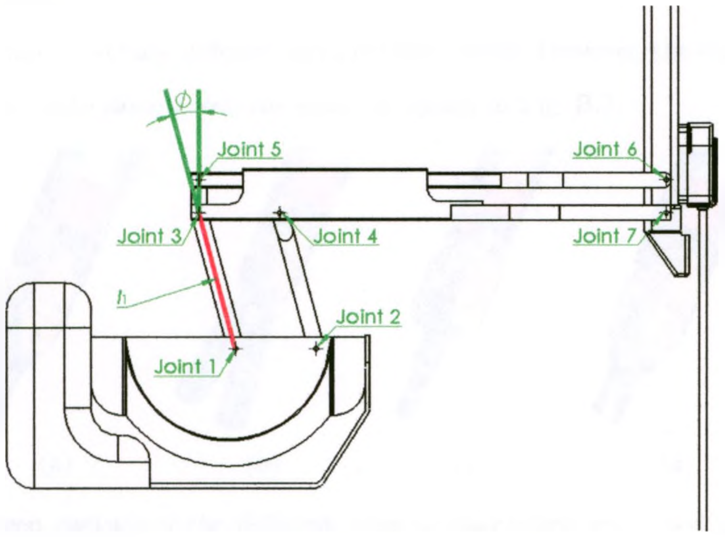


Figure B.6: A schematic showing the values that can be optimized for, to reduce the error in the RCM motion in the CAD model of the tooled active section.

B.4 The Surgical Instruments

The surgical instruments of the da Vinci are mounted onto the tooled active sections to perform the procedures. They can be easily removed or interchanged based on what the surgeon requires

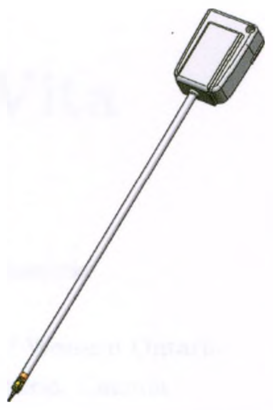


Figure B.7: A screen capture showing a typical surgical instrument for the da Vinci.

during a procedure. The following four surgical instruments were modelled, as they are the tools generally used by the da Vinci for cardiac procedures. These tools are the Cardiere Forceps, Permanent Cautery Spatula, Potts Scissors and the Small Clip Appliers. The wrists for these instruments are shown in Fig. B.8.

The four tools modelled have different wrist configurations. However, the configuration of each surgical instrument is the same above the wrist, as shown in Fig. B.7.

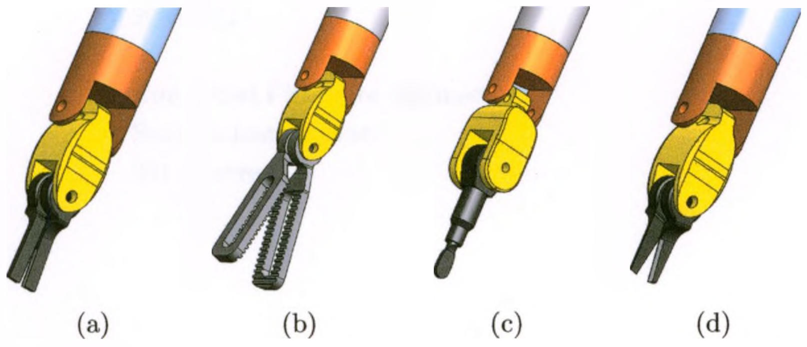


Figure B.8: A screen capture of the different surgical instrument ends used in the CABG procedures. The above ends were named as follows: (a) Alligator Clips, (b) Caudiere Forceps, (c) Permanent Cautery Spatula and (d) Pott's Scissors.