2015-04-30

Implementation of a Surgical Robot Dynamical Simulation and Motion Planning Framework

Adnan Munawar
Worcester Polytechnic Institute

Implementation of a Surgical Robot Dynamical Simulation and Motion Planning Framework

by

Adnan Munawar

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Robotics Engineering

by

May 2015

APPROVED:

_____

Professor Gregory Fischer, Major Thesis Advisor

_____

Professor Micheal Gennert, Thesis Committee Member

_____

Professor Dmitry Berenson, Thesis Committee Member

Abstract

The daVinci Research Kit (dVRK) is a research platform that consists of the clinical daVinci surgical robot, provided by Intuitive Surgical to Academic Institutions. It provides an open source software and hardware platform for researchers to study and analyze the current architecture and expand the capabilities of the existing technology. The line between general purpose robotics and medical robotics has segregated the two fields. A significant part of the segregation lies at the software end, where new tools and methods developed in general purpose robotics cannot make it to medical robotics in a short amount of time. This research focuses on the integration of a widely used software architecture for general purpose robotics with the dVRK with the hope of utilizing the research and development from one field to the other. As a first step towards this bridging, a motion planning framework and a dynamic simulator has been developed for the dVRK using ROS. The motion planning framework is aimed to assist the surgeon in performing task with additional safety and machine intelligence. A few use cases have been proposed as well. Lastly, a Matlab Interface has been developed that is standalone in terms of usage and provides capabilities to interact with dVRK.

Acknowledgments

I would like to acknowledge the continuous support of my parents throughout my Masters and in fact my entire academic life. Thanks to Professor Fischer for being such a helpful advisor, giving constructive feedback all along and to all the members of the AIM lab.

I would also like to mention my previous advisor at LUMS, Dr. Abubakr and the team at CYPHYNETs including Talha Manzoor, Zeeshan Sharif, Bilal Talat and Zubair Ahmed who I have worked with and led me to pursue my Masters in Robotics. Lastly I would like to thank my two very good friends, Shahrukh Athar and Dr. Hassan Khan for their motivation and guidance.

Contents

List of Figures

Chapter 1

Introduction

1.1 Laparoscopic Surgery

In traditional laparoscopic surgeries, long and slender shaped tools are used by the surgeon for insertion into the patients body. These tools have specially designed end-effectors for different uses, ranging from slicing/cutting tools to grippers and even small cameras for endoscopy [5]. A small incision is made at the point of entry and from that point the tool is held by the surgeon. The nature of the procedure provides many advantages over non-laparoscopic surgery. Studies show, that in the surgeries performed laparoscopically, there is less hemorrhaging which directly reduces the need for blood transfusions[27]. Due to smaller and accurate incisions to just the affected area, other organs are not exposed and there is much lower risk of infection. Hospital stays are much shorter, often with the same day discharge. Earlier discharge is perhaps one of greatest advantages of laparoscopic surgery from the patient's point of view as normal life can be resumed much sooner[27]. Additionally, patients report much less pain, following the laparoscopic surgery, due to which much lower to no dosage of pain killers is required.

Despite the many advantages of the traditional laparoscopic surgery over the non-laparoscopic surgery, the drawbacks include the loss of touch for the surgeon, reduced degree of motions and the indirect view of the area being operated, to name a few[27]. Additionally, the view is limited to 2 dimensions on a computer screen as opposed to the natural view while performing open surgery. The tremors induced by the surgeon's hand are transmitted through the surgical instrument and can cause unwanted incisions and damages to the area being operated. Even for an expert surgeon, studies show that the tremors happen when the surgeon gets tired and exhausted. Thus for longer period of surgery, the surgeon can very easily get tired of holding the surgical tool and this can induce tremors that can be fatal to the patient in precise surgical requirements. The disadvantages and shortcomings of traditional laparoscopic surgery infact set up the use-case for Tele-Operable Surgical Robots

1.2 A Historical Perspective to Robotic Surgery

For the past several decades researches and industries have applied robotics to various fields, ranging from deep underground mining to space and interplanetary applications. Not surprisingly, application of robotics in the field of Medicine and surgery has found use cases as well. From a historical point of view the PUMA 560 was employed for medical surgery [19] for the very first time in 1985. The PUMA 560 was a general purpose robotic manipulator, possessing 6 Degrees of Freedom, with a spherical wrist. Kwoh et al[19] were behind this project and were able to achieve neurosurgical biopsies with the PUMA 560. The motivation, and to great extent the success of the project was improved precision achieved by the use of a robot.

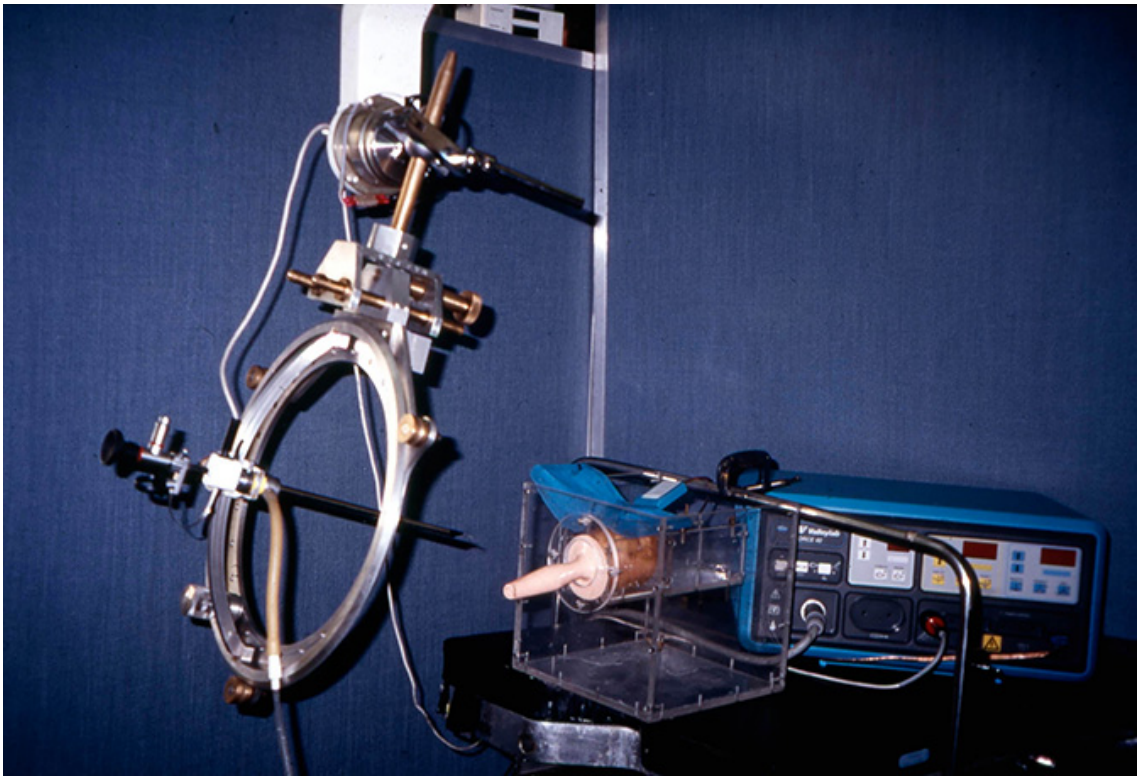


Figure 1.1: PROBOT for Transurethal Resection of Prostate

Following the success, a couple of years later the team utilized the PUMA 560 to perform transurethal resection of a prostate[27].

While the PUMA 560 was applied to perform non-minimally invasive surgeries with some noticeable advantages, the real breakthrough in Minimally Invasive (or Laparoscopic) surgery was achieved in 1987. The surgery aimed to remove a gall blader with minimal intervention[10]. Since then, robots has been used more frequently in surgical procedures. There success and potential advantages of the PUMA 560 in surgery related to the transuretheral resection of prostate, led to the development of PROBOT[8]. Similar to experimental setup of PUMA560 for its first surgery, the PROBOT(Fig 1.1) was specialized to perform prostate surgeries[8].

(a) The ZEUS system with Master and Slave Console

(b) The ZEUS Slave Console

Figure 1.2: Figures (a) and (b) showing the ZEUS surgical robot in very early procedures

Meanwhile, a new system by the name of ROBOTDOC[3] (Fig. 1.3) was being developed by Integrated Surgical Supplies at California, US. The functionality of ROBODOC was quite different from that of PROBOT as ROBODOC was designed to assist in Hip Replacement surgeries[3]. The procedure used abrasion of the femur in the hip, to provide room for the replacement. ROBODOC got the approval of the Food and Drug Administration (FDA) to perform limited clinical procedures[31].

The advent of Tele-Operated surgery as we know of today, was initiated by a joint collaboration between researchers at Ames Research Center at National Air and Space Administration (NASA) and Stanford Research Institute (SRI)[32]. The researchers were developing a Tele-Operated tool for surgical procedures. This sparked interest in the US military to fund the development of such a system for war time[27] [32]. The motive behind the funding and development of the project was to reduce the mortality rate of the wounded soldiers by providing on-sight rehabilitation and surgical capabilities. This would have potentially provided valuable time to prevent the wounded from exsanguinating. The system was called Mobile Advanced Surgical

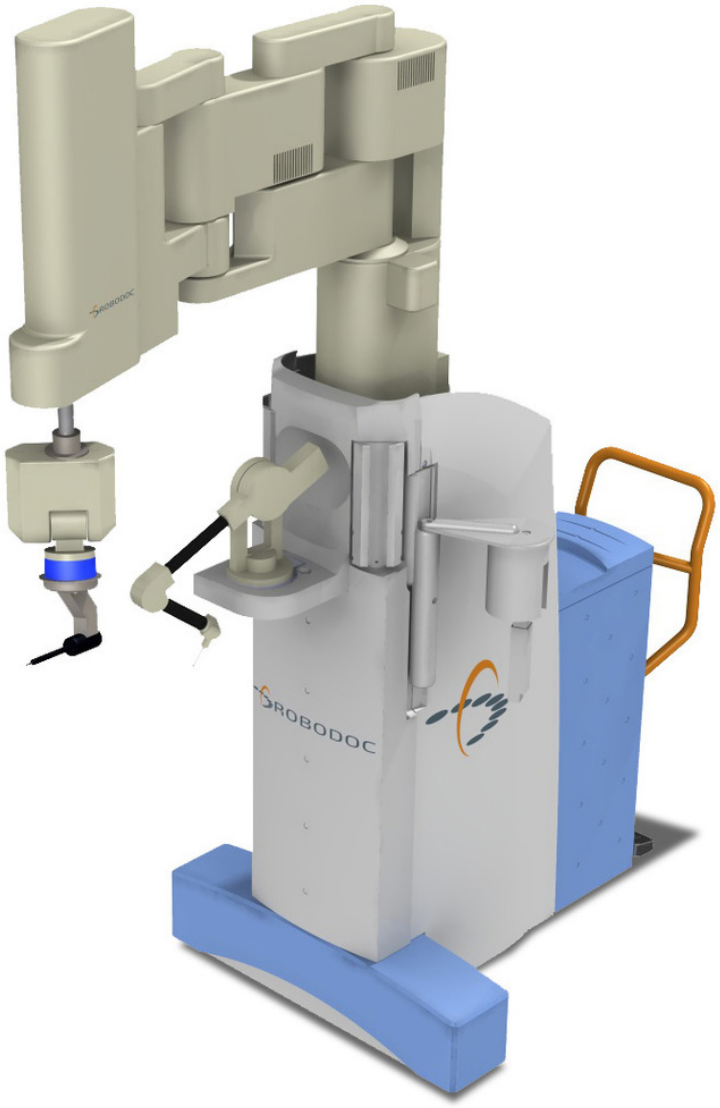


Figure 1.3: ROBODOC robot by Integrated Surgical Supplies

Hospital (MASH) and was devised to be controlled by expert surgeons from some remote location. The system was never tested on humans, however several animals were used for various experiments with good results[31].

Meanwhile ZEUS Robotic Surgical System (Fig 1.2) was another medical robot that was uder development. The ZEUS system employed two robotic arms for

laparoscopic and thoracoscopic surgeries. It also had a third arm for endoscopy. This arm was called the Automated Endoscopic System for Optimal Positioning Robotics System (AESOP) and was voice controlled. The AESOP was granted the FDA approval in 1994[2], meanwhile the ZEUS system got FDA clearance in 2001[2].

During the same course of time the ZEUS surgical system was being developed and tested, Integrated surgical licensed the research project by SRI and improved the system with massive redesign to form one of the most advanced Tele Operated Surgical Robot at that time [4] [32]. The system was called the daVinci Surgical System. The company now called Intuitive Surgical, renamed from Integrated Surgical Supplies, is currently producing the fourth generation of the daVinci Surgical Robots. The daVinci surgical system has been used to perform more than a million surgeries since its inception, with more than 200,000 surgeries in the year 2012 alone[25].

1.3 The daVinci Surgical Robot

For the daVinci Surgical System, the design has been chosen to provide a comfortable console to the surgeon that prevent fatigue over long periods of surgical procedures. It also provides greater degrees of freedom, 7 in the daVinci's case[27], which is the same as the DOF for a human hand. A stereoscopic view of the surgical area is provided, that gives the same sense of maneuverability as viewing naturally. Tremors are taken care of by filtering th inputs by the surgeon, so even if the surgeon gets tired of longer surgical procedure and begins to transmit tremors to the MTM, the system isolates those tremors from translating into the PSMs movements, while

Figure 1.4: The daVinci Surgical System with the surgeon operating the MTM console and the helping nurse at the PSM station

still allowing dexterous movements.

The advantages mentioned above are mostly to cater for the shortcomings of the traditional laparoscopic surgery. There are many more advantages of surgery using the daVinci Surgical systems apart from the ones mentioned above. However, as of any surgical procedure, the daVinci surgical system also lacks in a few aspects and not all surgeries performed by the daVinci were successful. These include cuff dehiscence, bladder injuries and some technical faults that prevented the normal operation of the robot [28]. This is expected and somewhat understandable, as no surgical procedure is always successful. But the shortcomings of the daVinci surgical robot do require more research into the use of robots in surgery. Beyond any doubt, a lot of scrutiny, experimentation and a structured thought process is required to improve upon the current generation of robot assisted surgery.

1.4 The daVinci Research Kit (dVRK)

Intuitive surgical for the first time initiated a research platform where universities and researches could study the daVinci Surgical platform and extend the capabilities of the existing system. The research platform has been named "daVinvi Research Kit" or for shot the dVRK[36]. As a kick off, 12 universities/institutions were provided with the dVRK.The stripped down research kit consists of two Master Tool Manipulators (MTMs), two Patient Side Manipulators (PSMs) and a stereoscopic display unit[36]. Some universities are offered four instead of two PSMs. The control of PSMs from the MTMs is made possible with the coordination of a foot-pedal module that comes standard with the basic daVinci surgical robot. In a nutshell, the role of the foot pedal is somewhat analogs to that of an automobile, engaging the PSMs through MTMs, idling the PSMs and re positioning the MTMs while locking the PSMs. There are additional uses of the foot pedal as well which are discussed in the coming chapters.

The daVinci Surgical System has been provided to WPI as part of the dVRK Program.The stripped down dVRK lacks the hardware and software interfaces for communication between the MTMs, the foot pedal modules and the PSMs. This comprised the initial challenge of getting the system powered-up and running. WPI collaborated with the John Hopkins University **JHU** to acquire the controllers for establishing hardware interface between the dVRK components. More on the general design and architecture of the controllers is discussed in chapter 3.

Additionally, the controllers developed by **JHU** are integrated and coupled to the software libraries developed and provided by them. The core libraries used for this purpose are the CISST libraries. These libraries are utilized by an open source platform called Surgical Assistant Workstation (SAW), developed by the same team at JHU. SAW integrates the cisst libraries to develop applications for different types of medical equipment/machines and robots. SAW and CISST are discussed in more detail in chapter 2 and 3. The universities/institutes that collaborate on the dVRK share research with each other. There are dedicated discussion forums and repositories that help the universities share their experiments and results.

The work presented in this thesis is summarized in the following points:

- Developed a cisst-ROS bridge for integrating ROS as a high level controller for the dVRK
- Implemented a motion planning framework for the dVRK and demonstrated a use case using assistive motion planning for the surgeon during pre and intra-operative surgeries with the daVinci
- Developed a dynamical simulation of the dVRK manipulators.
- Implemented a Matlab based class to allow for the control of the dVRK using Matlab. This feature extends the dVRK framework and allows more room for development in different platforms

This work in presented in different chapter that are summarized as follows:

- **Literature Review:** This chapter discussed the software frameworks that were developed before ROS in dVRKs perspective. Their life cycle, limitations and the evolution of ROS is then presented. CISST/SAW is elaborated and

presented with its advantages for dVRK and its shortcomings for developing new algorithms and carrying out experiments.

- **System Architecture:** The overall dVRK system is broken down into hardware and software architecture for the sake of understanding. All the major hardware units are then discussed and evaluated. This evaluation includes the design of the MTMs and PSMs and their capabilities. Next the software architecture is discussed, which includes the cisst-ROS bridge.

- **Models for dVRK:** The CAD models for the dVRK manipulators are discussed in this section and their conversion to URDF format for use with ROS and Gazebo. The difficulties associated with the design of the CAD models and subsequently the URDF files is discussed and the improvements to cater for some of the problems is presented.

- **Integration of ROS and CISST/SAW:** Detailed integration of the CISST/SAW with ROS is discussed in this section. The changes to be required in the CISST/SAW libraries for the cisst-ROS bridge to be possible are presented. Additionally a brief overview of SAW applications is presented keeping cisst-ROS bridge in perspective. The chapter also details the use of the cisst-ROS bridge interfaces to set up subscribers and publishers to control the dVRK.

- **Motion Planning for the dVRK:** A motion planning framework is implemented for the dVRK and presented in this chapter. The life cycle of development from Matlab to ROS is presented and a few use cases have been proposed. Comparison between various planners using custom Matlab bench marking interface has been proposed. The chapter concludes with some experiments in simulations.

- **Dynamic Simulator for the dVRK and a Matlab Interface:** This chapter presents a dynamic simulator for the dVRK and a Matlab Interface for complete control of the manipulators.

- **Conclusion and Future Work:** Finally a conclusion is presented in this chapter followed by the future work lined up.

Chapter 2

Literature Review

This chapter discusses the historical issues of code re usability and software integration of general purpose robotics. Many popular software frameworks during the past decade have been briefly discussed in relation to dVRK. The advantages and disadvantages are discussed that led to the development and widespread used or Robot Operating System(ROS).

2.1 Limitations to Robot Specific Software Framework

In the field of science and technology, there are many instances where reinventing the wheel is required. This slows down the research quite significantly. Robotics research is no different and infact suffers (or historically suffered) from this problem much drastically. From developing new algorithms to conducting experiments, most of the times, research starts by creating of a small robot or a specific mechatronic system. Even with the use readily available power, control and hardware components, the time required is still significant before any experimentation or studies

can be conducted. Due to this work flow, the time required to reach to a stage where experimentation can be done or newly algorithms/techniques can be developed overshadows the entire time line of the project. This is specially true for academic research where both time and funds are limited.

Towards hardware end, the availability of various components have improved over the past, with readily available controllers, power boards and sensors that can almost be used in a plug and play fashion. At the software end (both low level and high level) however, the gap only widened due the plethora of hardware components and their varied interfaces for software integration. The codes developed at different research labs were usually too tightly woven around the application and robots that they were designed for. Modifying that code to suit other applications could potentially take a lot of time and effort, provided one got access to the code in the first place. High level control using advanced visual and sensing applications remained copyrighted, vaguely implemented or un-maintained if available open source. In addition to that, the plug and play feature that is available in hardware components was rarely available in the software applications, and requiring a lot of time to configure for application specific usage.

2.2 Development of CISST/SAW at LCSR

The Lab for Computational Sensing and Robotics (LCSR) at John Hopkins developed a set of libraries named "The Computer Integrated Surgical Systems and Technology" [9] or CISST for short. CISST is a collection of general purpose libraries for tasks ranging from basic vector manipulation, arithmetic to libraries employed for OS layer abstraction, multi threading and data logging [9]. A subset of the

various libraries developed under CISST are:

- cisstCommon
- cisstVector
- cisstNumerical
- cisstStereoVision
- cisstRobot
- cisstOSAbstraction
- cisstMultiTask

To give a general overview of the naming convention used, in the list above, cisstStereoVision is a dedicated library for video acquisition, processing and display[22]. cisstRobot is a library for solving robot kinematics and dynamic equations[22]. The reason for mentioning these libraries here with the naming convention chosen is to explain the purpose and motivation of the development. The libraries have been built from the ground up in C++ to provide API for the development of complex software for robotic applications.

The Surgical Assistant Workstation (SAW) is a component based software architecture that provides API for creating software for Medical Robotic Applications[39]. The component based approach allows for modularity and designed to connect multi-level systems with relative ease. Medical Robotic applications that involve tele-operation using a Master-Slave pair(s) are examples of the target devices for SAW[11]. Figure 2.2 shows many medical robotics hardware components connected

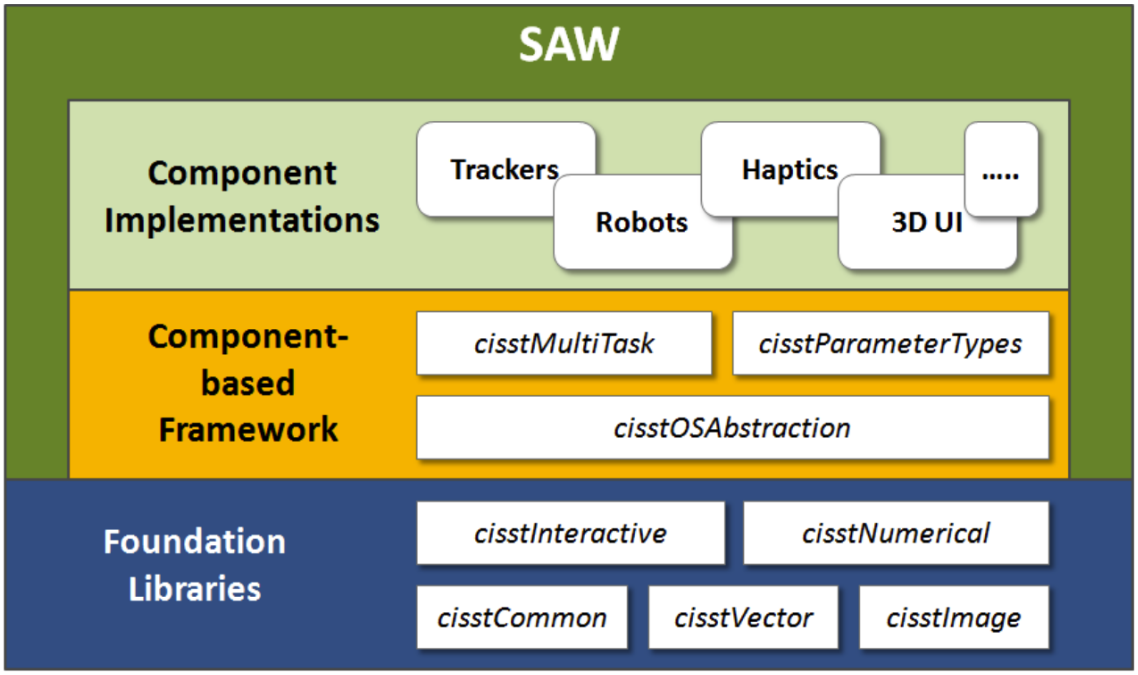


Figure 2.1: SAW Application built upon the CISST libraries [11]

to SAW using the component based approach.

Figure 2.1 shows a visual description of the integration of CISST in SAW. Each software and hardware unit is considered as a component and data exchange between the components is carried out in real time using defined interfaces. Hardware components are wrapped inside device drivers for abstraction purposes. Thus the SAW framework is designed to include video, haptics, tracking and other devices as components to connect to each other and the software components, modularly, and thus providing high speed data communication between them. SAW has been built open the CISST libraries using the component based infrastructure of CISST[39].

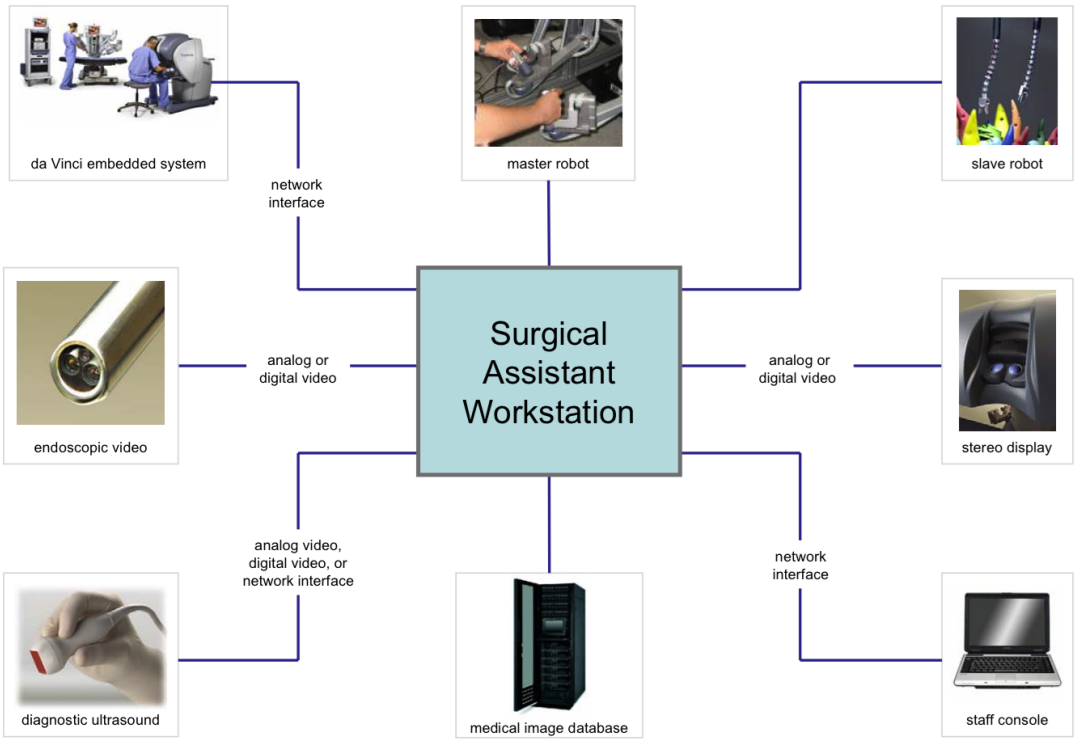


Figure 2.2: Multiple Hardware Components Connected to SAW [38]

2.3 The OROCOS Project

The Open Robotics Control Software or OROCOS, was designed as a general purpose, real time and component based architecture for robot control applications. The software was initially released in 2005 [26]. Since then the software has evolved into four C++ implemented libraries [7]. These libraries are named such as to demonstrate their target applications. Following are the four libraries:

1. The Orocos Real-Time Toolkit (RTT)
2. The Orocos Components Library (OCL)
3. The Orocos Kinematics and Dynamics Library (KDL)
4. The Orocos Bayesian Filtering Library (BFL)

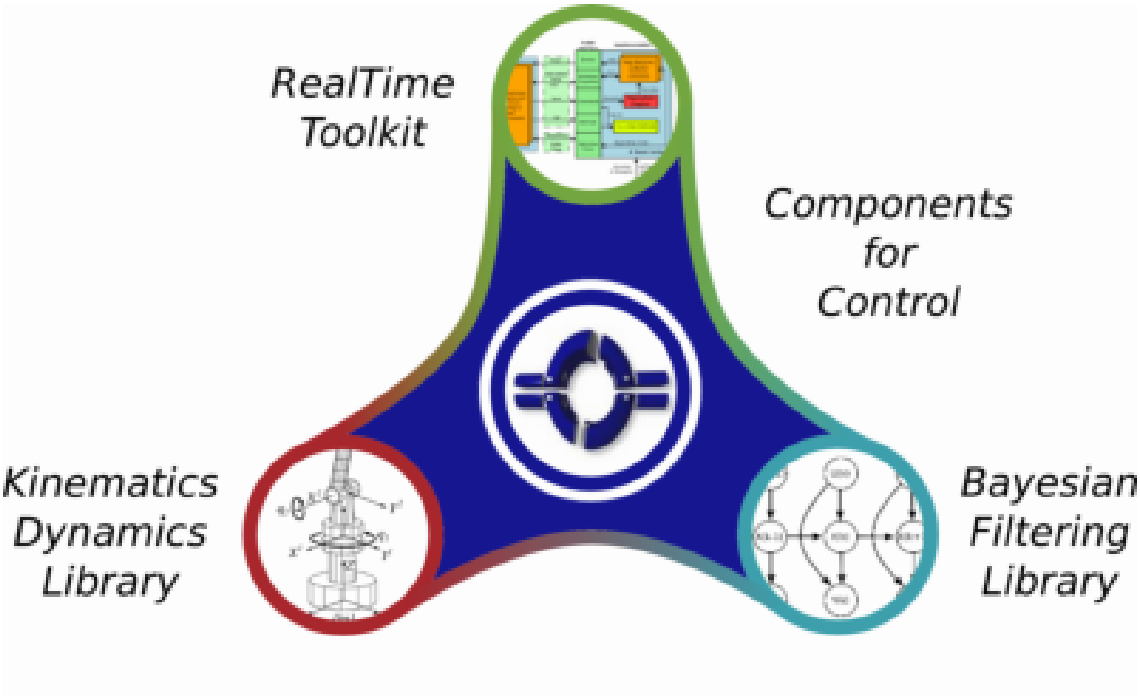


Figure 2.3: The OROCOS Project [7]

The RRT and OCL library are the base implementations for using KDL and BFL, however, both KDL and BFL are available as standalone libraries for use with other C++ libraries. From the four categories, it is evident that OROCOS was geared towards robot control and achieved great precision for real time applications. Infact as shown in figure 2.3, taken from the official documentation, the project is aimed towards advanced control of robots.

2.4 Similarities between OROCOS and SAW

While CISST libraries had been in development from as early as 2002, SAW is a much later development. It came around 2006, aiming towards a component based architecture. In that respect, SAW resembles most closely to OROCOS [15] [9], as discussed in the previous section. There are numerous similarities between the two while they differ as well is some core implementations. As an example,

the design similarity between SAW and OROCOS has allowed the Kinematics and Dynamics Library (KDL) to be recently added to the SAW, thereby replacing the native kinematics and dynamics implementations.

2.5 Advent of Robot Operating Systems

Robot Operation System (ROS) came into development in 2007 at Stanford Artificial Intelligence Laboratory[29] with Willow Garage taking its active development and maintenance from 2008 to 2013. The goal behind the creation of ROS as explained in the official ROS documentation is:

> *"ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license."*

The design choices behind the architecture of ROS and the initial collaboration of 20 universities, are most likely the reason that lead to the initial adoption. The framework was easier to use, code was reusable by design, and the level of abstraction between the hardware and software allowed researchers to quickly adopt the framework to their required use. One of the reasons for continued adoption of ROS can be attributed to the ROS Community and open source availability.

As more and more researchers and labs started using ROS, hardware agnostic code that was readily usable began pouring in. The "Reinventing the wheel" was no longer necessary by adopting ROS. This alone was a good enough reason for many

Figure 2.4: ROS usage demographics as of 2011

to start using it. Thus the ease of usage, being open source and a node based architecture lured many researchers in, who eventually created packages that further lured more people in. The cycle continues to this day, with ROS being the popular choice among the many robotics communities worldwide. At this point, ROS has more than several thousand packages for different types of robot and sensory applications, available free and open source in the ROS community.

ROS is a very popular architecture across the world as of now and its adoption rate is quite impressive [1]. The demographic (Fig. 2.4) shows the usage of ROS across the world a few years ago in the all major continents with thousands of research labs using ROS as a development environment. The usage has definitely increased from then as researchers is South Asia specifically started adopting ROS. ROS applications can be generally categorized to the following fields:

- Perception

- Object Identification
- Segmentation and recognition
- Face recognition
- Gesture recognition
- Motion tracking
- Egomotion
- Motion understanding
- Structure from motion (SFM)
- Stereo vision: depth perception via two cameras
- Motion
- Mobile robotics
- Control
- Planning
- Grasping

As of Robotics in general, each individual category can be combined with one or more categories to employ for a different application. For example, for medical robotics combining the features of Grasping, Control, Perception as so on, could lead to improvement in many surgical procedures. The feature set of ROS is what distinguishes itself from OROCOS and SAW, as ROS includes support for most of the modern sensors, displays, algorithms and packages that are running on today's robots.

Chapter 3

System Architecture

In this chapter both the hardware and software component of the dVRK are defined and then discussed. Many of the software components have been developed at JHU, other at WPI by myself and a few other graduate students that worked on the dVRK prior to me.

3.1 Hardware Architecture

For the purpose of explanation of the core dVRK system, smaller or custom hardware components such as stereo cameras, tools of the PSMs are not defined here. They are either considered part of the components already described or not discussed for now.

3.1.1 The Master Tool Manipulators

Keeping that in mind, the Hardware architecture of the dVRK (Fig. 3.1) consists of two Master Tool Manipulators MTMs, two Patient Side Manipulators, a stereoscopic viewer and a foot pedal array [36]. The MTMs are operated by a surgeon

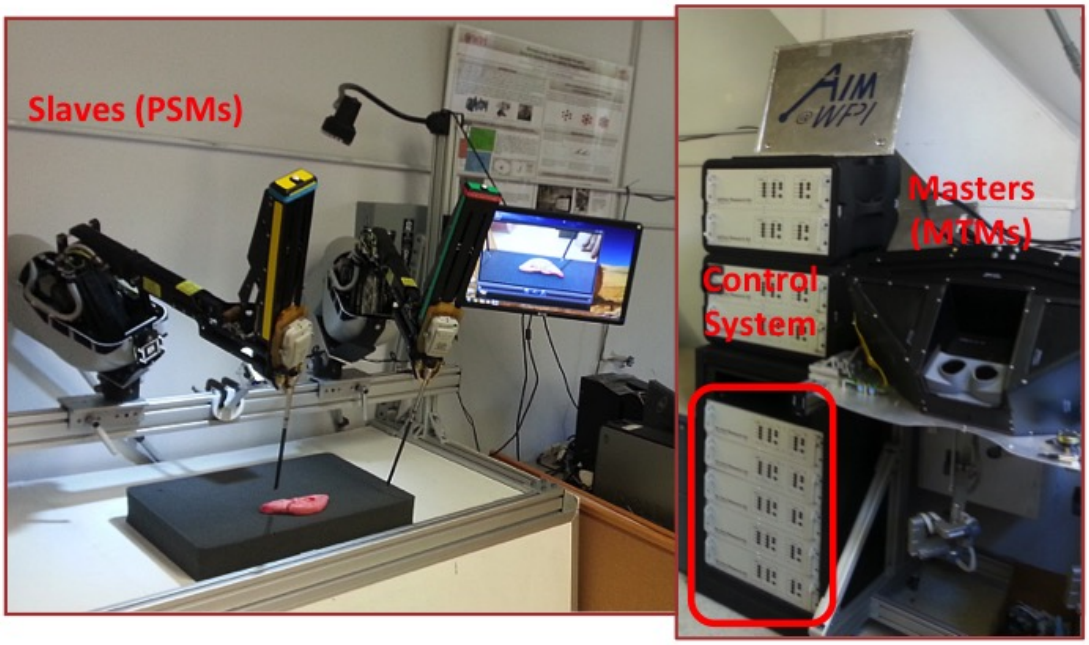


Figure 3.1: The dVRK Components at the AIM, WPI

remotely. Two MTMs are for each corresponding hand of the surgeon, thus the MTMs are referred to from here on wards as MTM right (MTMR) and MTM left (MTML) based on which hand they are designed for.

Each MTM has a 7 DOF configuration space [25], and the arms are designed such that a very natural motion of the surgeons hand is possible while gripping the MTM's end effector. Each joint of the MTM is mounted with a motor. The purpose of these motors is to provide gravity compensation when the surgeon releases the MTMs. The MTMs at AIM lab are shown in figure3.2.

A more elaborate view of MTM is shown in figure 3.3. The direction of move-

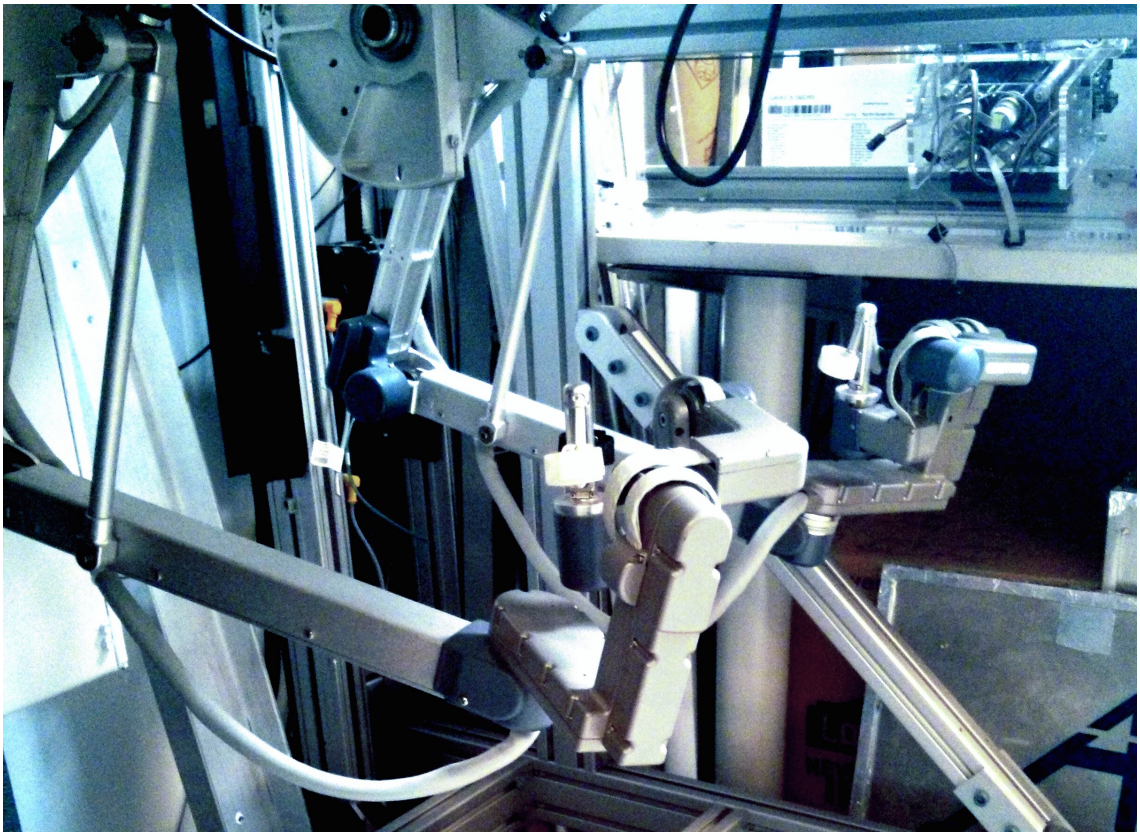


Figure 3.2: The two MTMs at AIM Labs, WPI

ment of each joints is shown with a bidirectional arrow. All the joints of the MTM are revolute. The joints are driven by brush less DC motors and are all cable driven using a capstan drive, except for the base joint, which is directly driven by the motor. It is important to notice that in some instances, MTM is considered to have 8 joints, with the 8th joint being the pinching at the tip of the MTM. This allows for the closing and opening of the grippers of the PSM. For the purpose of Inverse Kinematics, this last joint is ignored since it plays no part in the construction of the configuration space.

Describing the joints backward i.e. starting from the last joint and moving towards the first joint, is more helpful in realizing the design choice of the kinematic

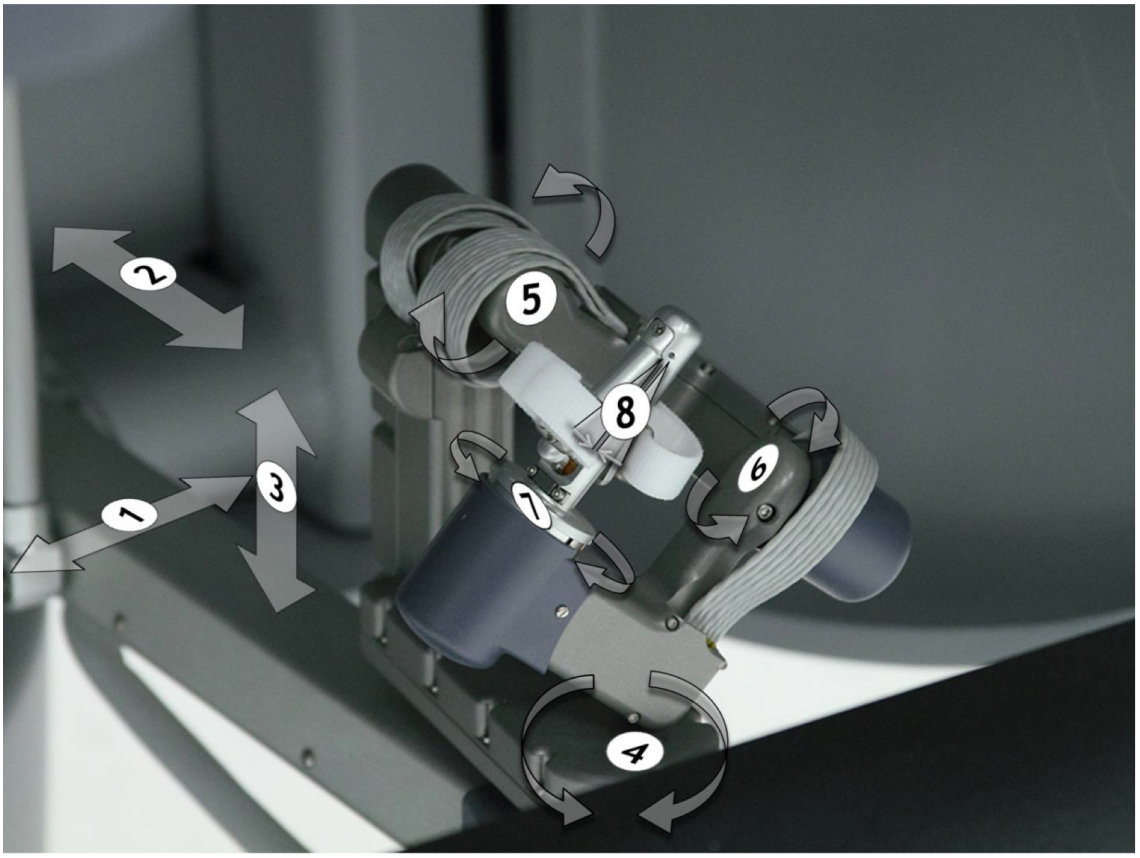


Figure 3.3: Numbered joints with their direction of motion shown by arrows

structure of the MTM. Ignoring the 8th or the last joint, the wrist of the MTM forms a gimbal mechanism with the axes of the joints 4 to 7 culminating into a single point. This kinematic structure allows any orientation of the end effector with one redundant joint. Going further back, the MTM is left with the first 3 joints. These joints merely allow any position of the wrist of the MTM with in the dexterous work space.

3.1.2 The Patient Side Manipulators

The Patient Side Manipulator (PSM), has an altogether different kinematic design as compared to the MTM. Each manipulator has an overall DOF of 7. To provide a fixed entry point into the patient's body, a remote center is designed such that any

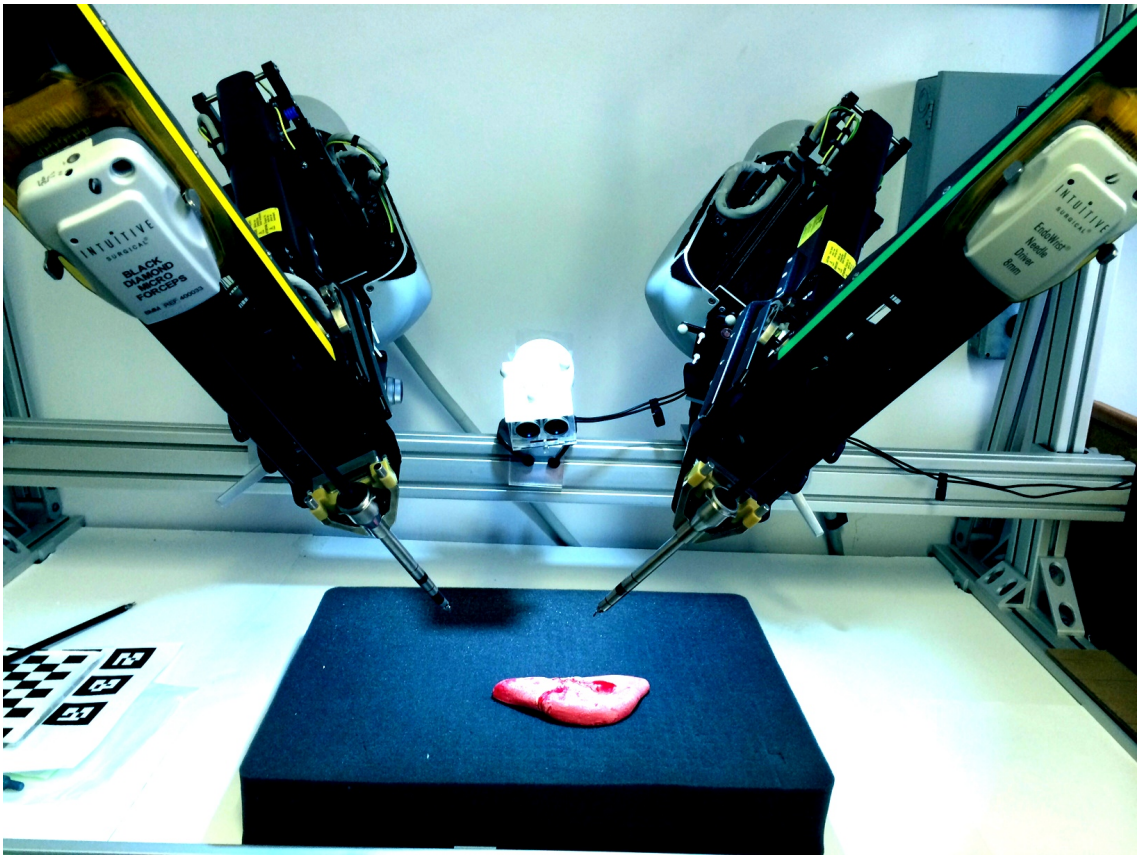


Figure 3.4: The two PSMs at AIM labs, WPI

motion of the PSMs joints does not change that point in space. The remote center is shown in the figure 3.5. The position of the remote center can only be changed by moving the base of the PSM manually, thus during surgical procedures, the nursing staff orients that PSMs to align the remote center at the entry point selected for the operation.

The PSM setup at AIM labs is shown in figure 3.4. The length of space between the two PSMs is adjustable. Going into the detail of the kinematic structure, the first joint of the PSMs is a revolute joint, which is use to achieve the overall yaw rotation around the Remote Center. In the next joint, the PSMs have a parallel joint. It is this parallel joint that provides a remote center, kinematically. The next

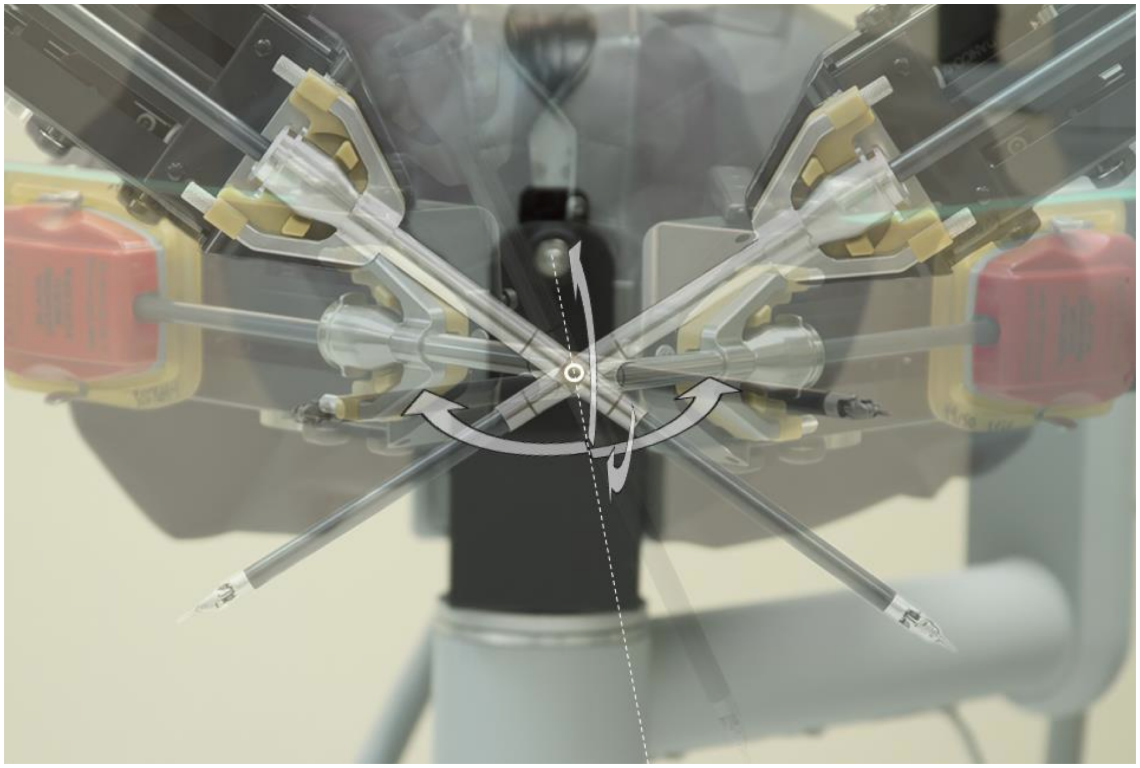


Figure 3.5: The Remote Center of the PSM. (Source: dVRK Manual)

joint is a prismatic joint, used for insertion into the patients body, about the Remote Center. The fourth joint is the rotation of the end effector around the last prismatic joints axis of translation.

The fifth joint is the tool yaw and the sixth joint is the tool pitch angle. The last joint does not play a part in the kinematics of the PSM, similar to the MTM. The last joint is the pinching of the end effector tool. The pinching is achieved if the attached tool is a clipper/gripper type. For tools that are designed for suturing or pulling (using hooks) there is not capability of pinching, thus the last joint becomes inactive. The last joint of MTM, i.e, the gripper pinch is used to achieve a direct control this joint if a gripper like tool is used.

3.1.3 The Foot Pedal Tray

The foot pedal is used to connect/disconnect the motion of the MTMs to the PSMs. Starting from the left, the first food pedal is called mono. This pedal should be pressed in order for the PSMs to be activated by the MTMs. The next pedal is not used for anything as of now. The next pedal is for camera movement for a camera attached to a third PSM. Since we do not have the third PSM in the dVRK, this pedal is not used. The last pedal is used to re position the MTMs while keeping the PSMs where they are. This pedal when pressed allows the surgeon to move the MTMs anywhere, usually within his/her zone of comfort to continue on where he left of at the PSMs end. During the re positioning of the MTM, while holding the clutch down, the orientation of the end effector of both the PSMs and MTMs in locked. The foot pedal array is shown in figure **??**.

3.1.4 Controllers for the dVRK

To control the MTMs and PSMs, IEEE-1394 Fire Wire Controllers (Fig. 3.7) are used. These controllers have been developed at JHU. The power and cooling systems and the mounting and packaging have been done at WPI. I formed the last link in the chain before these controllers are shipped to other universities for the previous year. I have tested each controller with all of its sub modules before shipping them out to different universities, replacing, reporting issues with the sub-components if any. These controllers provide the necessary control of the motors of the PSMs and MTMs and can report (to PC) and set joint parameters(via PC) at around 6 kHz. Each IEEE-1394 controller consists of two boards, a 1394 FPGA board and a Quad Linear Amplifier QLA Board. the QLA board has interface for driving upto four

Figure 3.6: The Foot Pedal Array

brushed DC motors. Since the MTMs and PSMs have 8 and 7 joints respectively; for each manipulator, two controller boards are required. In case of the MTMs all the eight amplifier channels are used and for the PSMs, 7 channels are used and one remains idle.

The 1394 FPGA boards are designed to retrieve certain parameters over the fire wire to a PC running the control loops. These parameters include variables such as joint currents, joint positions, joint efforts, amplifier temperatures and so on. It should be noted that the control loops are not executed on the boards themselves, but on a separate machine to which the fire wire interface connects to. This design choice allows for using different control algorithms to be executed for a single robot without the need to touching or altering anything on the controllers.

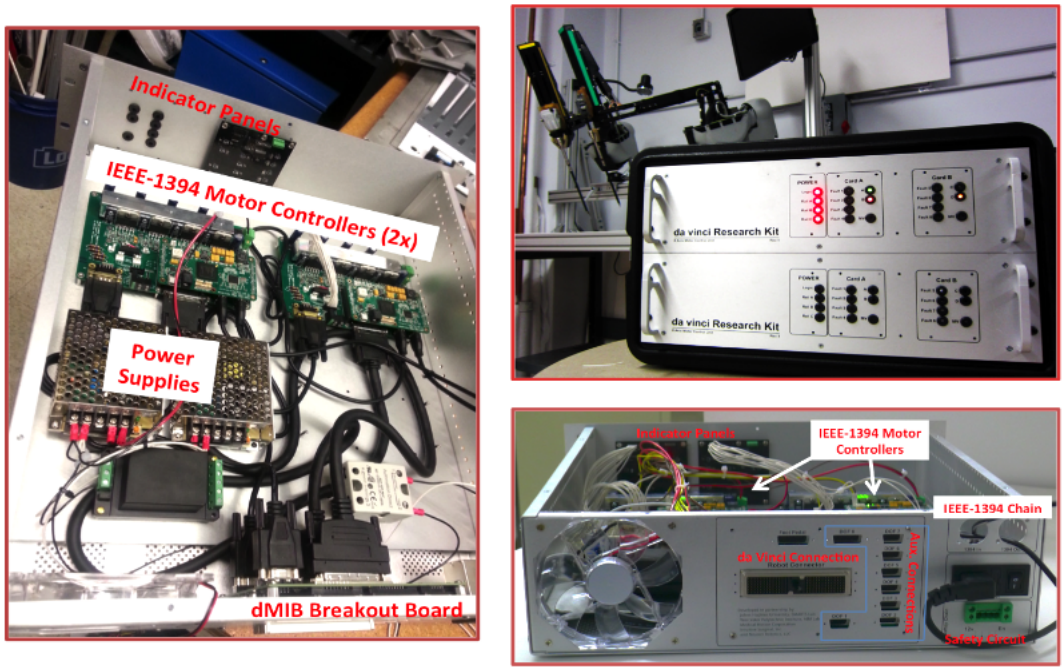


Figure 3.7: A controller for each dVRK manipulator with labeled components

3.2 Software Architecture

The manipulators and the foot pedal tray of the dVRK are connected with the IEEE-1394 Fire Wire controllers. These controllers are connected to each other using the daisy chain connection scheme of the fire wire. One end is connected to a PC. The IEEE-1394 controllers are equipped with FPGA's. The FPGA's are programmed such that the controllers provide the following information to the PC over the fire wire connection:

- Motor Positions read through the potentiometer
- Motor Velocities read through the encoders
- Current feedback for each motor
- Temperature of each amplifier

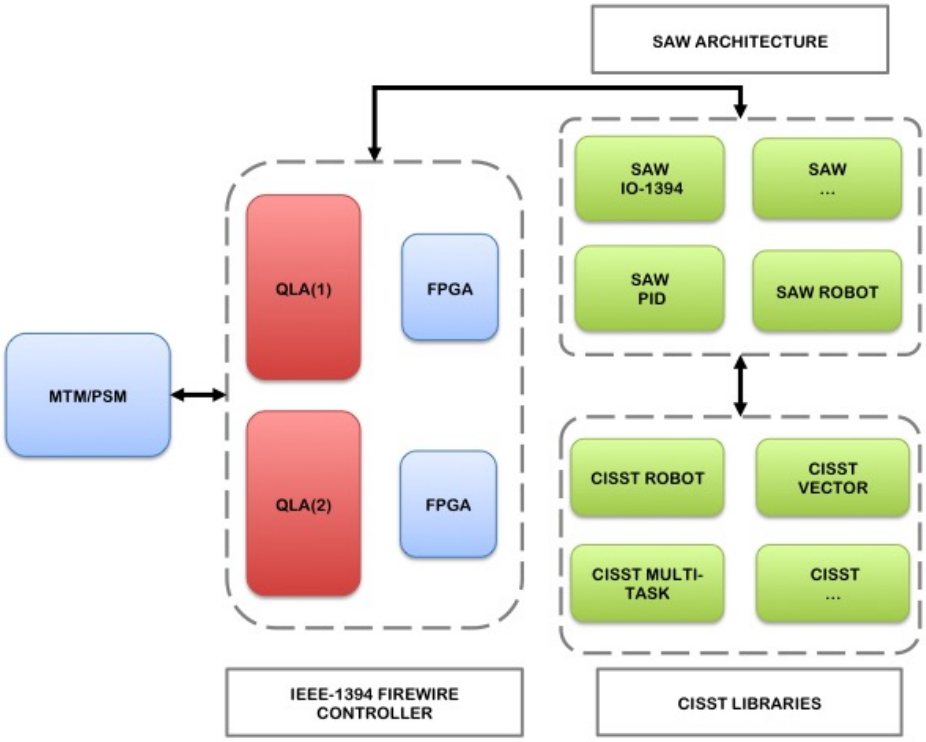


Figure 3.8: The basic system architecture

These parameters are available at the PC end to be read by any application that can access the fire wire port. An application has been developed using the SAW architecture to read these parameters by JHU. The application is called the sawIntuitiveResearchKit. As explained in 2.2, this application also uses many CISST libraries. Figure 3.8 shows a very simple block diagram of a single dVRK manipulator, the IEEE-1394 controller and the CISST/SAW. The sawInuitiveResearchKit application is the result of years of development. It also demonstrates one of the proposed applications, that led to the development of SAW. Using sawIntuitiveResearchKit, the following parameters can be set:

- Motor Positions

- Motor Velocities
- Motor Torques
- Motor Currents

At the back end the SAW libraries are taking care of these inputs and outputs while maintaining abstraction at the users end. For setting the motor positions, PID control laws are implemented in the SAW applications, that can run as high as 6 kHz, which take into account the sensors attached to the corresponding dVRK components for the feedback. Using the SAW application, the values of the P,I and D gains can be set on the fly to change the control strategy. Default values are provided for these parameters that achieves suitable control as the application starts.

Chapter 4

Models for dVRK

The PSMs and MTMs have been modeled in Solid Works to provide simulation models. The modeling has been carried out at both WPI (by Gang Li) and JHU. These models have then been converted into Universal Robot Description Format URDF and SDF files. URDF format is very popular for development in ROS frameworks. The URDF format can be used to model the kinematics and dynamics of the modeled robots. URDF is readily used by Simulation software such as Rviz (kinematic simulator), MoveIt (path planning related packages) [33] and even Gazebo (dynamic simulator)[16]. Gazebo first turns the given URDF format to SDF.

4.1 Development of the URDF files

The work of converting the actual design of the dVRK had been taken up at JHU and WPI at the same time. A model of the dVRK components did not exist and to generate a model from the actual robot relied on mere visual observations and measurements. Since it was inconvenient and rather difficult (both practically and due to the value of the dVRK) to open all the components of the dVRK and model them, many of the internal joint placements had to be improvised based on the

behavior of the robot. This approach was bound to introduce offset errors between the software designed models and the actual robot. Non the less the models have been developed and for this research the models designed at JHU have been used with modifications to remove some of the offset errors.

Once the models had been developed, they were converted to URDF format, using a 3rd party URDF conversion tool called "sw_urdf_exporter", for solidworks models. The URDF files is just an xml format file with a .urdf extension that maintains a tree structure of the robot based of joints. In general, for a serial chained robot, a fixed joint in the world is the first parent. The first link in the robot is the child to this world parent joint. Such a parent-child relationship continues iteratively till the end-effector or the last link the robot.

URDF format keeps modularity in mind with the capability of modeling a robot or a manipulator and a group of different sets. As as example, a humanoid robot could be modeled as a 4 URDFs each of two arms and two legs and one for the head and the torso. Even an individual arm can be modeled by any number of independent URDFs that can be later combined together.

4.1.1 Using Xacro for URDF

Since the dVRK consists of two PSMs and MTMs that are identical to each other, defining each manipulator separately makes it a redundant task. For this purpose, Xacro has been used. Xacro is an extension to the URDF format that allows creation of Macros. For the dVRK components, xacro files have been developed for a generic PSM and an MTM. The xacro files are incorporated in the URDF files which utilize the macros to defined all the joint and link names with a **right** or **left** prefix, depending on whether that arm type is MTM-Right or an MTM-Left. For the PSMs, the prefix **one**, **two**, **three** or **four** are used depending upon which PSM is

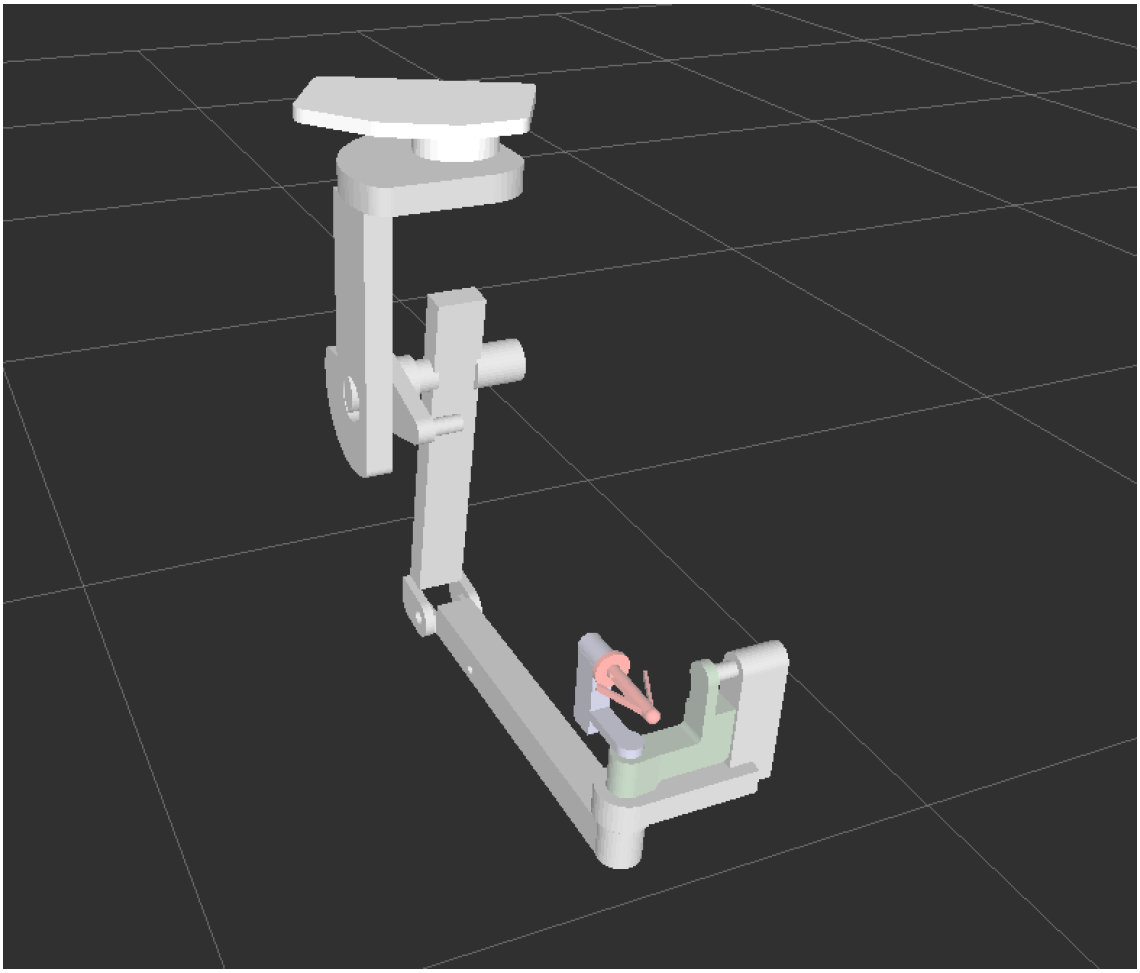


Figure 4.1: RViz model the Master Tool Manipulator

being modeled.

Using these capabilities of xacro, even more PSMs (2 and 3) can be modeled very easily and quickly by changing the name prefix and adding a world offset to the manipulator.

4.2 Spawning the URDF files in RViz

The biggest advantage of modeling the manipulators in URDF file format is visualizing them in RViz. RViz is integrated into ROS and used across the board for

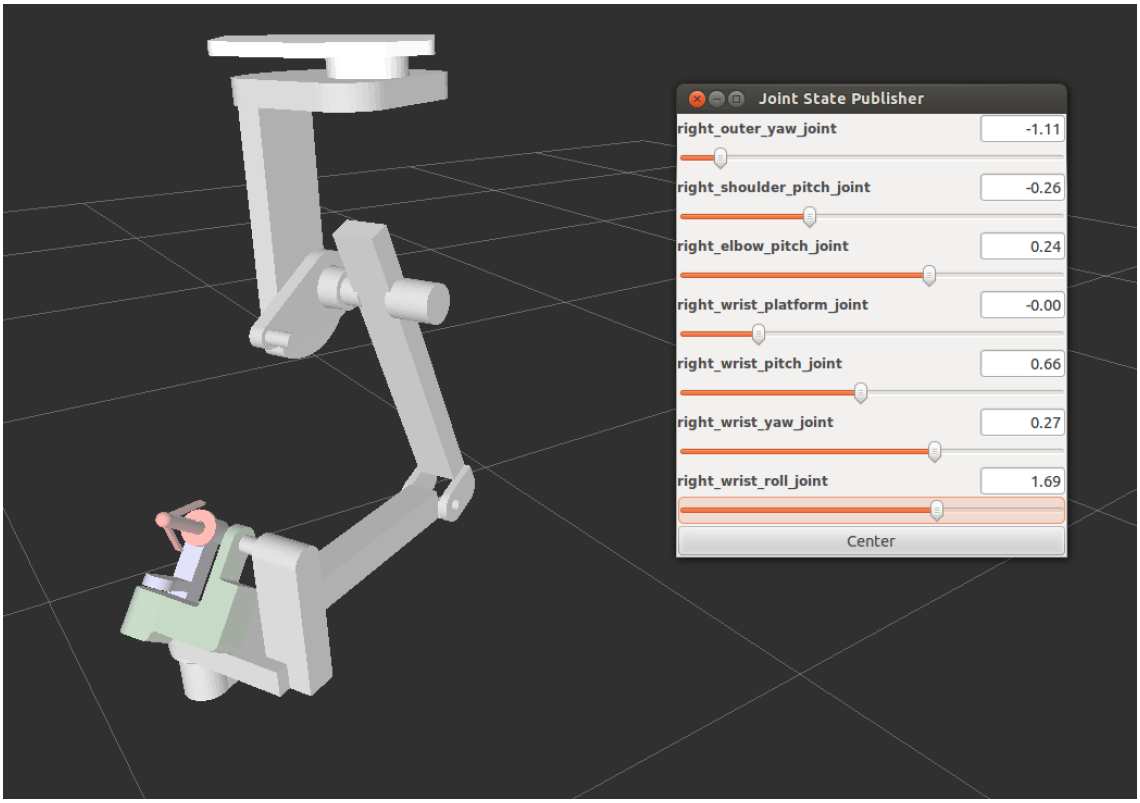


Figure 4.2: MTM is RViz with its Joint State Publisher GUI

atleast the kinematic simulations. Figure 4.1 shows the model of the MTM visualized in RViz. The model is spawned using only the URDF file and the meshes that are linked in the URDF file.

4.2.1 Utilizing the Joint State Publisher Package

I have used the Joint State Publisher Package to allow quick and easy control of the dVRK. This package reads in the URDF file from the ROS server and computes all the variable joints and their names. For the case of MTM and PSM, both the manipulator types have parallel linkages, thus the Joint State Publisher Package takes only the active joints. Active joints are distinguished in the URDF files by specifying a **mimic** tag for non-active joints. Non active joints are the ones that are

not activated directly, such as the renaming joints in a parallel linkage other than the joint have the actuator.

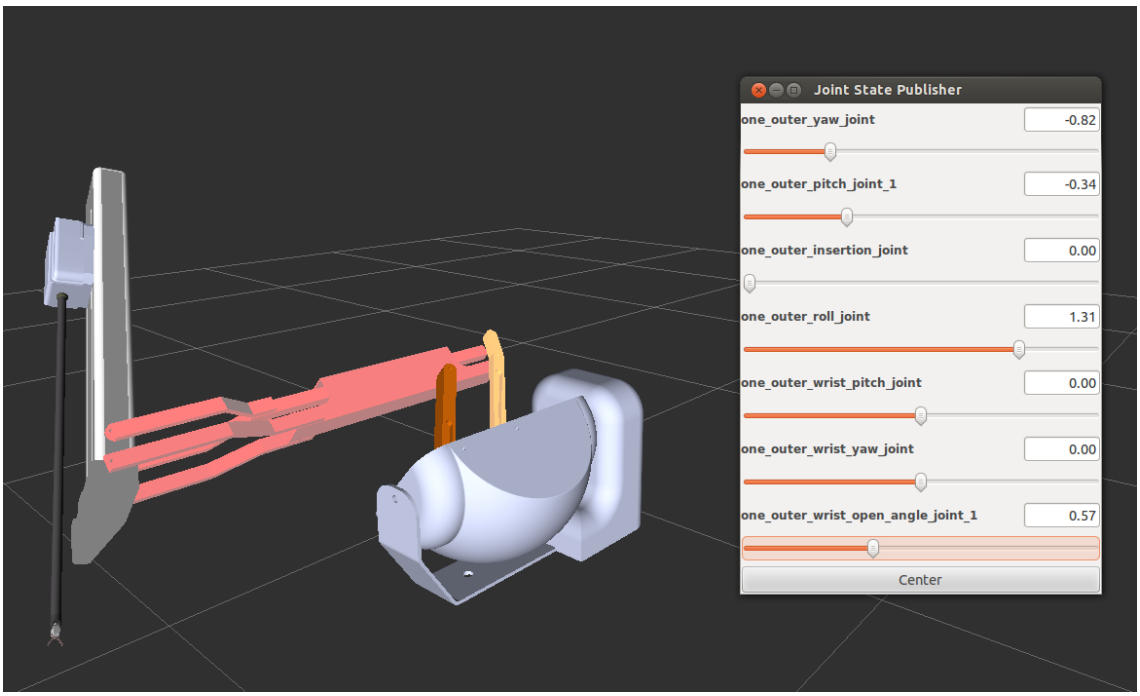


Figure 4.3: RViz model of the PSM with its Joint State Publisher GUI

After reading in the joint names and limits from the URDF files, the Joint State Publisher Package spawns a GUI with sliders to interact with the simulated robot in RViz. Sliding a slider in the GUI moves the corresponding joint in RViz. At the back end, the Joint State Publisher package uses ROS messages of type **sensor_msgs/JointState**. This message type is discussed more in detail in 5.5.1. Figure 4.2 and figure 4.3 show the Joint State Publisher GUI interface with the MTM and PSM simulation models respectively.

Chapter 5

Integration of ROS and CISST/SAW

5.1 Motivation and Requirement

For research purposes and extended control of the dVRK, it was very important to port the control of the dVRK manipulators to a platform that is open and familiar to researchers. The selection of such a platform had to play a huge role in the future development of the dVRK. At our end, Robot Operating System ROS was chosen as the architecture to build upon for extending the capabilities of the dVKR. As mentioned in section 3, the PSMs and MTMs are programmed with the SAW framework by JHU. SAW provides a great platform for the low level control of the dVRK but has several shortcomings for being used at our end.

It is worth mentioning, that a significant amount of work had been put forward in the developed of the SAW architecture and the CISST libraries. The libraries have great compatibility and work coherently with each other for controlling medical robots. Due to the extensive capabilities of the CISST/SAW many base libraries are

used which are essential for development. These base libraries, sometimes, re-derive some core functionalities of standard C++ libraries, such as data types, function parsing etc. The documentation of these base libraries are not as detailed, as the libraries have been developed and used mostly at John Hopkins. To work on the dVRK and come up with experiments and results, a significant understanding of CISST/SAW is required.

It was decided to develop an interface between SAW and ROS rather than dVRK and ROS. This prevented the time and effort being spent on reinventing the wheel and at the same time utilizing the features of SAW and CISST as the base, low level controller and adding ROS as a higher level controller.

5.2 The cisst-ROS Bridge

To cope up with the problem mentioned above, a cisst-ROS bridge has been developed. The bridge was programmed in C++ using the core CISST libraries and compiled with the ROS build system. The initial work of developing this bridge was done by researchers at JHU (Zihan Chen and Anton). The initial build of ROS bridge, provided very brief capabilities to ROS and allowed for retrieving just the joint positions of the PSMs into ROS.

The implementation was bare bones and relied on the specific knowledge of the SAW framework to connect a separate node of ROS to communicate with the SAW internals. Nonetheless, for the very first time, it was possible to get real time joint positions of the PSMs into ROS. This could be used to simulate the models of the PSMs to mimic the actual movements of the PSMs.

Since then I have extended the cisst-ROS bridge to make if much more powerful and bi-directional(setting parameters from ROS rather than just retrieving them) as its supposed to be. The cisst-ROS bridge is created as a component of SAW. Any component in SAW can have required and provided interfaces. Interfaces allow exchange of data between the components. The cisst-ROS bridge is created using the following code syntax.

```
mtsROSBridge::mtsROSBridge(const std::string & componentName, double
    periodInSeconds, bool spin, bool sig)
```

An example brdige that has been used is as follows:

```
mtsROSBridge robotBridge("RobotBridge", 20 * cmn_ms, true);
```

5.3 Revision of the cisst-ROS Bridge

I revised the cisst-ROS bridge to accommodate several new features since it was first designed. Currently, the cisst-ROS bridge is capable of extracting the following parameters from the actual robot.

1. Joint Positions of all the joints of PSMs/MTMs
2. The velocity of each joint of the PSMs/MTMs
3. The torque being applied to each joint of PSMs/MTMs
4. The K_p, K_i and K_d gains for the PID control laws for PSMs/MTMs
5. The current state of Manipulators PSMs/MTMs

Conversely, the following parameters can be set using ROS.

1. Joint Positions of all the joints of PSMs/MTMs
2. The torque being applied to each joint of PSMs/MTMs
3. The K_p, K_i and K_d gains for the PID control laws for PSMs/MTMs
4. The current state (Homing, Homed, Tele-Operation, Idle, Ready etc.) of Manipulators PSMs/MTMs.

Parameters for the dVRK are set using the cisst-ROS Subscriber and parameters are read using cisst-ROS Publisher. This should not be confused with ROS Publisher and ROS subscriber since they work the opposite way. Hence to put things in perspective, a ROS Publisher will connect to a cisst-ROS subscriber to set parameters on the dVRK. Likewise, a ROS Subscriber will connect to a cisst-ROS Publisher to retriever parameters from the dVRK. Additionally, cisst-ROS Events are used to report events such as buttons and footpedal presses to ROS. Each is discussed in more detail in sections 5.4, 5.5 and 5.6.

5.4 The cisst-ROS Publisher

The cisst-ROS publisher is used to publish the dVRK parameters to the software defined, ROS server. These parameters can be read from the terminal or in applications, depending upon the usage. These parameters range from Manipulator Joint Positions, Encoder Values to Torque encountered by the motor at each joint. The syntax for using the cisst-ROS publisher is as follows:

```
bool robotBridge.AddPublisherFromReadCommand<cisstDataType, rosDataType>(
const std::string & interfaceRequiredName,
const std::string & functionName,
const std::string & topicName);
```

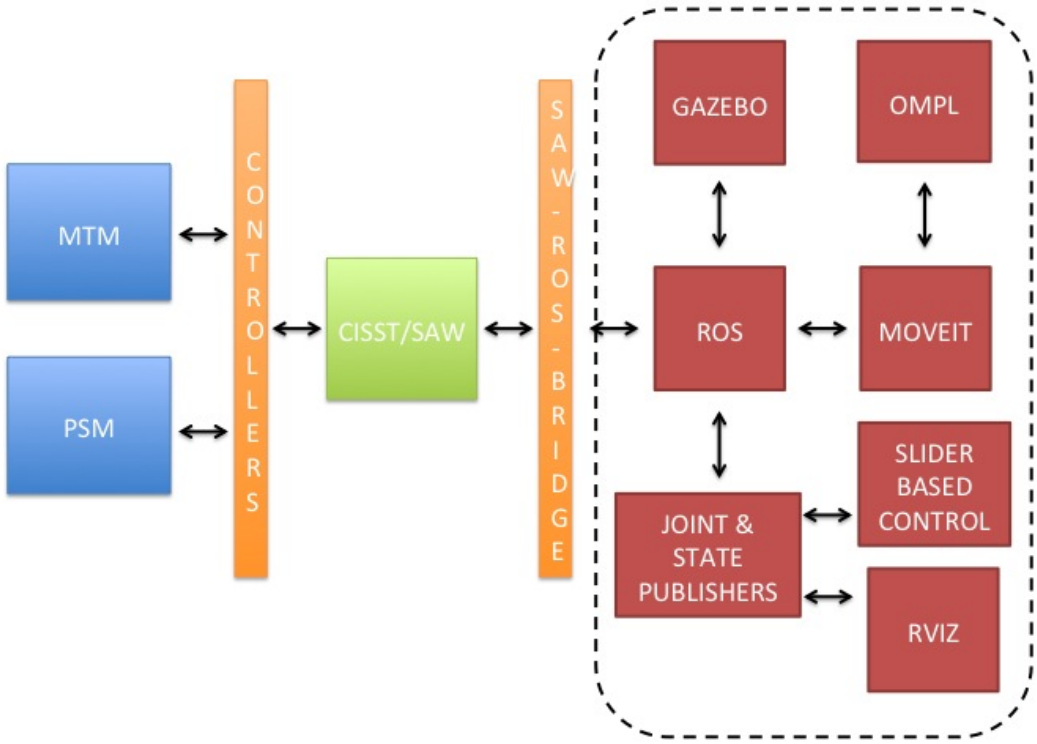


Figure 5.1: RRT-lazy in 3 Dimensional C_{space}

In the function described above, the **interfaceRequiredName** is the name given to the interface by the user. For the dVRK, the **interfaceRequiredName**'s are MTMR, MTML, PSM1 and PSM2 respectively. Of course, any name could be used. The **Required Interface** connects to a **Provided Interface** that is implemented in the CISST libraries.

The **functionName** is the name of the function that is implemented in the **Provided Interface**. This function name is very important and has to follow the name that is actually implemented in the SAW source files for the corresponding interfaces.

Finally, the **topicName** is the name of the topic that would publish the data retrieved. This could be any name. But a naming convention has been used to make

it easier to keep different topics more intuitive.

5.4.1 Publishing Joint Positions

Based on the cisst-ROS Publisher discussed above, the joint positions of the dVRK are taken into ROS using the function:

```
robotBridge.AddPublisherFromReadCommand<prmPositionJointGet,
    sensor_msgs::JointState>(config_name, "GetPositionJoint",
    "/dvrk_mtm/joint_position_current");
```

5.4.2 Publishing End Effector Pose

The 6 DOF Pose or the transform of the end effector of any arm is retrieved in ROS using the following function syntax:

```
robotBridge.AddPublisherFromReadCommand<prmPositionCartesianGet,
    geometry_msgs::Pose>(config_name, "GetPositionCartesian",
    "/dvrk_mtm/cartesian_pose_current");
```

5.4.3 Publishing Joint Torques

For retrieving Joint Torques in ROS, the following function syntax is used:

```
robotBridge.AddPublisherFromReadCommand<vctDoubleVec,
    cisst_msgs::vctDoubleVec>(config_name, "GetEffortJoint",
    "/dvrk_mtm/joint_effort_current");
```

5.4.4 Publishing PID Gains

For the PSMs and MTMs to follow the desired joint values, a PID control law has been implemented in the SAW architecture. This control law uses the PID gains for error correction. The current gain values being used for the control law can be read at the ROS end.

5.5 The cisst-ROS Subscriber

The real achievement of developing the cisst-ROS bridge is the capability of setting the robot parameters from ROS. To make the cisst-ROS bridge bi-directional, the CISST libraries had to modified to accept incoming data from ROS. The parameters can be set using ROS publishers and cisst-ROS subscribers. The general syntax is as follows:

```
bool AddSubscriberToWriteCommand<cisstDataType, rosDataType>(
const std::string & interfaceRequiredName,
const std::string & functionName,
const std::string & topicName);
```

Similar to the cisst-ROS Publisher the **interfaceRequiredName** is the name given to the interface by the user. For the dVRK, the **interfaceRequiredName**'s are MTMR, MTML, PSM1 and PSM2 respectively..

The **functionName** is the name of the function that is implemented in the **Provided Interface**. This function name is very important and has to follow the name that is actually implemented in the SAW source files for the corresponding interfaces.

Finally, the **topicName** is the name of the topic that would publish the data from ROS to the SAW components.

5.5.1 ROS Message Types for cisst-ROS Bridge

For writing the robot parameters from ROS, the following ROS message types are used most often. They are described below for the sake of understanding.

sensor_msgs/JointState

The ROS message type for setting joint positions and efforts is sensor_msgs/JointState. This message type is generic and is used widely across the board in the ROS community. The sensor_msgs/JointState consists of the following generic types for carrying information.

```
{
std_msgs/Header header
string[] name
float64[] position
float64[] velocity
float64[] effort
}
```

Depending upon the usage, one or multiple number of the data fields are used to control the robot. The **string** array always consists of the names of the joints. These joints are the joint names of the corresponding manipulator of the dVRK. Since the dVRK has a parallel linkage in both families of the MTM and PSM manipulators, only the active joints (that engage the entire parallel assembly) are controllable.

Hence the array of **string** name consists of only these active joints.

geometry_msgs

This message type is convinient for carrying task space information as compared to joint space information in the **sensor_msgs** type. During the Tele-Operation mode of the dVRK, the end effector transforms of each MTM are used to control the tool tip frame of the corresponding PSMs thus this ros message comes in handy in emulating those situations. Moreover, due to different control scenarios, it is easier to realize the use of task space trajectories rather than the joint space trajectories. For this purpose, geometry_msgs have been integrated with the cisst-ROS bridge.

The components of the **geometry_msgs::Pose** are as follows:

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

Where **geometry_msgs/Point** consists of :

```
float64 x
float64 y
float64 z
```

and the **geometry_msgs/Quaternion** consists of:

```
float64 x
float64 y
float64 z
float64 w
```

Hence, in a nutshell the **geometry_msgs/Pose** Consists of the positions and

orientation information for a point in 6-DOF space.

5.5.2 Subscriber for setting Joint Positions

The cisst-ROS bridge can be used to set the joint positions for any PSM or MTM. The syntax for constructing such a subscriber is as follows:

```
robotBridge.AddSubscriberToWriteCommand<prmPositionJointSet,
    sensor_msgs::JointState>(
config_name,"SetPositionJoint","/dvrk_mtm/set_position_joint");
```

5.5.3 Subscriber for setting Joint Torques

For controlling the motor torques of the MTMs or the PSMs from ROS, the cisst-ROS subscriber is built by the following syntax:

```
robotBridge.AddSubscriberToWriteCommand<prmForceTorqueJointSet ,
    sensor_msgs::JointState>(
pid->GetName(), "SetTorqueJoint", "/dvrk_mtm/set_joint_effort");
```

5.5.4 Subscriber for setting the Cartesian Pose

The Cartesian Pose can be set likewise, from ROS using the following code syntax for creating a cisst-ROS subscriber:

```
robotBridge.AddSubscriberToWriteCommand<prmPositionCartesianSet,
    geometry_msgs::Pose>(
config_name, "SetPositionCartesian", "/dvrk_mtm/set_position_cartesian");
```

5.6 cisst-ROS Events

While the cisst-ROS Publisher and Subscriber are used to publish and subscribe to parameters such as Joint Positions, Joint Torques, etc. the cisst-ROS Events are used to set or retrieve events that mostly depend on a single button press. These include the event of any of the foot pedal being pressed to a safety clutch being pressed on any PSM. Additionally events that relate to safety measures of the robot such as current violations are also included in the cisst-ROS events.

Two examples of using cisst-ROS events are described below. The first example relates to using the COAG foot pedal being pressed at the MTMs Console.

```
robotBridge.AddPublisherFromEventWrite<prmEventButton, std_msgs::Bool>(
            "Coag","Button","/dvrk_footpedal/coag_state");
```

The second example related the gripper pinch event, where pinching the gripper of the MTM registers an event.

```
robotBridge.AddPublisherFromEventVoid(
            config_name,"GripperPinchEvent","/dvrk_mtm/gripper_pinch_event");
```

In the two examples, two different types of events have been discussed, both of which publish to ROS.

5.7 Connecting Interfaces in SAW Components

Once the cisst-ROS bridge is created as a components in SAW, its interfaces are defined. Each interface has functions that set or retrieve data from other SAW components. For example, getting joint positions would require the corresponding interface in cisst-ROS bridge component to connect to the correct components in-

terface that has the joint positions. For this purpose, the following syntax is used to connect different interfaces after the cisst-ROS bridge component is registered in the component list.

```
bool Connect(const std::string & clientComponentName,
const std::string & clientInterfaceRequiredName,
const std::string & serverComponentName,
const std::string & serverInterfaceProvidedName);
```

Example of the implementation of the above syntax is:

```
componentManager->Connect(robotBridge.GetName(), config_name,
    mtm->GetName(), "Robot");
```

Another example is:

```
componentManager->Connect(robotBridge.GetName(), pid->GetName(),
    pid->GetName(), "Controller");
```

A component can have several interfaces defined for it and each interface can have several functions defined within it.

5.8 Compiling SAW Code with ROS Build

To make the use of cisst-ROS bridge, I have compiled the SAW code with ROS to make ROS executables. These executables take care of setting the cisst-ROS bridge and making the ROS Publishers and Subscribers readily available for use. These executables also set up the entire configuration for running the actual dVRK manipulators and eventually make them operational thus saving the difficulty of launching the SAW applications and ROS applications separately. A brief overview

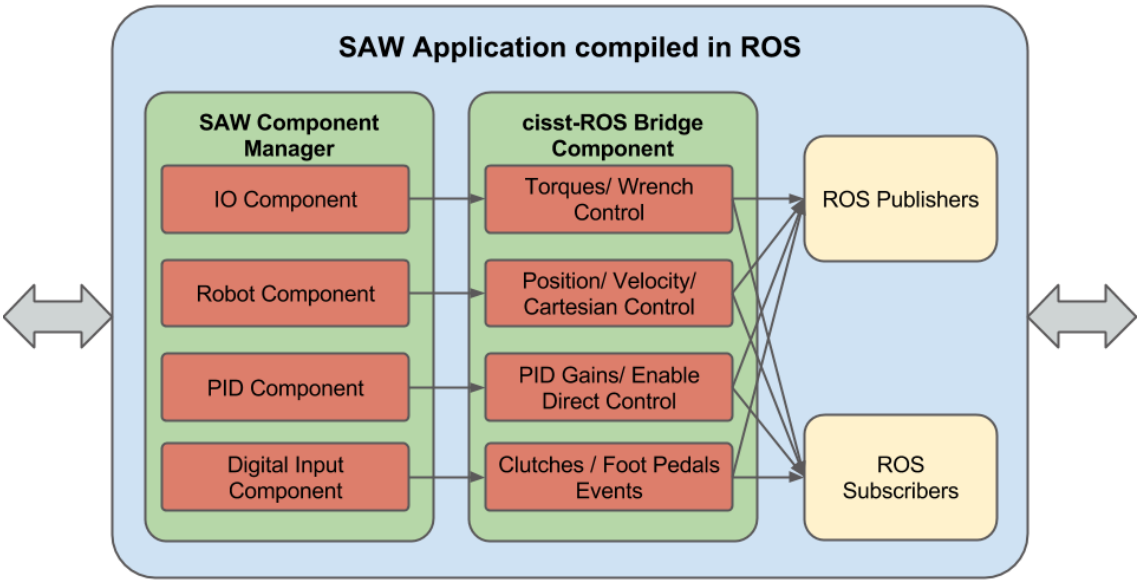


Figure 5.2: SAW application compiled with ROS build system to get a ROS executable. The figure shows the SAW components connected to their corresponding cisst-ROS bridge interfaces to get publishers and subscribers for getting and setting robot parameters

of this compiled executable is shown in figure.

5.9 Discussion

A detailed description of the hardware and software components of the dVRK has been presented in this chapter with examples of the newly created cisst-ROS bridge for interaction with the actual dVRK components. The cisst-ROS bridge has been possible with many extension and additions to the CISST and SAW libraries and I have been at forefront of their development. The dVRK is no longer a monolithic robot with complex software architecture that makes researchers wary of using it, the control has been ported to numerous ROS interfaces so any researcher can pick up the open source code and make experimental analysis with owning a daVinci in their lab.

Chapter 6

Motion Planning for dVRK

6.1 Introduction

In this chapter, the implementation of a motion planning framework for the dVRK has been discussed. The motion planners have been implemented first in Matlab as a proof of concept and then a more general framework has been extended in ROS explicitly for the dVRK manipulators. In Matlab, RRT and its variants have been developed to plan for robot agnostic environments and the comparison between the planners has been presented. For ROS an existing software package has been utilized, using which some experiments have been performed and some results are presented.

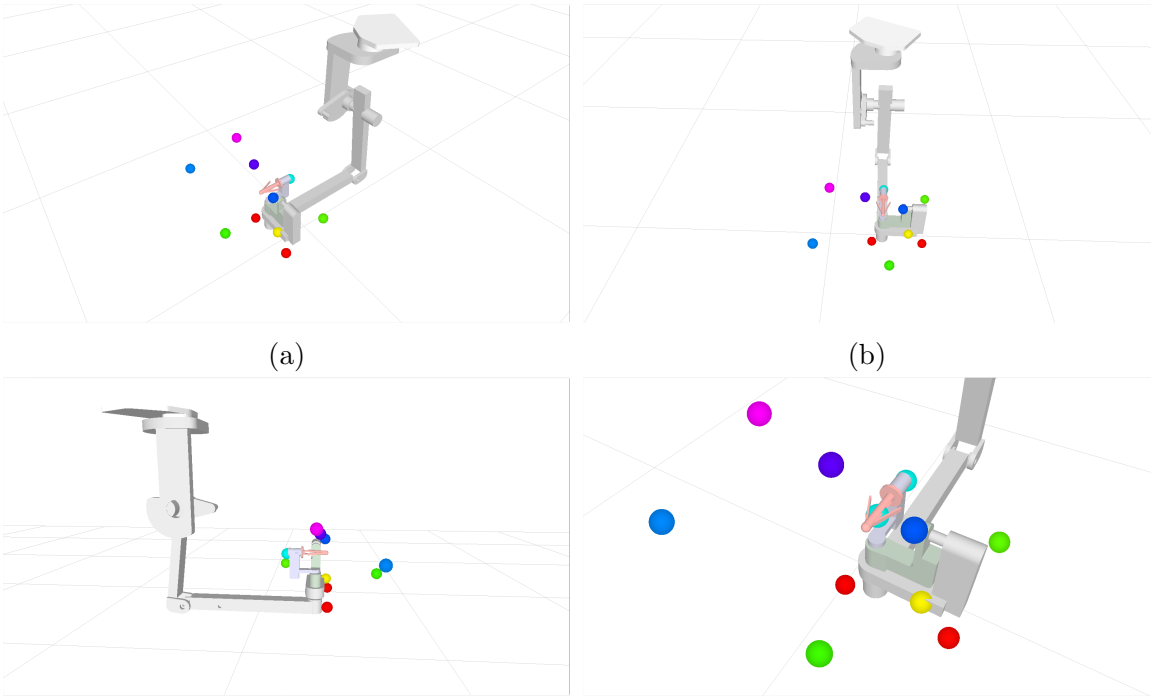
6.2 Motivation

Use of Motion Planning to Robotic Surgery is a novel idea since it has not been applied to any real surgery as of now. Depending upon the different types of surgeries, the use case of Motion Planning is different. For minimally invasive surgeries, the human operator views the surgical area most oftenly, by a pair of stereoscopic

cameras and rarely does the surgeon has the haptic feedback of the operational area. Due to this, the very nature of minimally invasive surgery introduces occlusion of the target organs by other surrounding organs, tissues and skin. From the surgeons point of view, this creates difficulties in spanning around by accidentally touching or damaging organs due to occlusions and loss of haptic feedback .

There could be several different ways of addressing this issue. One way, is by using the simulated models of the PSM and the real time emulation of the surgical area/body part. Using simulation, the surgeon can specify the start and goal points where s/he intends the PSMs to move to. This is done using the MTMs while having a visual feedback from the simulations of PSMs and the surgical area. Once the surgeon is confident about the goal point, s/he can lock these points. The motion planning algorithms can then take over and compute a collision free path between the start and goal pose of the PSM(s), in simulation. The path can be visualized against the operated area so the surgeon can be confident and aware of the approach being taken. Such a procedure can help relieve the accidental contact of the PSMs to the sensitive organs by the tele-operated movements of the PSMs. This procedure does require accurate models of the PSMs, the patient's operated area and the knowledge of the transformation between the two.

This idea can be extended to include more than a pair of start and goal points. The surgeon could specify several points, creating a chain, that leads to an abstract path emanating from the start point and following through towards the goal point. The algorithms should take care of realizing the validity of the points chosen by the surgeon. There could be additional validations, including whether any of the chosen points are within the achievable work space, each point itself is in not in collision



(a) (b)

(c) (d)

Figure 6.1: Four different views of the points sampled by the MTM for assistive path planning

with any organ and the goal point can be achieved by a collision free pose of not just the end effector, but the entire PSM. More validations can be introduced depending upon the need.

Motion planning in this sense will be hereby termed as assistive path planning. The use of word "assistive" merely refers to the additional supervisory control, over the path produced by the path planners. It should be noted that this particular implementation considers only the simulation of the movement of PSMs and not the control of the actual PSMs by the algorithms themselves.

6.3 Using Random Time Planners

The planning algorithms that are used for path planning are random time planners. Random Time Planners guarantee a solution if one exists only if the planners are run for infinite time[20, Ch. 5, p. 185-186]. The use of random time planners is recommended in problems where the configuration space is of higher dimensions. Having 7 and 6 DOF configuration spaces for the MTMs and PSMs respectively, the use of random time planners is justified. Considering the way random time planners work, there is no guarantee that the algorithm would converge to a solution in a given amount of time. It might also happen that a solution might not exist at all and the planner still keeps running. This property makes random time planners tricky to use in theory. However in many real world applications, such a condition of convergence does not come to test, as a solution is found without the planner running for significantly longer periods.

6.4 Initial Development in Matlab

6.4.1 Selection of Matlab for Development

A significant amount of work has been done in matlab to visualize, validify and analyse the results of planning in configuration space. RRT and its several variants have been implemented. The planners are implemented in configurational spaces of 2 and 3 dimensions but can be easily extended to plan for higher dimensional spaces. The code is scale-able and modular thus further research into the use of Matlab for motion planning is possible.

I chose Matlab for developing the planners at first due to the ease of development

and since there was no framework available for the control of dVRK manipulators except CISST/SAW at that time. In a short span of time the planners were programmed and analyzed using the simple-to-use, yet powerful visualization tools. Additionally, the availability of Matlab on all major Operating Systems made the code functional without any modifications.

6.4.2 Implementation in Matlab

To start things off, the traditional RRT was implemented in Matlab. The planner was developed to work both in 2 and 3 Dimensional configuration spaces. The kinematic structure of the manipulators was abstracted away from the planner. The space is bounded explicitly, with a start and a goal state provided to the planner at the start. The details of implementation of each variant of RRT is discussed below.

RRT (RRT Extend)

The basic RRT planner is also known as the RRT-extend planner. The RRT-extend planner, initiates with a start state q_{start} and a goal state q_{goal} in a d dimensional configuration space. The planner starts filling up the free configuration space C_{free} and keeps on building the tree incrementally, till one node from the tree connects to the q_{goal} or falls within the specified $GoalRegion$[6]. This triggers the Planner Termination Condition. The Planner Termination Condition (PTC) can be composed of a number of things, most basically the tree reaching the goal region and a valid branch leading upto that q_{goal}, additionally the PTC can include the time allowed for the planner to plan and the number of states to be explored. If either of the three condition is met, the planner terminates.

Additional termination conditions can certainly be planned that include a limit

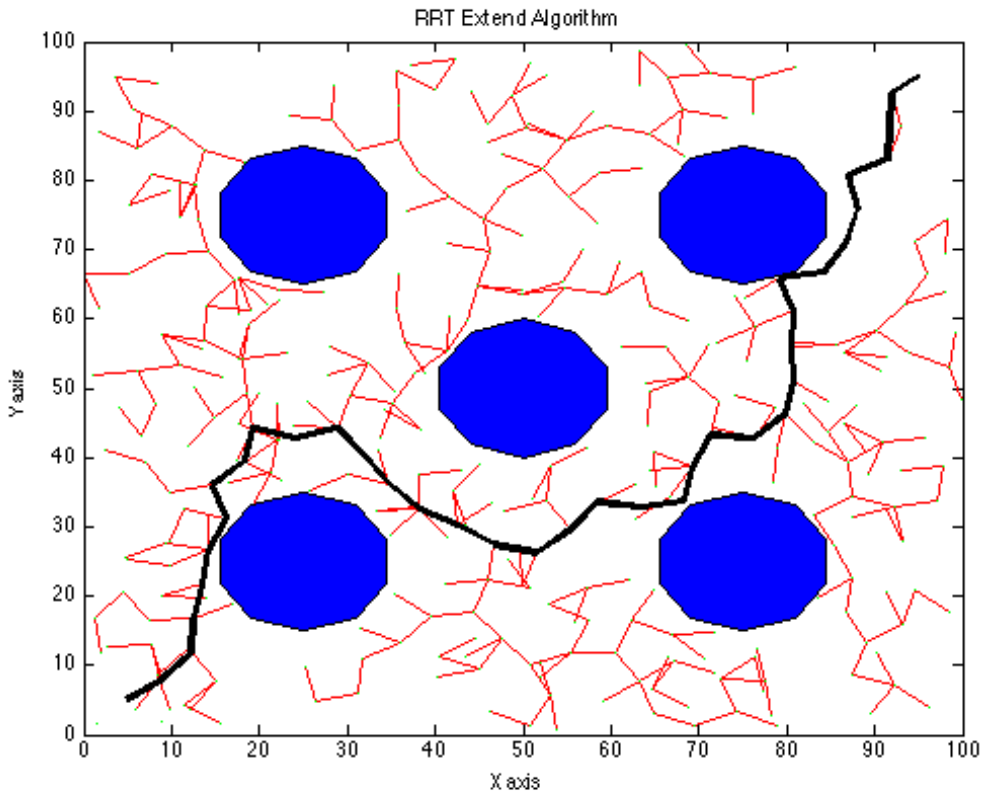


Figure 6.2: RRT-extend in 2 Dimensional shows a path (in black) found between pair of start and goal points while avoiding the obstacles in C_{space}

to the overall cost incurred by the planner and so on. The Planner Termination Condition function has been implemented as a polling function which keeps on checking for the PTC after every new state in sampled.

For an existing tree, the planner samples a random state q_{rand} in the d dimensional configuration space. The function $NearestNeighbour$ searches for the nearest neighbor in the $Tree$ to q_{rand} based on a pre-specified distance metric. Since the configuration space is a d dimensional Euclidean space, the Euclidean distance metric is justified in its use. This function thus returns the state $q_{nearestNeighbor}$ in the $Tree$ that minimizes the Euclidean distance metric between the state q_{tree} and q_{rand}.

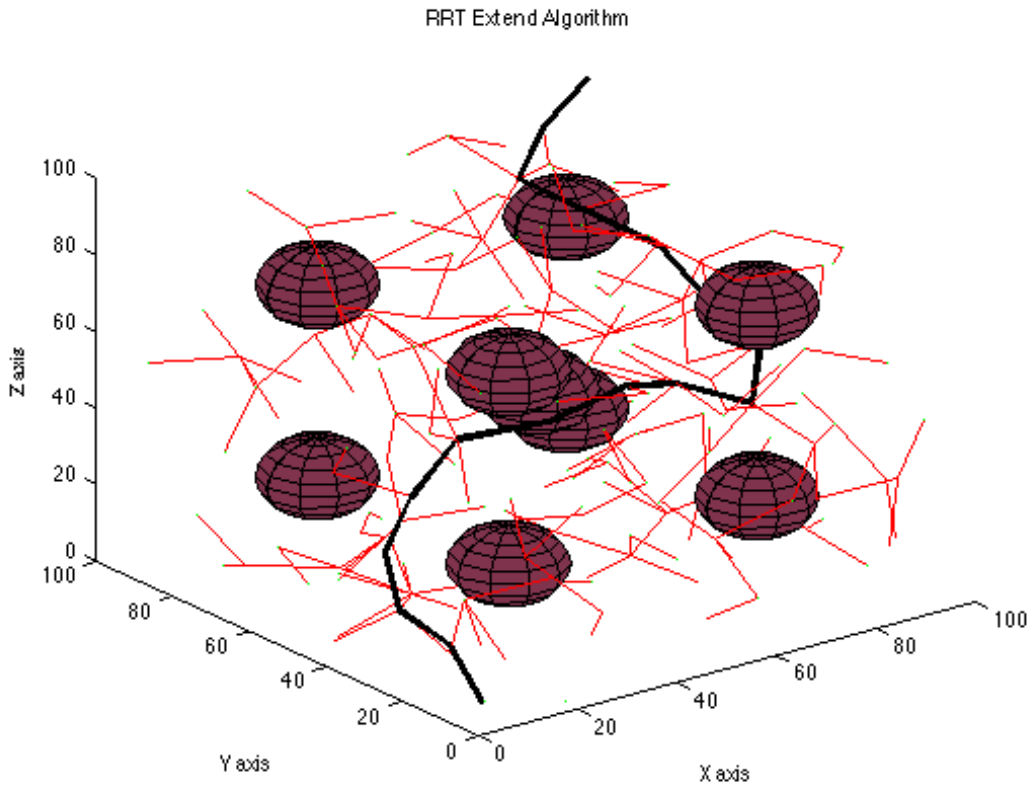


Figure 6.3: RRT-extend in 3 Dimensional shows a path (in black) found between pair of start and goal points while avoiding the obstacles in C_{space}

The next step is to propagate from $q_{nearestNeighbour}$ in the direction of q_{rand}. The function $Extend$ is used for this propagation. $Extend$ takes $q_{nearestNeighbor}$, q_{rand} and a pre-specified step size as input arguments. The extend function thus produces a new state q_{new} in the direction of q_{rand} from $q_{nearestNeigbour}$. The distance corresponding to the chosen metric from $q_{nearestNeighbor}$ to q_{new} is equal to the step size. When q_{new} has been computed, the function $StateValidation(q_{new}, q_{nearestNeighbor}, C_{obstacle})$ is called. The function checks for collision and if the two states are collision free, q_{new} is added to the tree. The whole process is repeated until a node from the $Tree$ falls inside the goal region or achieves any other condition mentioned to the planner termination. The algorithm is summed up in Algorithm 1.

Algorithm 1 RRT Extend

1: $RRTextend(q_{start}, q_{goal})$
2: $Edges \leftarrow q_{start}$
3: $Tree \leftarrow Edges$
4: **while** Planner Termination Condition = **false do**
5: $q_{rand} \leftarrow Sample(C_{free})$
6: $q_{nearestNeighbour} \leftarrow NearestNeighbour(Tree, q_{rand})$
7: $q_{new} \leftarrow Extend(q_{nearestNeighbour}, q_{rand}, h)$
8: **if** $StateValidition(q_{new}, C_{obstacle})$ = **true then**
9: $Edges \leftarrow q_{new}$
10: $Tree \leftarrow Edges$
11: **end if**
12: **end while**

Algorithm 2 Extend

1: $Extend(q_{nearestNeighbour}, q_{rand}, h)$
2: $D = MetricDistance(q_{nearestNeighbour}, q_{rand})$
3: **if** $D > h$ **then**
4: $q_{new} = q_{nearestNeighbour} + (D/||D||) \times h$
5: **else**
6: $q_{new} = q_{rand}$
7: **end if**
8: $return \quad q_{new}$

The results for the planner in 2 and 3 Dimensions are shown in figure 6.2 and figure 6.3. The black path shows the valid path leading from start state to goal state, the colored spheres are the obstacles and the red lines show the explored states.

RRT Connect

The RRT-Connect planner is an extension of the RRT-extend planner [17]. The only modification lies in the *Extend* function where instead of adding one new state q_{new} to the tree after each q_{rand} is sampled, the planner tries to connect $q_{nearestNeighbor}$ to q_{rand} incrementally by adding states distanced by the step size h in the d dimensional configuration space. The *Propogate* function is used for this purpose 4. If an obstacle is encountered during the propagation the planner stops and adds the last

valid state to the *Edges*.

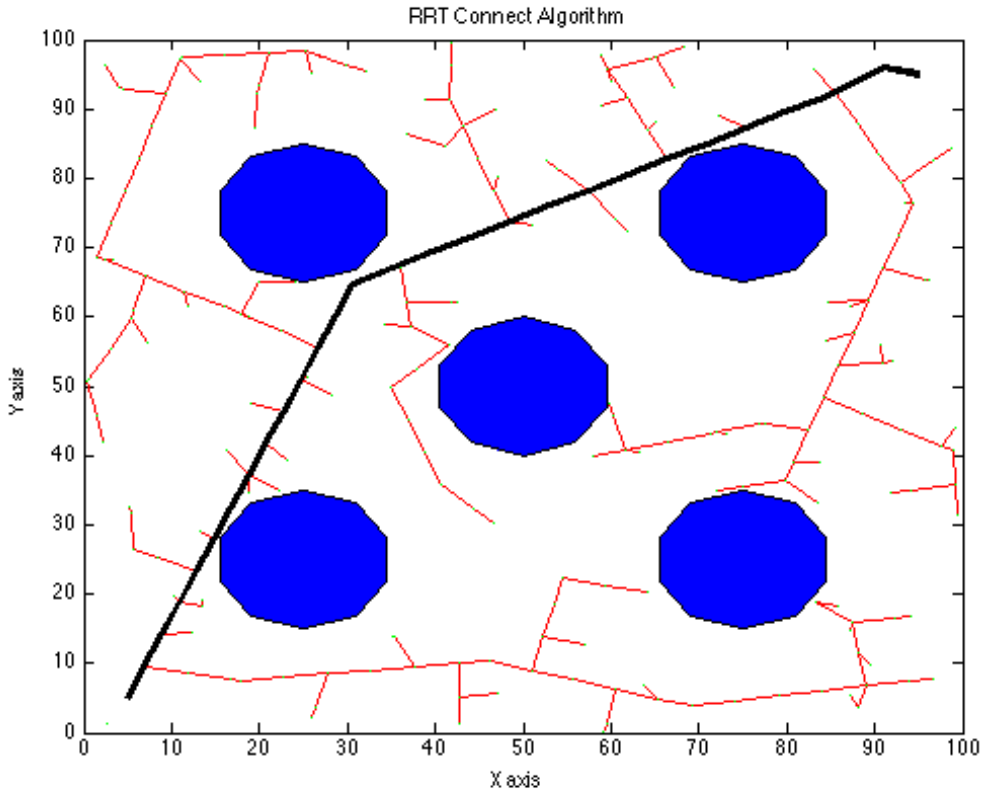


Figure 6.4: RRT-connect in 2 Dimensional shows a path (in black) found between pair of start and goal points while avoiding the obstacles in C_{space}

RRT Lazy

The Lazy RRT planner can be implemented using either the RRT-connect or RRT-extend algorithm. However, in literature, a method similar to RRT-extend is used [18]. The key difference is that in Lazy-RRT, the planner tries to connect the start state to the goal state without checking for collisions. Once a path to the goal is found, the planner checks the path for collisions. The segments of the path that are in collision are repaired using a *RepairSegment* function. *RepairSegment* is implemented such that each of two states at the start and end of a broken segment

Algorithm 3 Extend for RRT Connect

1: $Extend(q_{nearestNeighbour}, q_{rand}, h)$
2: $D = MetricDistance(q_{nearestNeighbour}, q_{rand})$
3: **if** $D \geq h$ **then**
4: $\quad Propogate(q_{nearestNeighbour}, q_{rand}, h)$
5: **else if** $StateValidition(q_{rand}, C_{obstacle}) =$ **true then**
6: $\quad q_{new} = q_{rand}$
7: $\quad Edges \leftarrow q_{new}$
8: **end if**
9: $return \quad Edges$

Algorithm 4 Propagate for RRT Connect

1: $Propogate(q_{seed}, q_{rand}, h)$
2: $q_{new} = q_{seed} + (D/||D||) \times h$
3: **while** $D \geq h$ **and** $StateValidition(q_{new}, C_{obstacle}) =$ **true do**
4: $\quad Edges \leftarrow q_{new}$
5: $\quad D = MetricDistance(q_{new}, q_{rand})$
6: $\quad q_{seed} = q_{new}$
7: $\quad q_{new} = q_{seed} + (D/||D||) \times h$
8: **end while**
9: $return \quad Edges$

are considered as a pair of nested start and goal states. *RepairSegment* then uses RRT-extend to connect the two broken states in the path using state validation as each q_{rand} is sampled. This is done for each segment of the path in collision.

The motivation behind the use of Lazy RRT is the effect of collision checking on the computational cost of the planner. Depending upon the topology of the C_{free} and $C_{obstacle}$, lazy RRT could prove to provide a solution with a siginifcant reduction if computational costs. While at the same time for different C_{free} and $C_{obstacle}$, it could incur as much or even more cost as the other RRT variants. Thus the advantage of using Lazy RRT is case based.

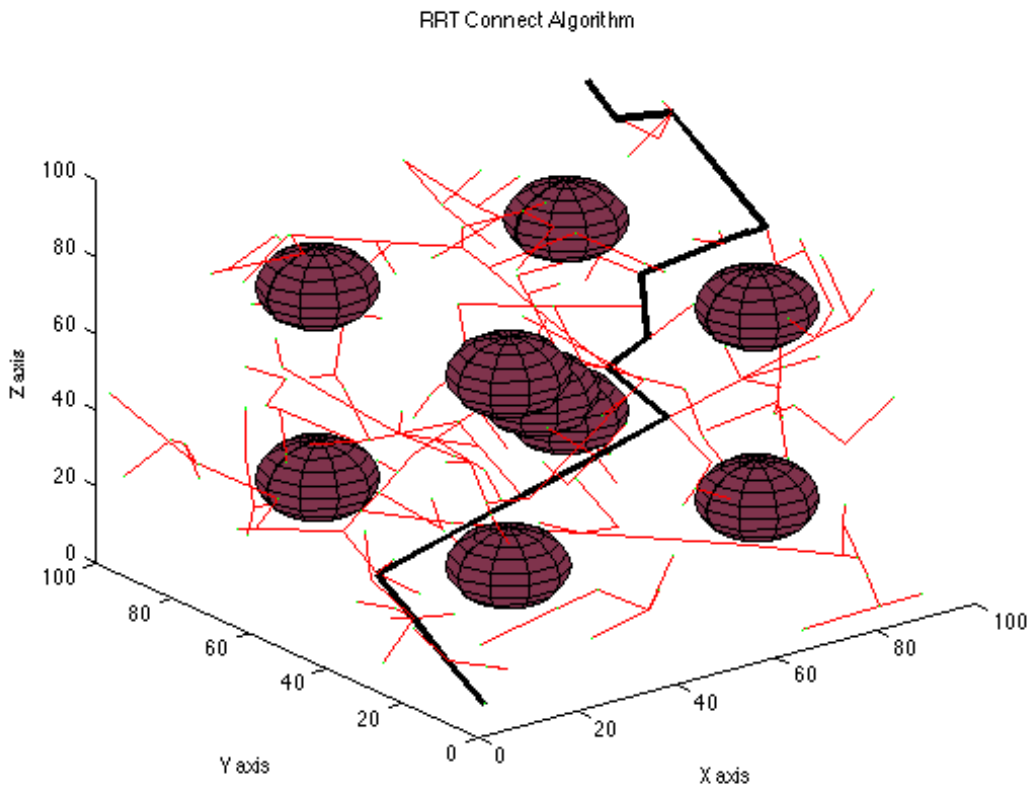


Figure 6.5: RRT-connect in 3 Dimensional shows a path (in black) found between pair of start and goal points while avoiding the obstacles in C_{space}

RRT* Planner

As discussed in [12] and [13] the RRT planner almost always produces a non-optimal path. The initial study to improve the path produced by the RRT planner towards optimality was to continue producing samples even after the goal has been found [13]. These new samples would then be used to make new branches in the tree thereby producing a relatively more optimal path. Such a scheme does not improve upon the path produced [12].

The RRT* planner was thus introduced with a novel technique, involving a different criteria for connecting new states to the tree and re-organizing the already built tree.

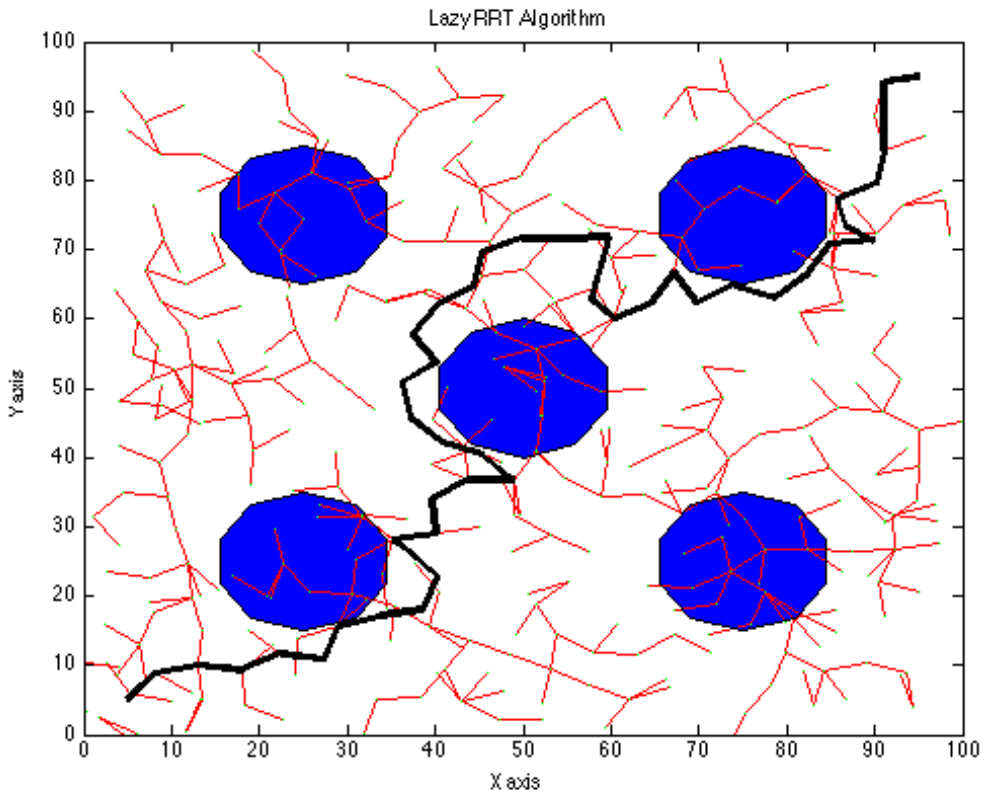


Figure 6.6: Lazy RRT in 2 Dimensional shows a path (in black) found between pair of start and goal points while avoiding the obstacles in C_{space}. The states in red protrude into the obstacles showing that RRT-Lazy does not account for obstacles until the goal has been reached

IMPLEMENTATION

The implementation of the RRT* planner can be broken down into two separate steps. Both steps are discussed below:

The first step involves sampling a random state q_{rand} in the d dimensional configuration space. The function $NearestNeighbor$ returns the nearest state in the tree to q_{rand} based on a distance metric (Euclidean distance metric in our case). The function $Extend$ propagates from $q_{nearestNeighbor}$ in the direction of q_{rand}, producing q_{new}. Up to this point the algorithm in pretty similar to the traditional RRT planner. Once q_{new} is computed the function $StatesInBall$ is called. $StatesInBall$

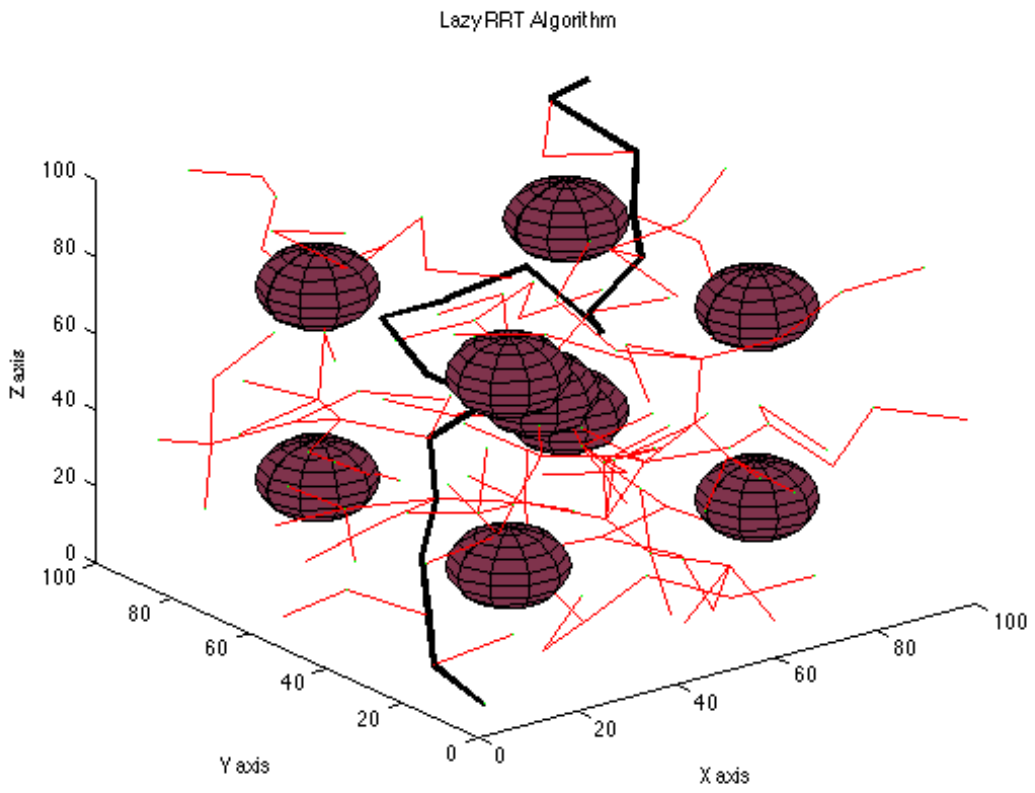
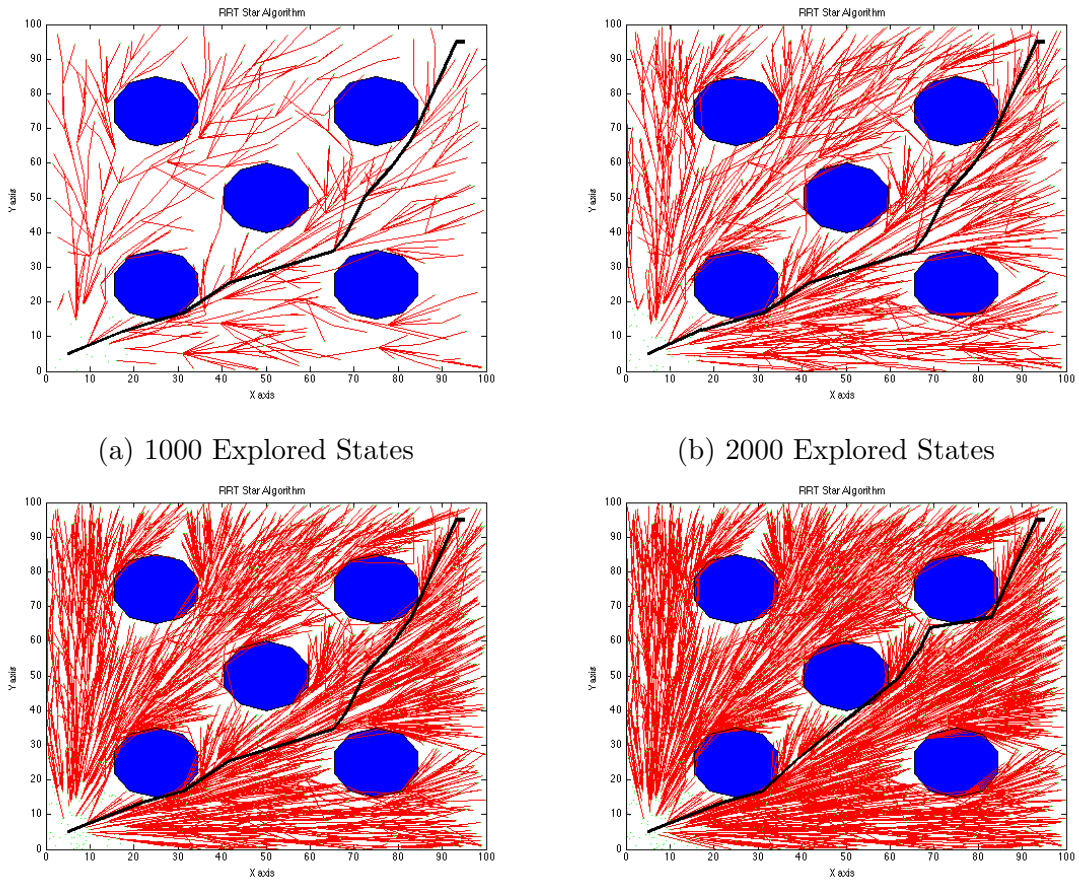


Figure 6.7: RRT-lazy in 3 Dimensional shows a path (in black) found between pair of start and goal points while avoiding the obstacles in C_{space}. The states in red protrude into the obstacles showing that RRT-Lazy does not account for obstacles until the goal has been reached

takes as an input, the state q_{new} and searches for all the states in the $Tree$ that are inside a d-dimensional ball of specified radius r_n, centered at q_{new}. r_n is calculated from equation 6.1. The enclosed states are stored in a vector Q_{near}.

Once the vector Q_{near} is computed the function $MinimumCostConnection$ is called. This function searches in the vector Q_{near} for the state that minimizes the $CostMetric$ while connecting to q_{new}. The cost metric is described in the section 6.4.2. The function $MinimumCostConnection$ thus returns a state q_{min} from the vector Q_{near} that has the minimum accumulated connection cost for connecting to



(a) 1000 Explored States

(b) 2000 Explored States

(c) 3000 Explored States

(d) 4000 Explored States

Figure 6.8: Subfigures (a), (b), (c) and (d) showing the Rewiring of Path and Explored States as more and more States are sampled

q_{new}. The function $Collision(q_{new}, q_{min})$ checks if the two states have a collision free path and if true a connection is made from q_{min} to q_{new}.

The second step of the RRT* planner is to check for the cost of all the states in the vector Q_{near}. These costs are compared to the cost of q_{new} added with the $LineCost$ from q_{new} to the corresponding state in Q_{near}. If the cost of q_{near} is greater than the cost of q_{new} + $LineCost$, the corresponding state is rewired such that its is assigned q_{new} as the new parent and its previous connection with the older parent is removed. The function $LineCost$, computes the Euclidean distance in d-dimensions

as the line cost between the two given states.

The special parameter of the RRT* algorithm is the radius r_n. The radius is calculated using Equation 6.1.

$$r_n = min\left(\left(\frac{\gamma \log n}{\zeta_d n}\right)^{1/d}, \eta\right) \tag{6.1}$$

In equation 6.1 η and γ are use set parameters, ζ^d represents the volume of a ball in d dimensional C_{space}. The radius can be set directly in the RRT* planner. A larger value gives a much more optimal path in relatively smaller number of iterations but increases computational time, a relatively smaller value takes greater number of iterations to produce on optimal path. In actual implementation, the tuning of r_n can be achieved by trial and error.

THE COST/LINE-COST FUNCTION

The cost or the line cost function computes the Euclidean distance between the given two states and assigns it as a cost. For any three states, the are not collinear, the triangle in-equality theorem holds.

6.4.3 Using Random Environment

The examples and figures above are for pre-defined environment with fixed number of pre-determined obstacles. However, the implementation of random environments has been also done. The user can specify the use of either the predetermined environment or random environment.

Figure 6.10 shows both the 2 and 3 Dimensional Environment cases for all the RRT variants. It is interesting to note the behavior of RRT connect in Figure6.10 (e) and (f), as it produces shorter and less jagged paths as compared to RRT Lazy

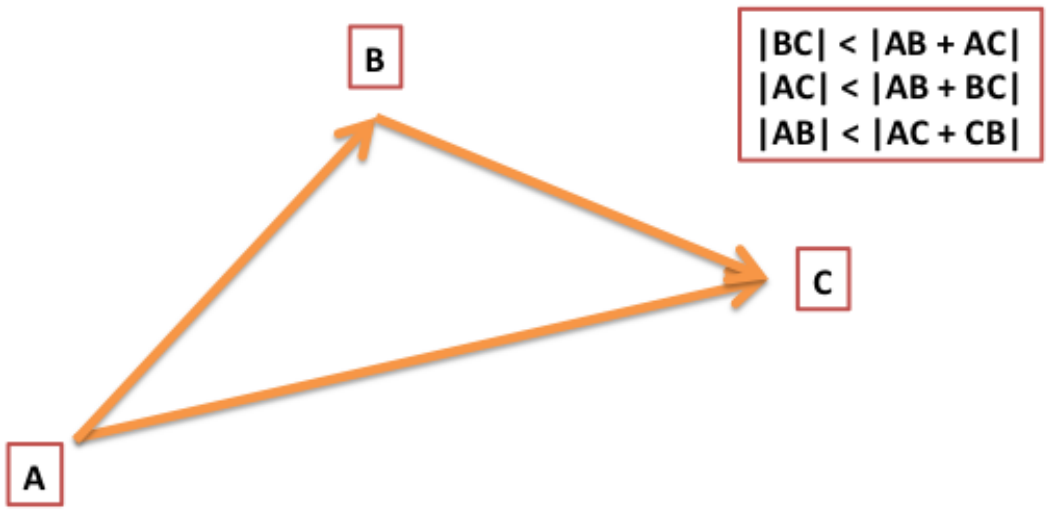


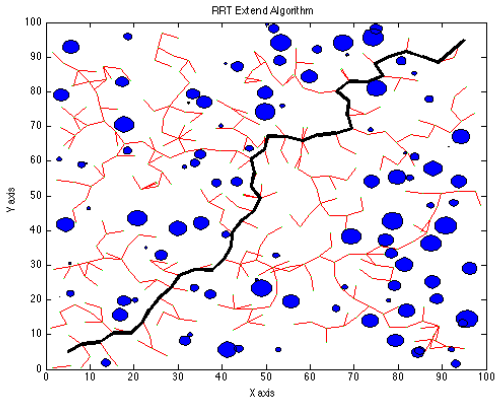
Figure 6.9: The Triangle Inequality

and RRT Extend.

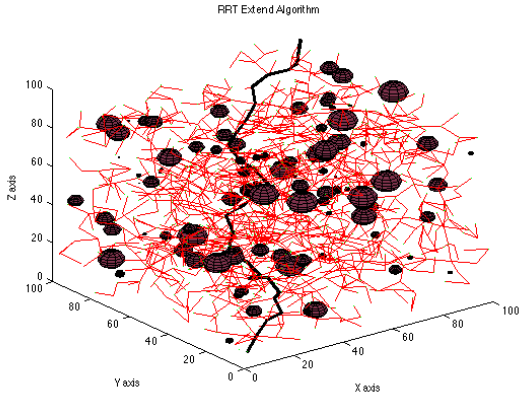
6.4.4 State Validation/Obstacle Avoidance

For all the planners developed in Matlab, the function $StateValidation$ validates two states by checking whether the shortest path between them, in the d dimensions, is collision free or not. For the Matlab implementation, the obstacles are explicit functions of balls or various radii in d dimensional configuration space. The choice of such obstacles does not account for realistic obstacles environments, but provides a an easy to compute model for bench marking the planners.

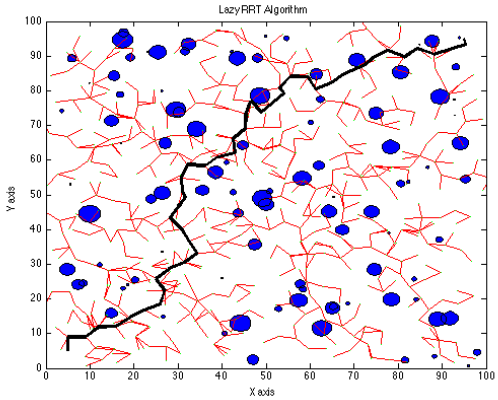
There are two modes for each environment, one with a predefined placement of pre-specified radii of obstacles, and the other randomizes the process by placing the obstacles of random radii at random places.



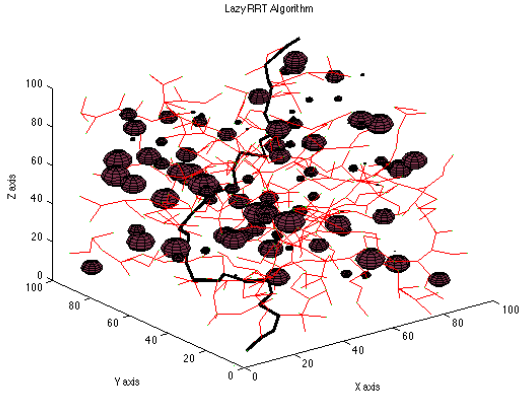
(a) RRT Extend in 2D



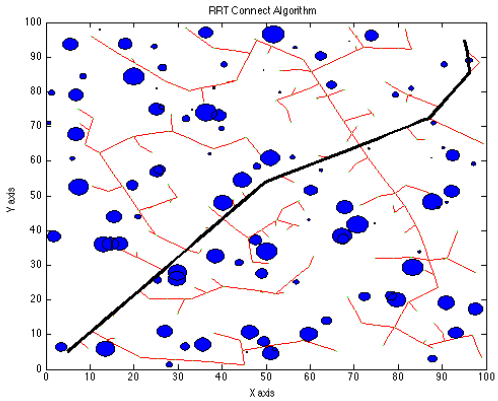
(b) RRT Extend in 3D



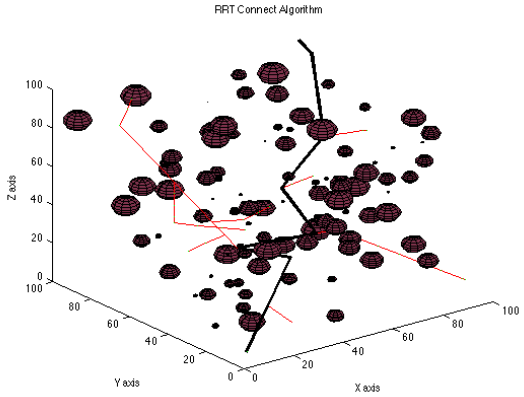
(c) RRT Lazy in 2D



(d) RRT Lazy in 3D



(e) RRT Connect in 2D



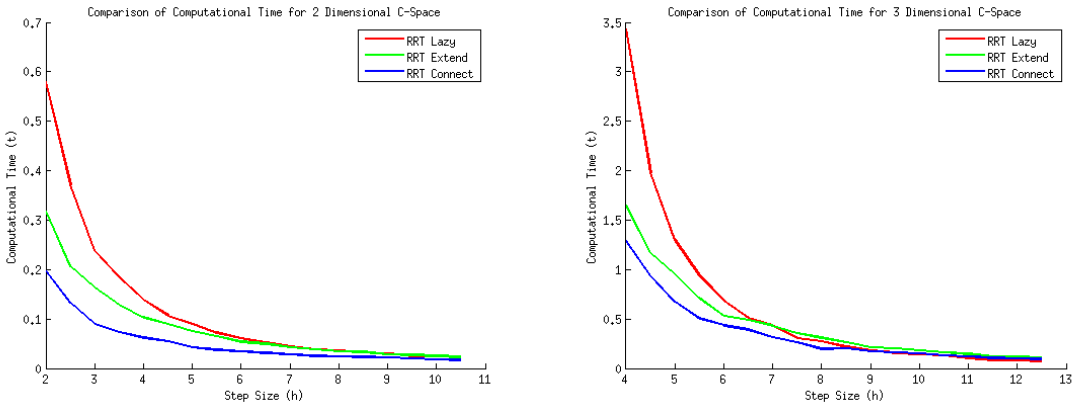
(f) RRT Connect in 3D

Figure 6.10: Random Environment implementation of RRT variants

6.4.5 Comparison of Various Planners

In the implementation of the RRT planner and its variant the control variable is the step size h. The bench marking of the planners with the different step sizes is

shown in figure 6.11 and 6.12. The results are presented for 2 and 3 dimensions with a sample size of 300 runs.



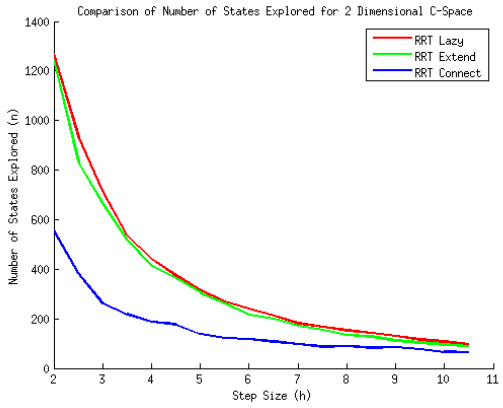
(a) Average Time in 2 Dimensions

(b) Average Time in 3 Dimensions

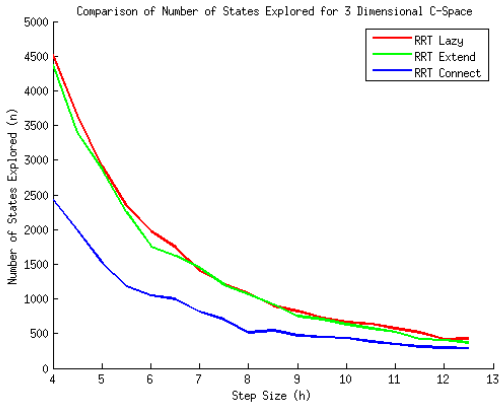
Figure 6.11: Comparision of computational time for RRT-Lazy, RRT-Connect and RRT-Extend in Matlab

Figure 6.11 shows that time for all planners decreases exponentially as step size is increased. For a sample of 300 Runs, it is clearly visible that for smaller step sizes, RRT-Lazy is computationally much expensive, taking more than double the amount of time taken by RRT-Extend and RRT-Connect. RRT connect seems to be the most computationally efficient planner for the C_space chosen. At bigger step sizes, the difference between the computational time of planners seems to diminish.

The comparison between the number of states explored before finding the path is shown is figure 6.12. Similar to the comparison for computational time, the number of states explored, decay exponentially with increasing the step size. In this comparison, RRT-Connect consistently finds the solution with minimum number of states explored for the given C_space. The difference is quite significant as for smaller step sizes, RRT-Connect explores less than half the number of states explored by either RRT-Extend or RRT-Connect.

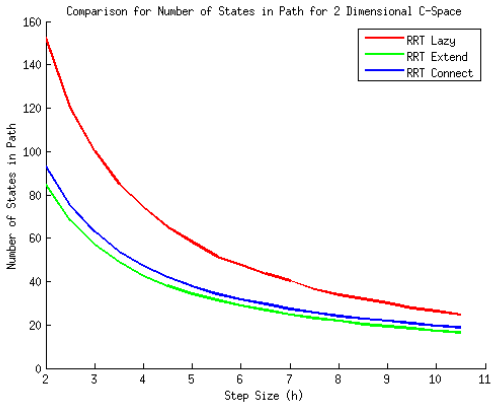


(a) Average Number of States Explored in 2 Dimensions

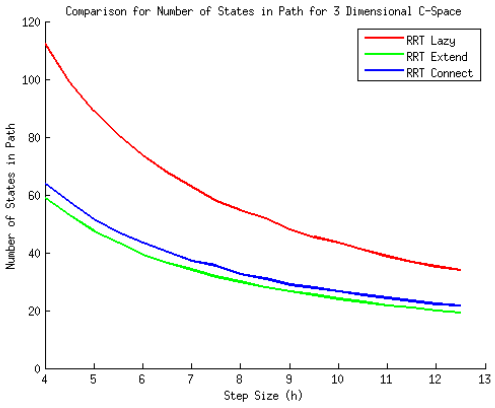


(b) Average Number of States Explored in 3 Dimensions

Figure 6.12: Comparision of number of states explored for RRT-Lazy, RRT-Connect and RRT-Extend in Matlab



(a) Number of states in Path 2D



(b) Number of states in Path 3D

Figure 6.13: Comparison of the number of states present in the computed path

The third comparison is the number of states the make up the path in both 2 and 3 Dimensions. This comparison might not have a direct relation to the other two comparisons but provides an insight to the planners themselves. As evident from the figure 6.13, RRT Lazy almost always produces the path with most number of states. This could be attributed to the *RepairSegment* function that repairs that path and adds more states to the path. This is however, highly dependent upon the

the C_space, and the results might not hold for different configurations and obstacle environments.

6.4.6 Limitations of using Matlab for Motion Planning with ROS

The development and testing environment in Matlab provided a quick and easy setup to implement various motion planner and use them to visualize results. However, following are the reason for which Matlab is not suited for carrying out motion planning work and using with ROS:

1. At the time of development in Matlab, an interface the provided the necessary framework of communication with ROS was non-existent. The interfaces that did exist relied on third party packages and mostly worked on Windows (PC), where as development at this stage was being carried out in Linux. Later on, Matlab officially started supporting Matlab interface with ROS, however, the capabilities of this interface were limited. Simple nodes that only allowed a few preset ROS data types was possible.

2. The biggest drawback of using Matlab with ROS was the slow computations in Matlab. Matlab implementations tend to be much slower than their corresponding C/C++ counterparts. Matlab is good for quick and easy development, but not suitable for real time applications, specially dVRK extensions.

3. Most of the packages in ROS developed by the ROS community are incompatible with Matlab. Packages such as TF[37] and Joint State Publishers[21], which maintain a server based knowledge of all the running robots and their corresponding transforms were not available for Matlab.

4. While Matlab is great for graph based visualizations, it lacks the tools and interfaces for quick and easy visualization of Robots and Frames. This is the area where ROS shines. Visualizing robots and transforms is simpler in ROS and is one of the greatest strengths of ROS.

6.5 Using MoveIt and OMPL

As time passed, I developed the cisst-ROS bridge that was capable of high speed communication with dVRK components thus is became possible to start development in ROS using ROS packages. MoveIt[33] is a standalone package that integrates Open Motion Planning Library (OMPL)[35]. The package has been used extensively in the current development to aid in the usage of planning algorithms and its compatibility with visualization software such as RViz. One other advantage of using MoveIt over using OMPL directly, is the abstraction of OMPL internals. OMPL does not have an integrated visualization package, although it can be tied up with any visualization package but that requires extra work. Also, OMPL cannot directly read urdf files to create the configuration space of the robot. For OMPL each joint has to be specified separately along with its joint limits. Thus working only in OMPL can be a tedious task for development.

MoveIt relies on the urdf file to start with. Using urdf files in proper formatting, move groups are created. Movegroups are kinematic groups in a kinematic chain that are used to distinguish which part of the robot needs to be analyzed and thereby solved for motion planning problems. Movegroups are easily understood with an example of a humanoid robot. For a given task, we would want to plan only for the right arm, for some tasks just the left arm, for some just the base movement,

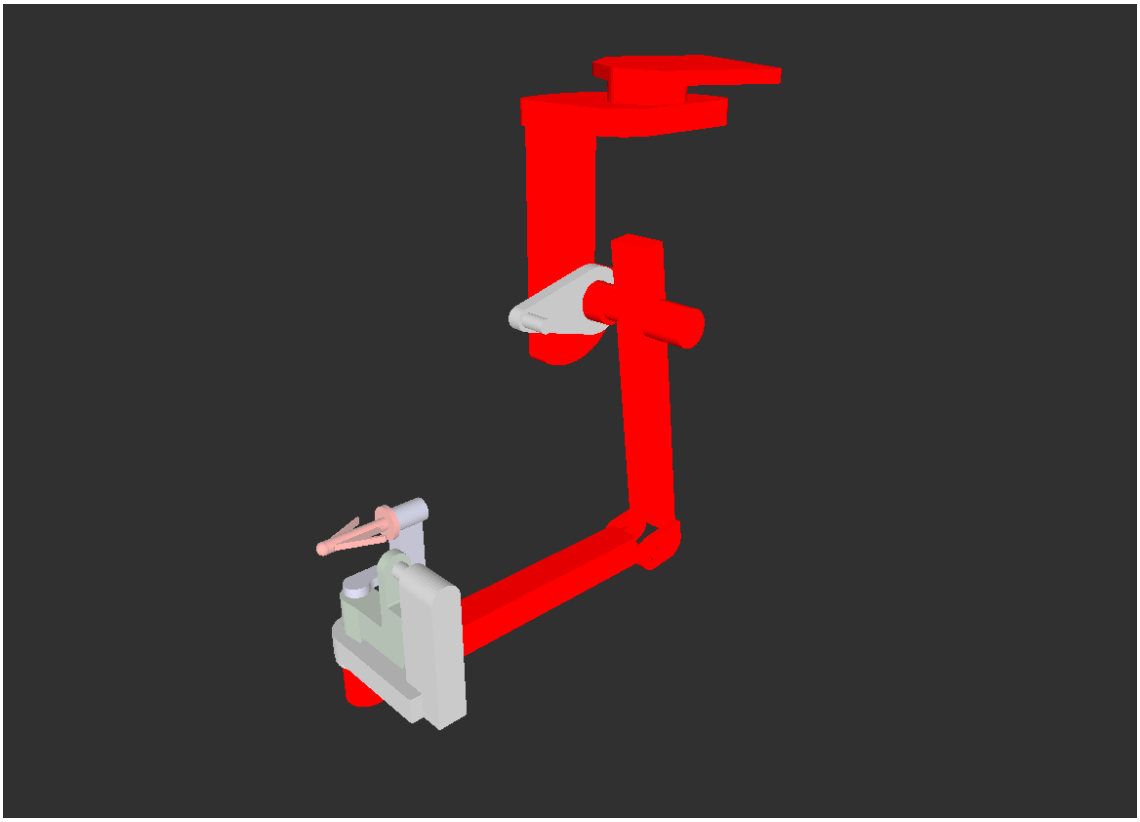


Figure 6.14: MTM's half_arm_group visualized in Red

and occasionally the entire robot for some tasks. For all these problems, a separate movegroup is created for the right arm, the left arm, the base and the full robot. Building upon the idea of modularity, even the right arm itself can have multiple move groups, one might be for the end effector, one for the joints before the end effector and one for the entire arm.

6.5.1 Movegroups for dVRK

The MTM and PSM represent different classes of robot manipulators having different kinematic structure and hence require separate classification of thier inverse kinematics formulation. One advantage of using movegroups is that this difference can be avoided as will be shown shortly. The movegroups are used for defining kine-

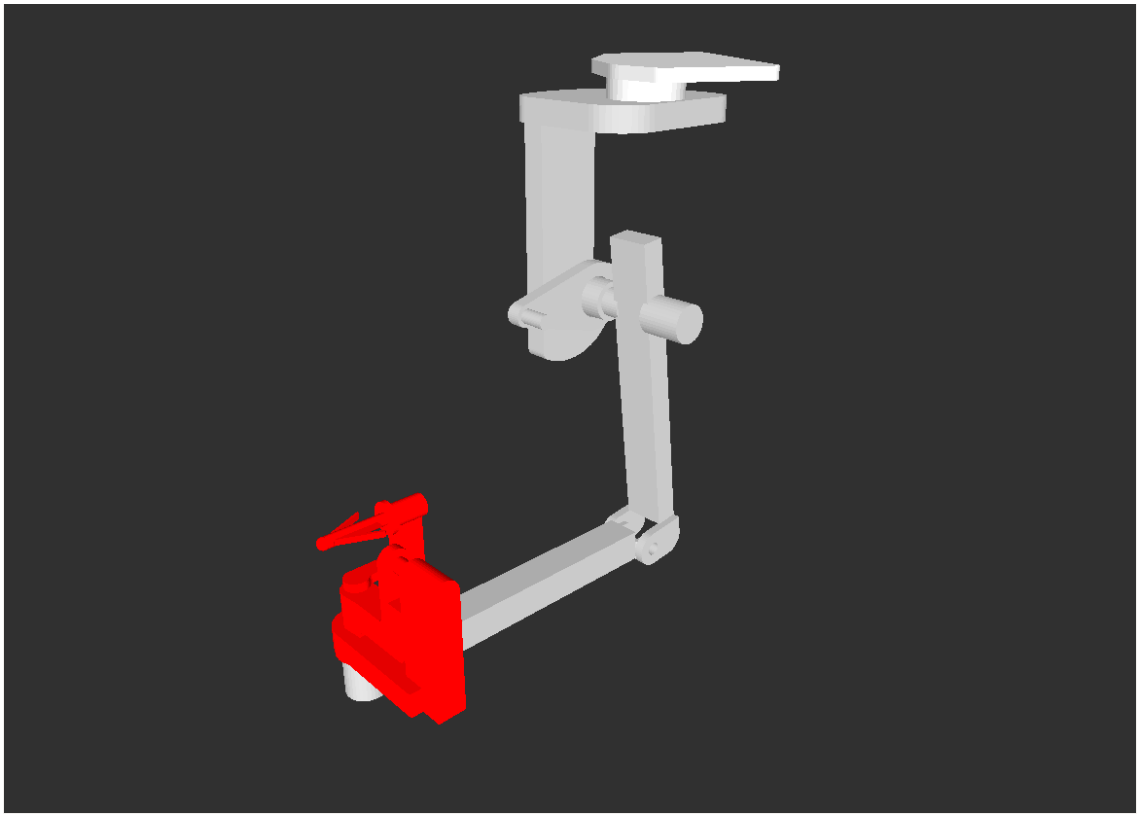


Figure 6.15: MTM's end_effector_group visualized in Red

matic chains within MoveIt. These kinematic chains are used for solving inverse kinematics for a given configuration. If we are planning in the operational space rather than the configuration space, the path planning problem will correspondingly search in the operational space for a solution. Operational space has 6 parameters for a non-plannar arm, as is the case for the dVRK manipulators. These 6 parameters are the x,y and z position in space of the end effector or the last link in the chain and its roll, pitch and yaw, usually in the world frame. Hence operational space parameters are not 1x1 mapped with the DOF. MoveIt allows for planning in both the configuration and operational space, directly.

For the MTM 3 movegroups have been created. The first movegroup is shown

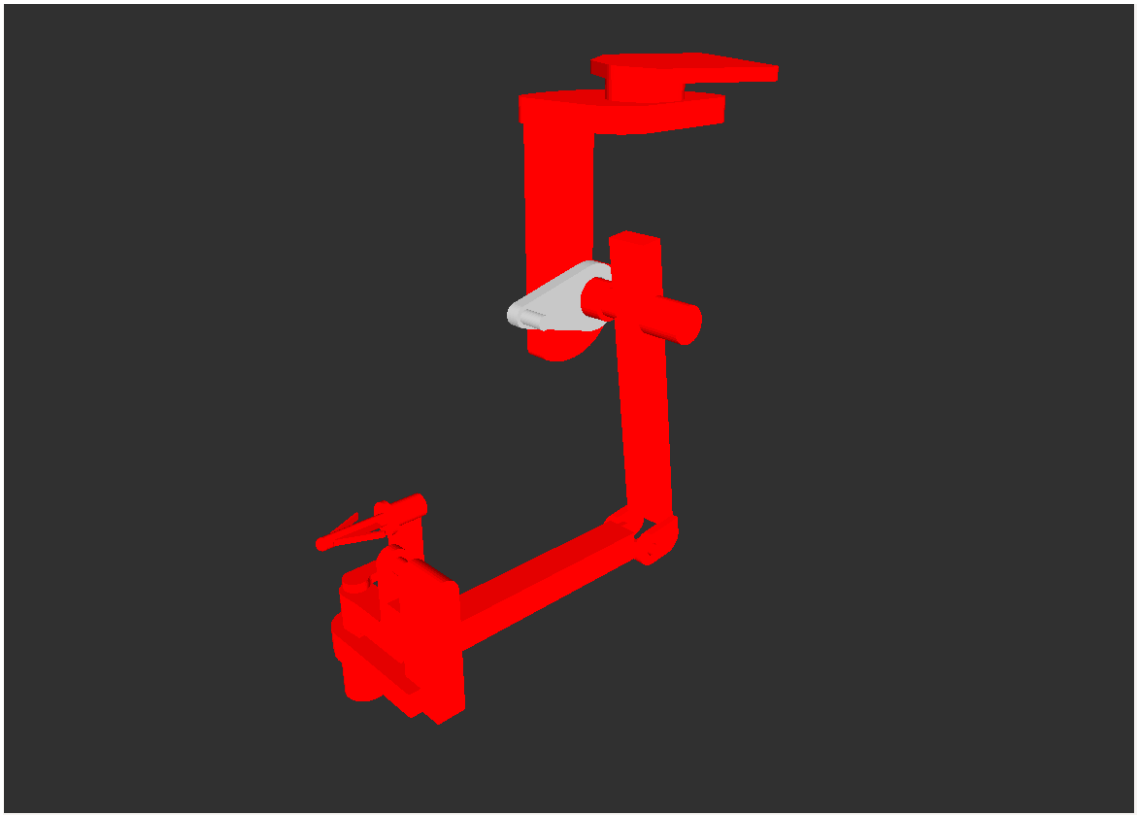


Figure 6.16: MTM's full_arm_group visualized in Red

in figure 6.14. This movegroup consists of joints starting from the base joint of the MTM to the joint in the wrist. This movegroup is called half_arm_group. All the movegroups are shown in red color in their corresponding figures.

The second movegroup is shown in figure 6.15. This movegroup consists of the end effector. The end effector of the MTM is a spherical mechanism with 4 joints in it. This movegroup is called end_effector_group. The third movegroup is shown in the figure 6.16. This complements that other two move groups shown in figure 6.14 and6.15. The movegroup comprises of the entire MTM manipulator and is called the full_arm_group.

Similar to the movegroups of MTM, the PSM is also classified into 3 movegroups.

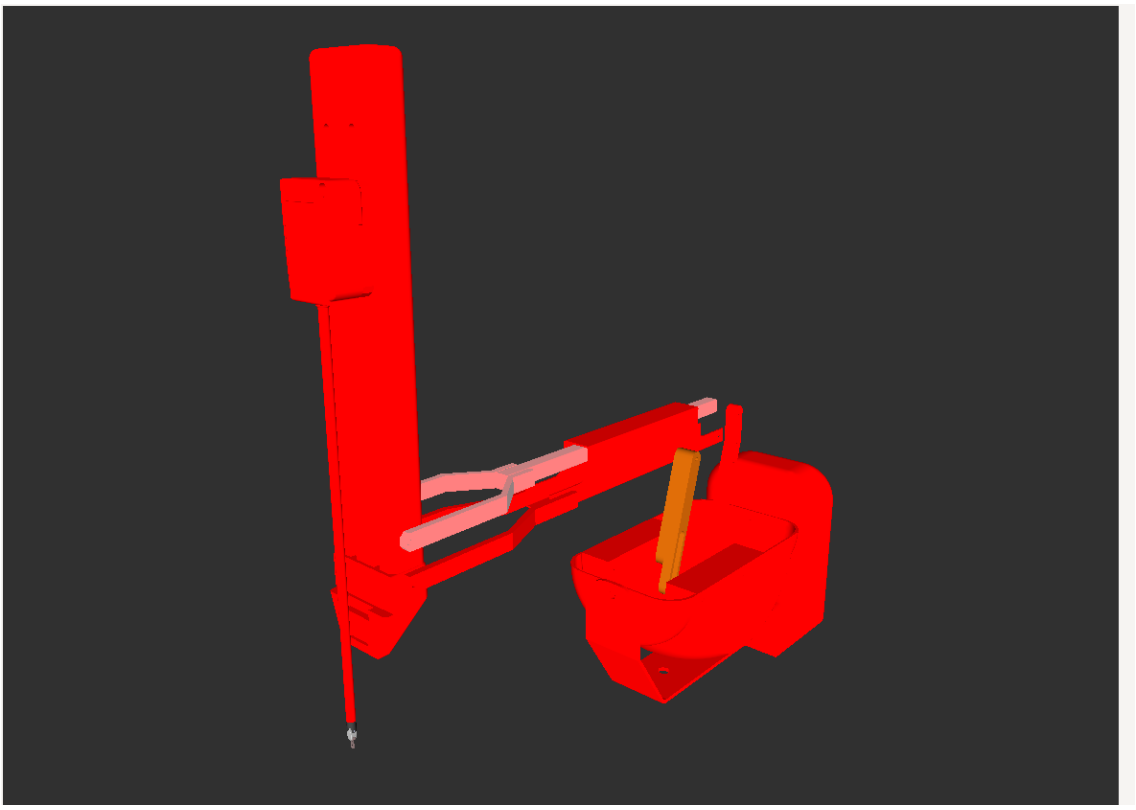


Figure 6.17: PSM's halfl_arm_group visualized in Red

These movegroups are half_arm_group, end_effector_group and full_arm_group. They are shown in figures 6.17, 6.18 and 6.19 respectively. It shoud be noted that this is not the only way the movegroups can be classified. The movegroups can be classified in any way, however, the current classification is generic and makes intuitive sense for the purpose of this research.

6.5.2 Movegroup integration in RViz

One key advantage of creating movegroups is that they can be easily analyzed in ROS. The movegroups are readily supported in RViz and a default kinematics solver is integrated for solving foward and inverse kinematics for each movegroup. The IK solver is based on an iterative solver that takes in the joint positions of the arm and

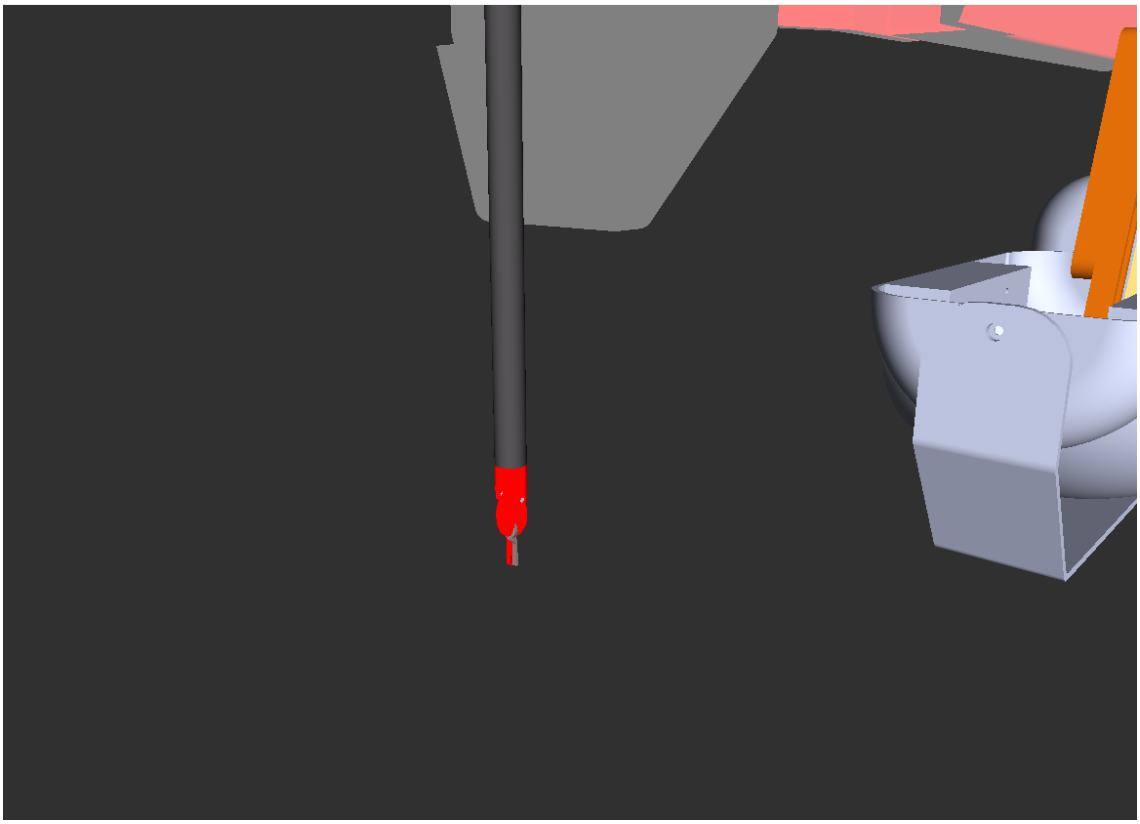


Figure 6.18: PSM's endl_effector_group visualized in Red

solves for the desired position. It is interesting to note that an identical IK solver in implemented in the cisstRobot libraries. Hence using this built in solver for MoveIt and subsequently for Rviz while using the cisstRobot solver for TeleOperation or other tasks produces similar results.

It is worth mentioning that the builtin in IK solver used in movegroup is taken from Kinematic and Dynamic Library (KDL), from the OROCOS project. KDL itself is a standalone package and is nicely integrated into MoveIt. Just like OMPL, KDL's direct interface is abstracted away from the user and the user directly deals with the simpler interace of MoveIt. KDL deals with chains and trees to deal with kinematic problems, which the MoveIt maintains internally with the creation of

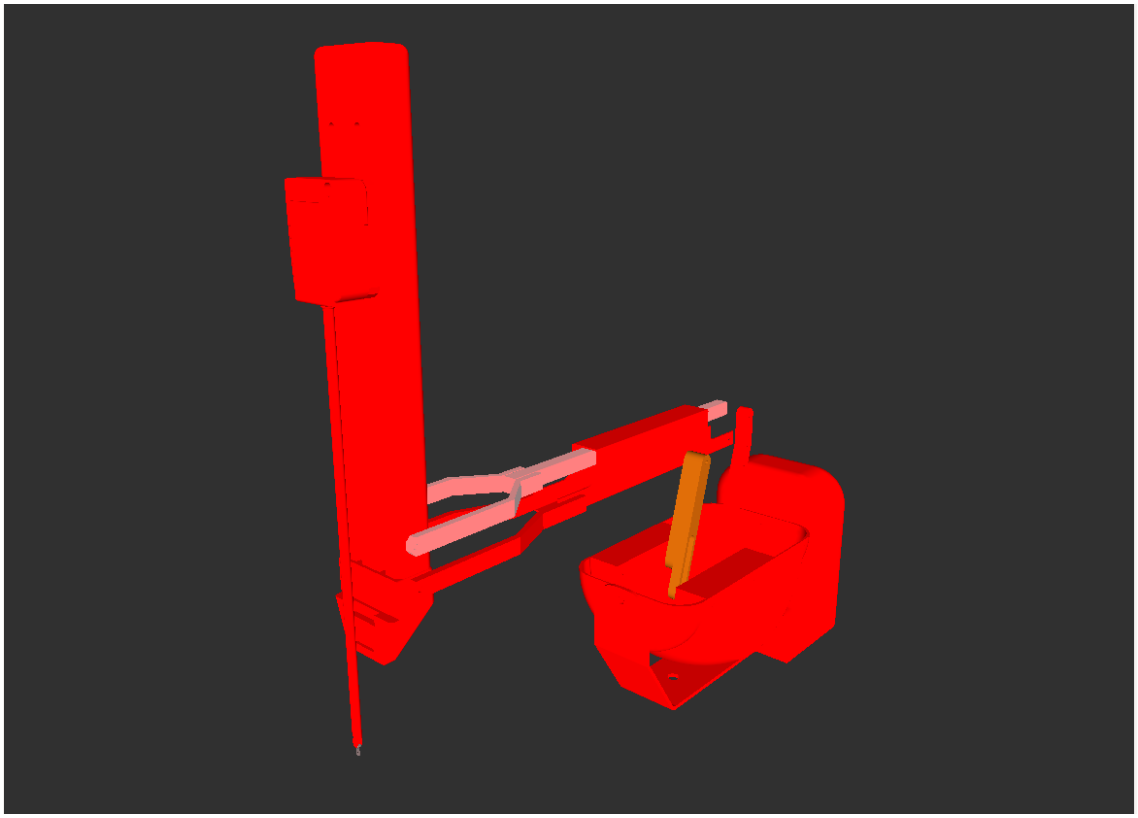
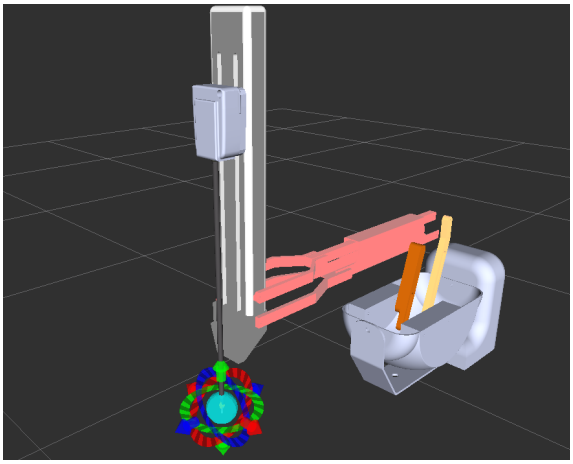


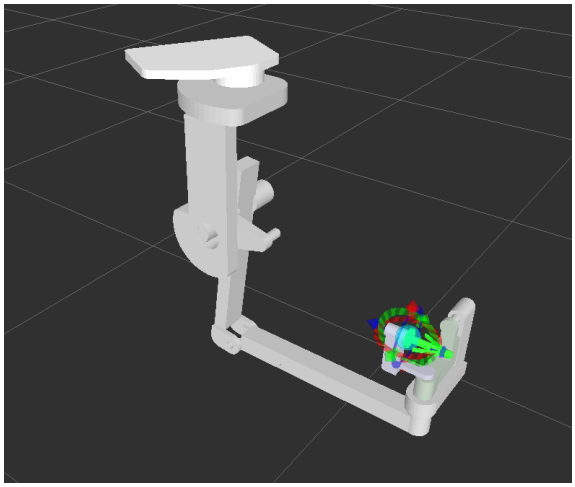
Figure 6.19: PSM's full_arm_group visualized in Red

movegroups from URDF files.

When ever a movegroup is visualized in RViz, an associated, interactive marker is visualized on the last link in that movegroup. This interactive marker provides the means of actuating the end effector and thereby creating new robot configurations. This provides a quick and handy tool for simulation purposes. Figure 6.20a shows the interactive marker associated with the half_arm_group for PSM. The movegroup for MTMs full_arm_group is shown in figure 6.20b.



(a) Interactive Marker associate with PSM's half_arm_group



(b) Interactive Marker associate with MTM's full_arm_group

Figure 6.20

6.5.3 Visualization of planning problems in RViz

For validation purposes, simple motion planning problems have been tested on the PSM in this section. Spheres have been generated together to form a bigger obstacle as shown in Figure 6.21 (a) and (b). The PSM has been manually given start and goal configurations for this problem shown is orange and green respectively. The start and goal configurations are chosen such that the PSM has to go near the remote center to reach the goal thus making it harder for the planner to compute.

The remote center of the PSM (Figure 3.5) restricts the motion in the horizontal plane about itself. Essentially, the configuration space collapses to a point at the remote center. Any obstacle near to this point, makes search for points in the valid configuration space intensive and difficult. By the definition of Random Sampling in continuous space, the probability of finding a valid point at the exact remote center is zero.

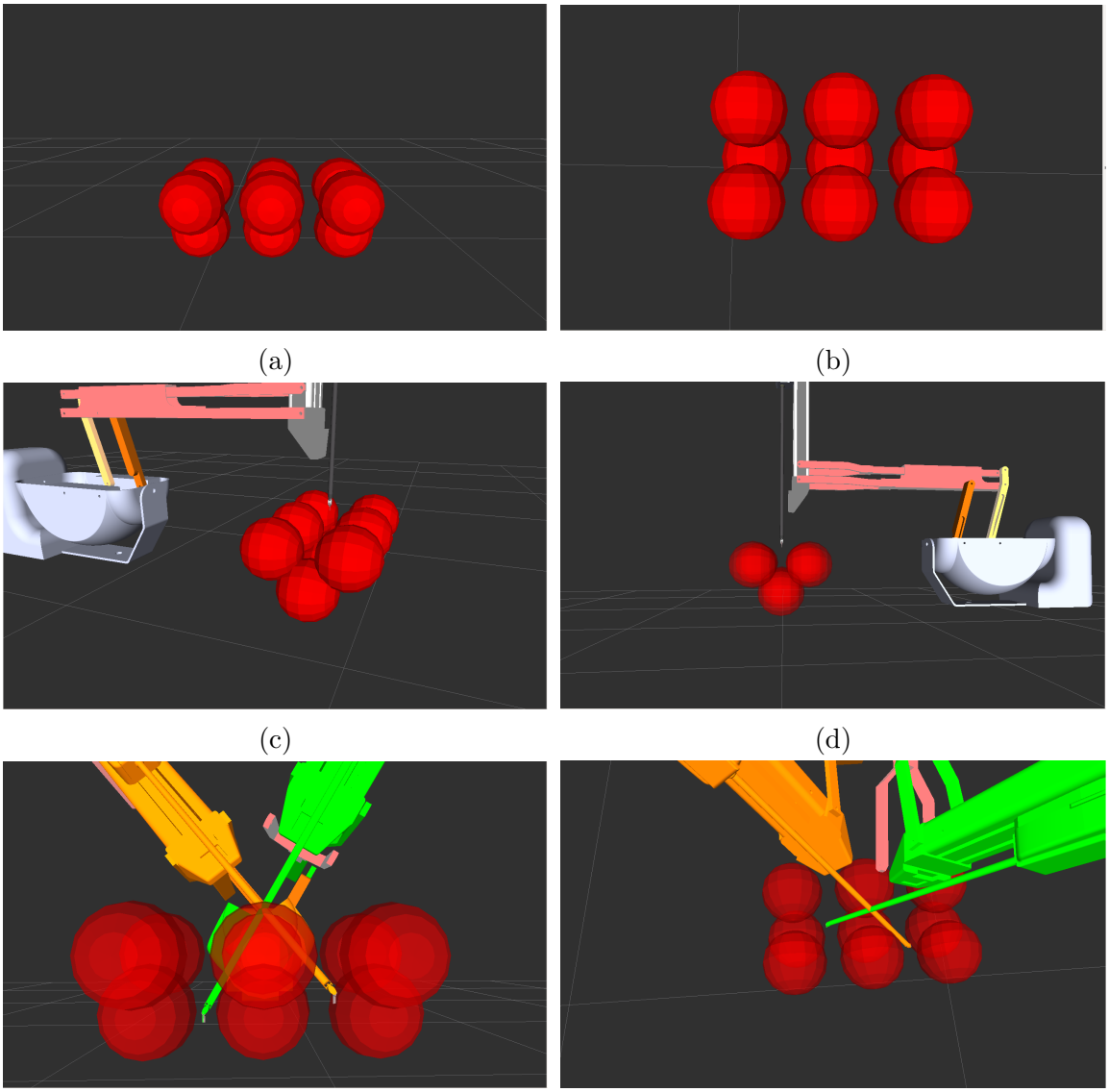
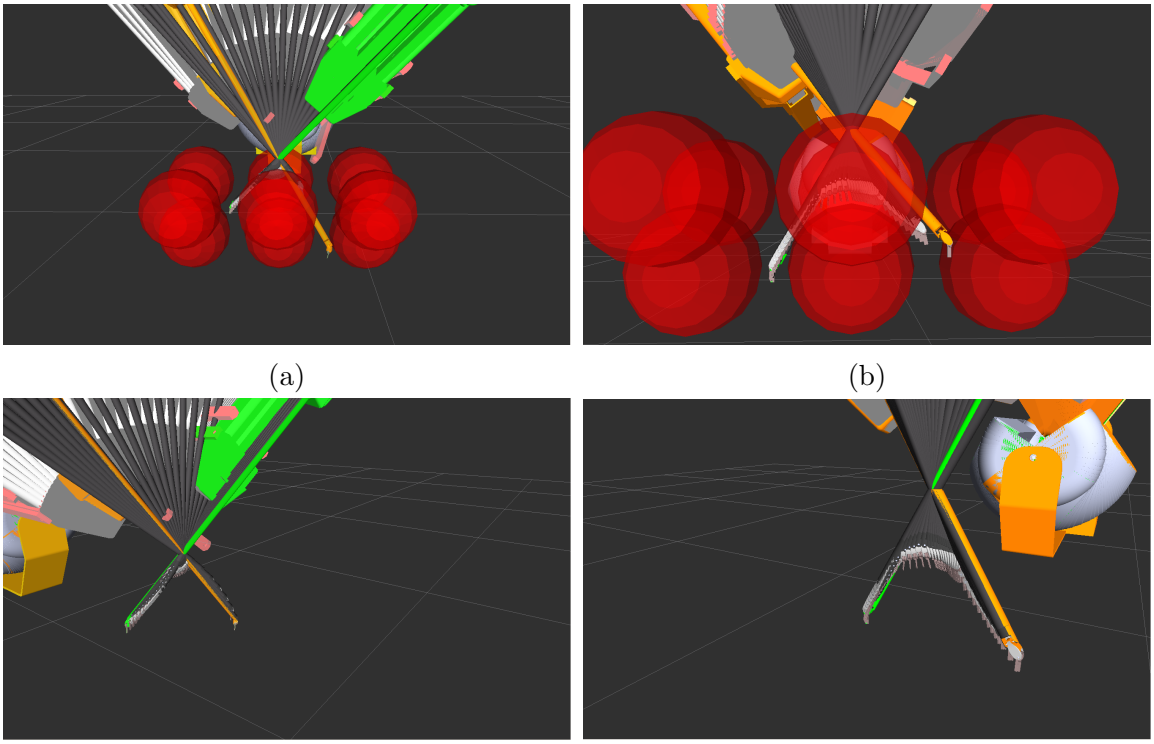


Figure 6.21: Figures (a) (b) showing the obstacle mesh from different angle, (c), (d), (e) and (f) showing different perspectives of the PSM and its start and goal state

The results of the path planning are shown in Figure 6.22. The aid in visualization, the results are shown as a trial of PSM configurations as it traverses through the solution points. The trail shows that the PSM has to lift the end effector, forcing it to come near the remote center to reach the goal position.



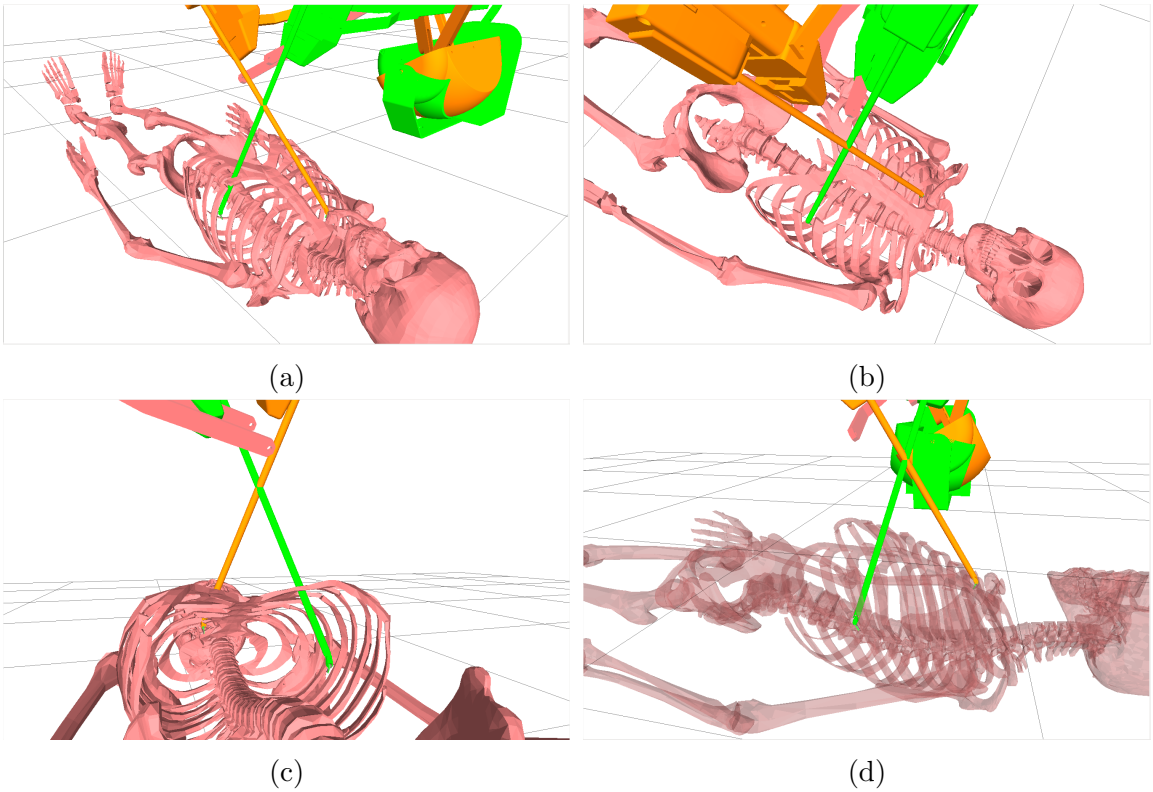
(a) (b)

(c) (d)

Figure 6.22: Figure (a) and (b) displaying the planned path with the obstacles in view. Figure (c) and (d) showing the same path from different perspectives with hidden obstacles

6.5.4 Visualization of Advanced Planning Problems

Having achieved satisfactory results for simple motion planning problems, more advanced planning problems were tested. In the previous experiment simpler meshes were used. When dealing with surgery the planning environment is much more complicated. The estimation of the environment with simple meshes is not feasible. To demonstrate the use of motion planning to more advanced planning environments, the experimental setup involves a significantly more complicated mesh that contains more than 10,000 faces. Such a mesh stretches the state validation functions and requires more time and memory than the previous experiment.



(a) (b)

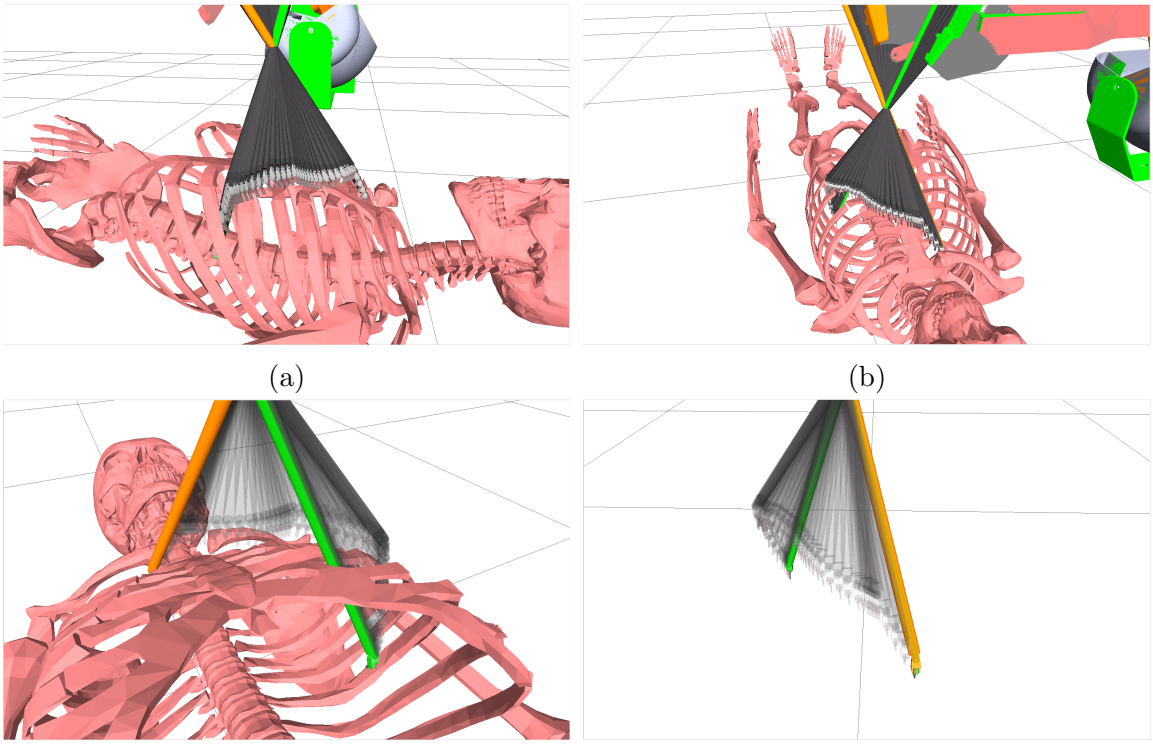
(c) (d)

Figure 6.23: Planning a path inside a 3D volumetric rendering of a human skeletal structure with figures (a), (b), (c) and (d) showing different views of the same scene

As can be seen in figure 6.23 a 3D volumetric rendering of a human skeletal structure is placed in front of the PSM. The mesh is much more complex and has areas which require significantly denser state exploration, such as the area surrounding the ribs. The goal of this planning problem is to take the PSMs end effector, located deep inside a pair of ribs to another deep insertion in a different pair of ribs just to demonstrate the validity of the path planning and obstacle avoidance algorithms. Figure 6.24 shows the result of the path planning. Figure 6.24 (a) (b) (c) show a trial of PSM configurations as it traverses through the volumetric skeletal model. Figure 6.24 (d) shows just the path, with the skeleton hidden.

This approx does not implicate any direct clinical application since the PSM goes from one deep insertion to another one by moving out of the body. This experiment

just demonstrates the concept of proof that based on the implementation of the motion planning framework, complex motion planning problems can be visualized and solved, atleast in simulation.



(a) (b)

(c) (d)

Figure 6.24: Planning a path in a dummy skeleton. (a), (b) and (c) showing different views of the planning scene and figure (d) shows just the path produced

6.6 Assisting the Surgeon with Assisted Path Planning

6.6.1 Use Case

As discussed in the introduction of this chapter, the proposed use case for the motion planning framework with the dVRK is to assist the surgeon plan paths

during surgical procedures. For pre and intra-operative procedures, the surgeon can set paths by teleoperating the virtual PSMs in the simulation environment with the representation of the actual surgical environment.

6.6.2 Requirements

The realize this use-case the following requirements have been outlined:

- The simulation environment must be easy to use.
- Existing component of the dVRK must be use and no additional hardware components must added.
- The simulation must provide a visual feedback of the motion of the PSMs controlled via MTMs by the surgeon.
- The TeleOp control of PSMs in simulations must not alter the actual PSMs motions. Thus a path should be planned first in simulation only.
- The simulated PSMs must provide the same TeleOperation experience as the actual PSMs.
- Visual feedback for the collision of the simulated PSMs with the simulated surgical area must be provided.
- The start and goal points provided by the surgeon must be clearly visible so that surgeon is aware of the points chosen.
- The resetting of the start and goal points, as many times as the surgeon wants, must be possible, without restarting the simulation.

6.6.3 The Software Experimental Setup

For achieving the goals listed in the requirements several ROS executables have been developed. Figure 6.25 gives a brief overview of the executables that need to be running to achieve the listed requirement.

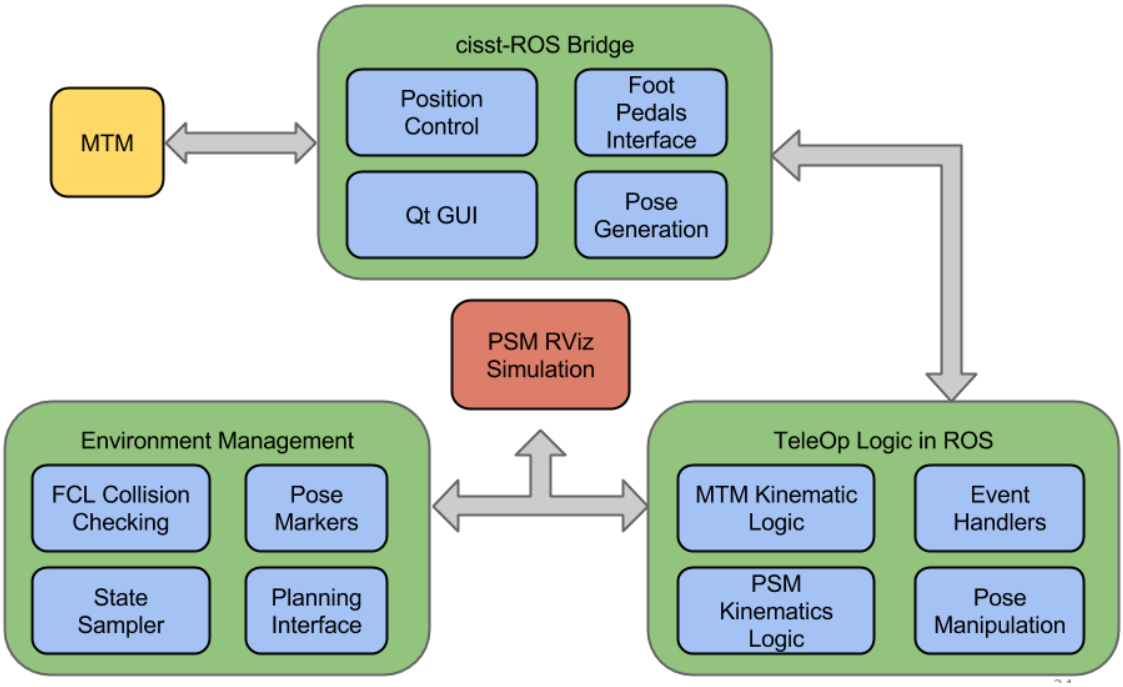
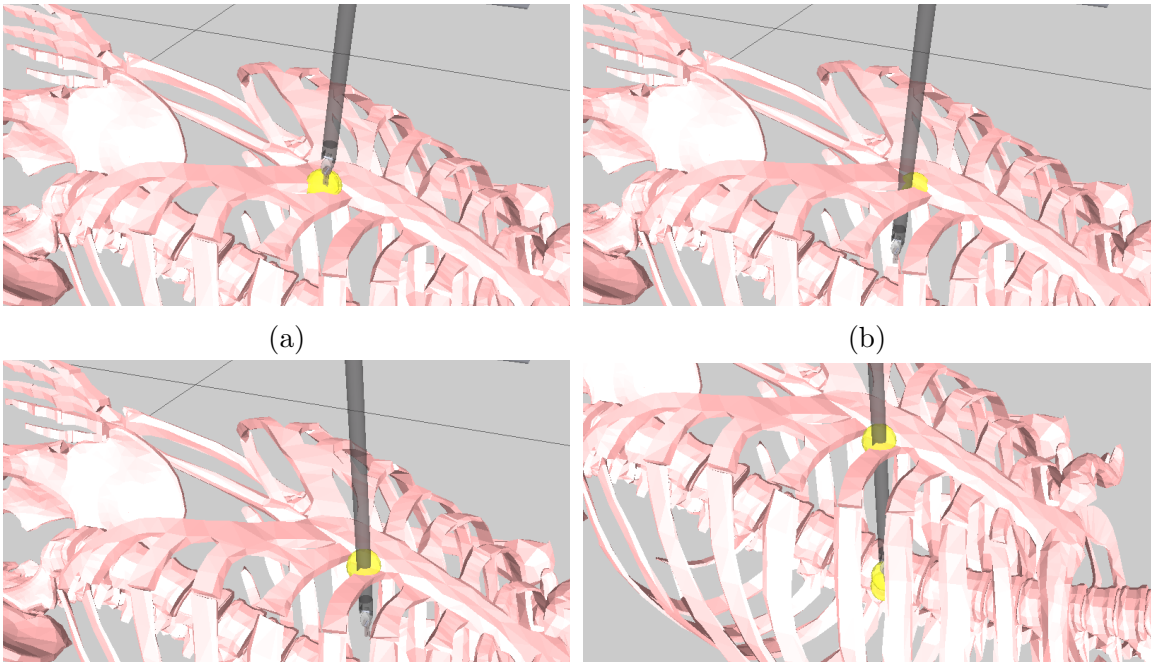


Figure 6.25: The experimental setup in terms of software for assisted path planning. The cisstROS bridge handles the publishers and subscribers for getting and setting dVRK parameters. The TeleOp namespace has several nodes that allow for Teleop of simulated PSM from actual MTM. The Environment Management namespace handles the environment representation, collision detection, visualization, start and goal state visualization and the motion planning related tasks

6.6.4 Obstalce Detection and Visualization

To satisfy the detection of collision between the PSMs and the obstacle environment and visualization several package have been used. These include the MoveIt wrapper for the FCL collision checking library and ROS visualization markers array. Once the environment has been represented, it is published as a planning_scene. RViz is

set to receive this topic and visualize the environment. The 3D volumetric skeletal model shown in figures 6.21 and 6.22 for example use this planning_scene as well.



(a) (b) (c) (d)

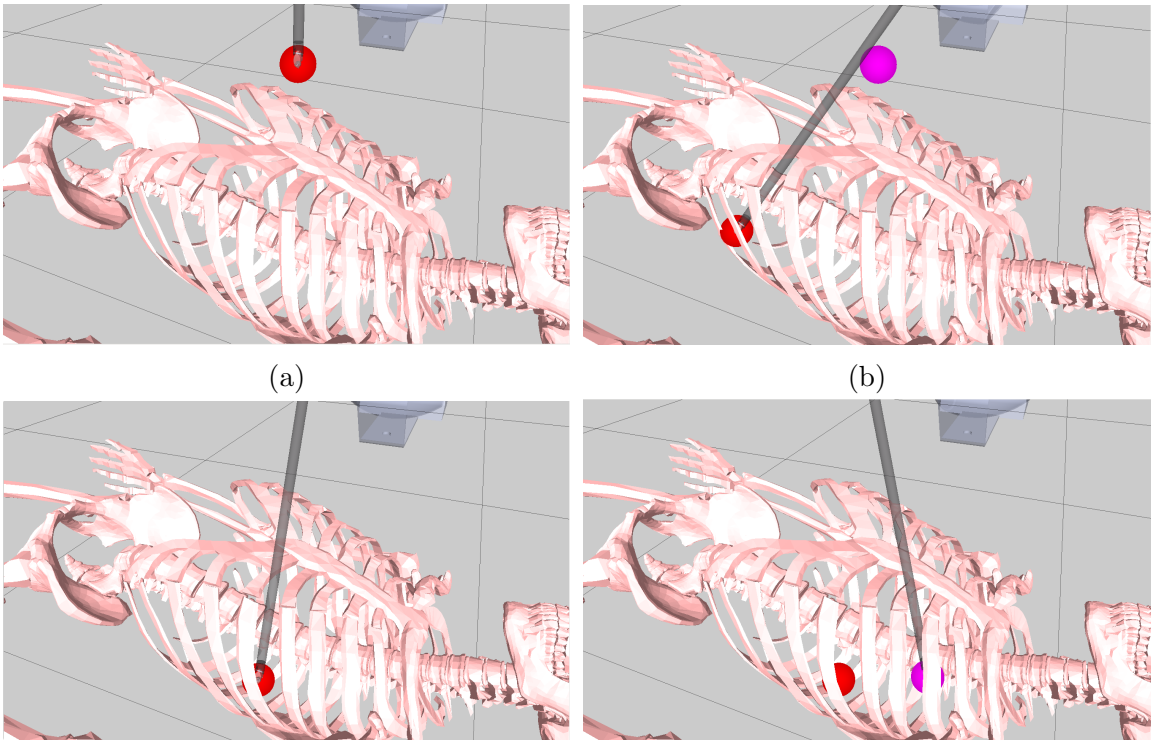
Figure 6.26: Collision contact of the PSM with the 3D volumetric skeletal model displayed as yellow spherical markers. Visual aid for collision awareness while Teleoperating the PSMs

Once the planning_scene is available over ROS server, the collision checking works in parallel. I have utilized the MoveIt wrapper for this purpose which allows for obtaining contact points between the PSM and the environment. If a collision occurs, the contact points are captured and are published to RViz for visualization and this blocks the selection of a start or goal point (Fig 6.26).

6.6.5 Visualization of the Start and Goal Points

While the collision contact markers are essential to navigate around the simulated environment with knowledge of the collision areas, additional visualization is provided to display the start and goal states that are chosen. If the chosen points need

to be changed, the user can just enter two more points and the previous pair of start and goal points will be erased.



(a) (b)

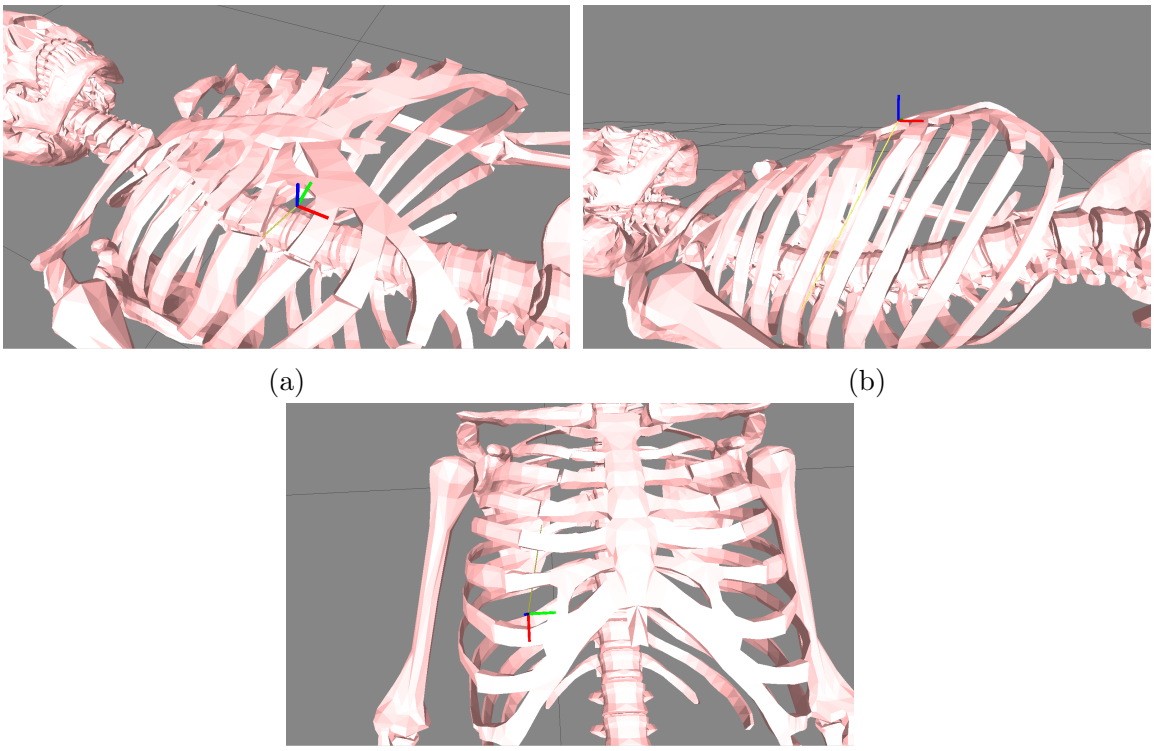
(c) (d)

Figure 6.27: Visualization of start and goal states chosen in figure (a) and (b). (c) and (d) show the capability of changing the start and goal states just by specifying additional pair of points which results in deletion of the previously registered start and goal points

6.6.6 Using the Foot Pedal Tray to get Start and Goal Points

As mentioned in the requirements of the assisted motion planning, no new hardware component must be added for path planning purposes. The dVRK Foot Pedal tray has unused pedals. These include the *camera*, *cam+* and *cam−* pedals. For capturing start and goal poses, the user presses the *cam−* minus pedal, which results in the visualization of a point (start) and then presses the pedal again to register the

goal point. If different points are needed, the user just keeps on moving the PSMs to the required points and pressing the *cam−* pedal until a pair of start and goal points are selected. To instantiate the motion planner, the user presses the *camera* pedal, after the planner terminates with a successful path,, the computed path is visualized in simulation.



(a) (b)

(c)

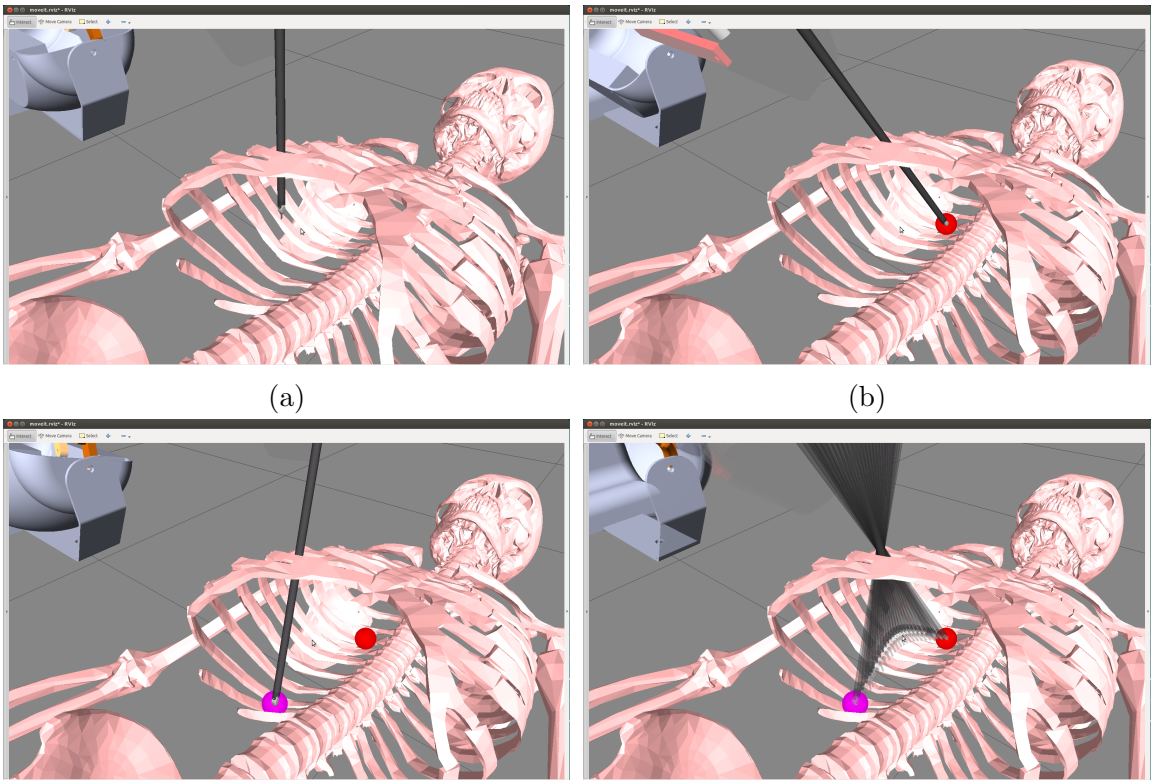
Figure 6.28: The sole frame is the three images shows the location of the remote center of the PSM. This location is chosen at the center of the ribs and at the same height as the ribs

6.6.7 Demonstration

To demonstrate a realistic scenario, the remote center of the PSM is placed between the ribs such that the movements around the remote center do not contact the 3D volumetric skeletal model. Figure 6.28 shows the remote center placement from

three different angles. In actual laparoscopic surgeries, a remote center is chosen in such a manner, depending upon which area is being operated. For this demonstration, a random pair of ribs has been chosen, which do not demonstrate any surgical preference other than the usefulness of the assistive path planning.

In figure 6.29 two points have been placed inside the 3D volumetric skeletal model using a single insertion point. The path computed for these pair of start and goal points is shown in figure 6.29(d).



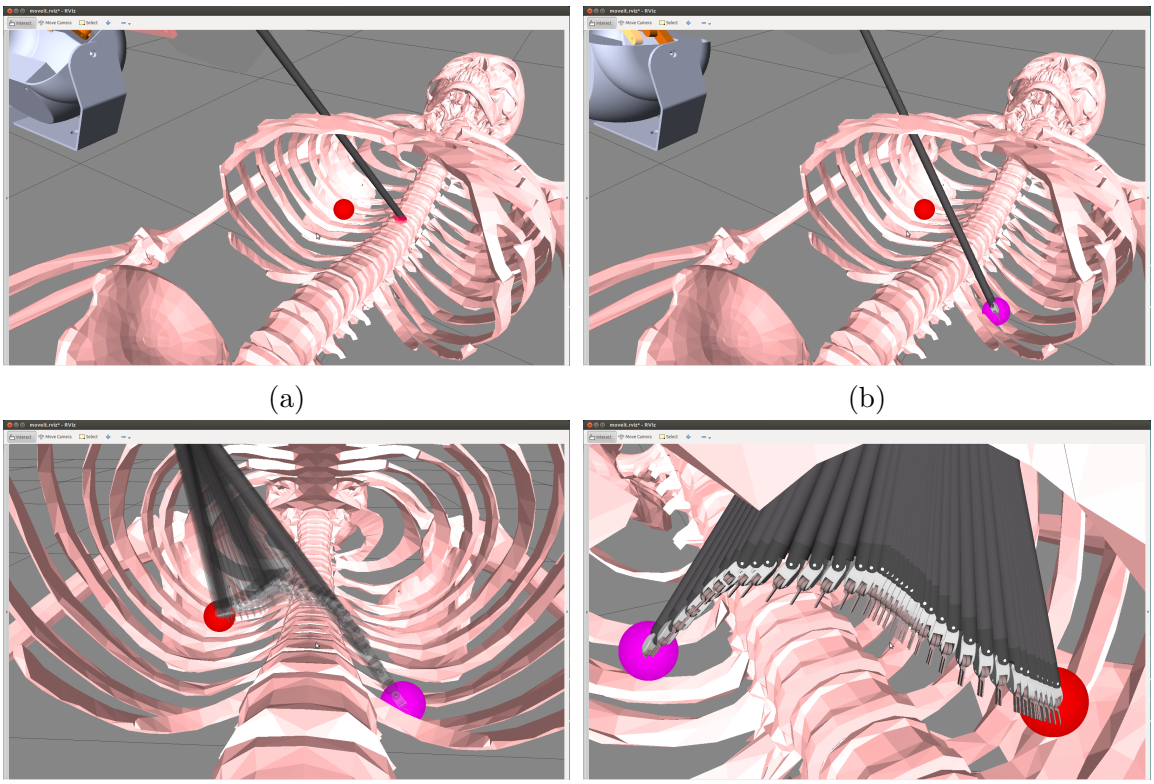
(a) (b)

(c) (d)

Figure 6.29: Placing a pair of start and goal points using a single entry point (a) in figures (b) and (c). The path produced is shown in figure (d)

For a more complex planning problem, a pair of points have been placed the opposite sides of the spine using the same insertion point for the PSM. The PSM

has to go over the spine and much deeper to reach the goal point for this problem. The results are shown in figure 6.30.



(a) (b)

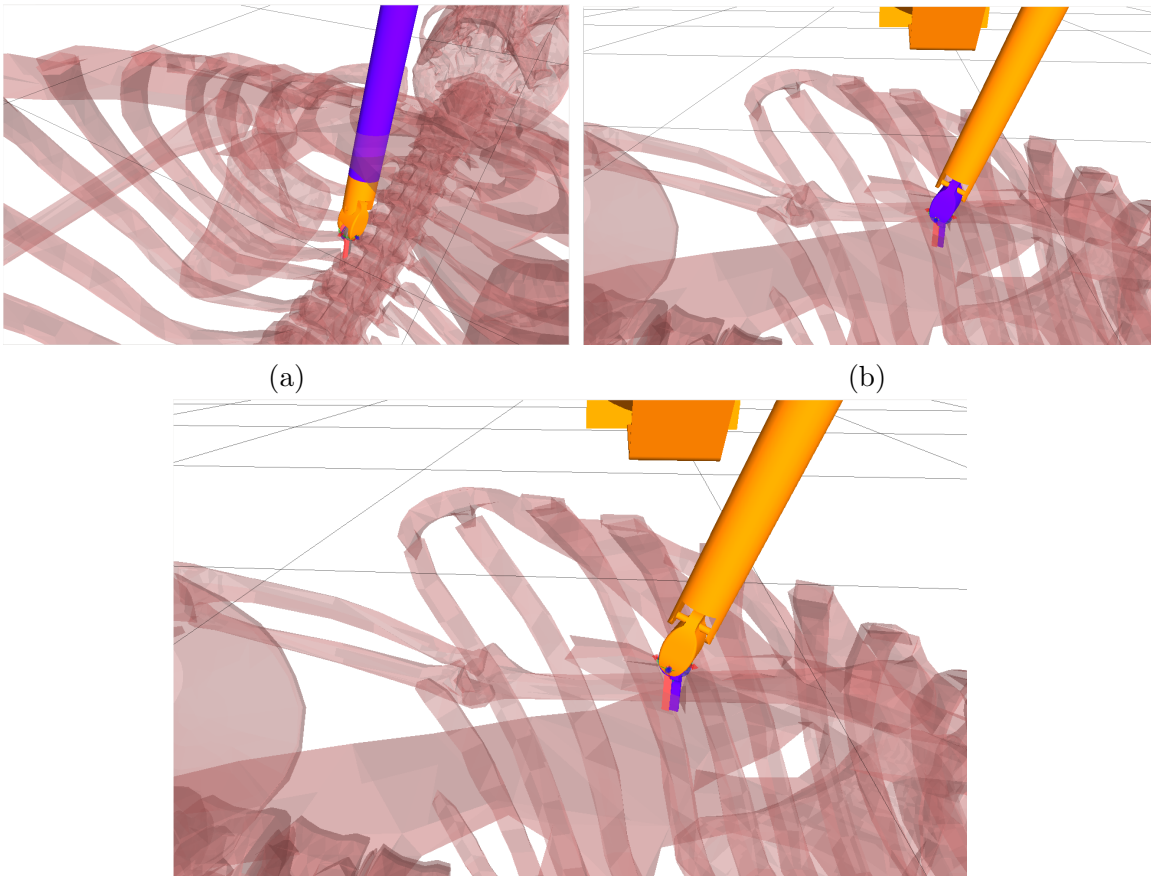
(c) (d)

Figure 6.30: Placing a pair of start and goal points at opposite sides of the spine using a single entry point. (c) and (d) show the path produced from different angles and different translucency

Discussion

The implementation of MoveIt with dVRK works well in the simulation environment with paths found on an average of a few milli seconds. The planners used and their comparison is discussed in section 6.6.8. These results hold for complex meshes, such as the 3D volumetric skeletal model. These results are encouraging since state validation is a computationally intensive process and depends upon the complexity of the mesh. FCL (Flexible Collision Checking Library) is the default collision

checking library implemented for MoveIt [24]. FCL can handle obstacles of three types; Obstacles represented as meshes, primitive shapes or as octo maps. This allows flexibility at the user end while defining obstacles.



(a) (b)

(c)

Figure 6.31: PSM links in collision with the mesh obstacle shown in Purple color

The links that collide with the mesh can be detected using a variety of ways, while MoveIt triggers and event instantaneously. This is shown in figure 6.31 where any link that collides with the mesh is programmed to show up in purple color. Figure 6.31 (a) (b) and (c) show the collision detection in a thin rib section of the meshes; as the PSM end effector is withdrawn out of the mesh, different link are in collision as depicted by the figure.

Before MoveIt could perform collision detection, the URDF model of the PSM was set up to be compatible for FCL. In MoveIt, this is achieved using a matrix representation called Allowed Collision Matrix (ACM)[23]. This setup insures not only the collision detection between the external objects and the PSMs links, but also with the links themselves. However, some links are in collision at all times, by design. To take care of this, ACM has to be modified by hand. This tweaking is partly scientific and partly iterative by observation. MoveIt setup assistant is a tool that helps in getting started with the ACM[23], however, in many cases, one needs to alter this ACM to get better collision detection of intricate and price movements in and and around obstacles. For simulation purposes, the ACM provides satisfactory results, however, for more precise experiments, the ACM has to been modified accordingly.

To get the collision contact points as shown in the Assistive Planning section, a collision request has to be created using the $planning_scene$ interface provided by MoveIt. The collision contacts can then be used in many different ways, and for these experiments, they are visualized in RViz.

6.6.8 Comparing Planners Using MoveIt Benchmarking Tools

Difference between the Matlab and MoveIt Benchmarks

The Matlab Implementation of the RRT and its variants was compared in section 6.4.5 using a self developed bench marking module. It was stressed due to Matlab's own overhead, the relative difference between the planners in the performed comparisons tells a lot about the workings of the planner however there is not much value to the absolute data for a single planner. The bench marking tools available in MoveIt are certainly different in this regard and provide a different insight into the

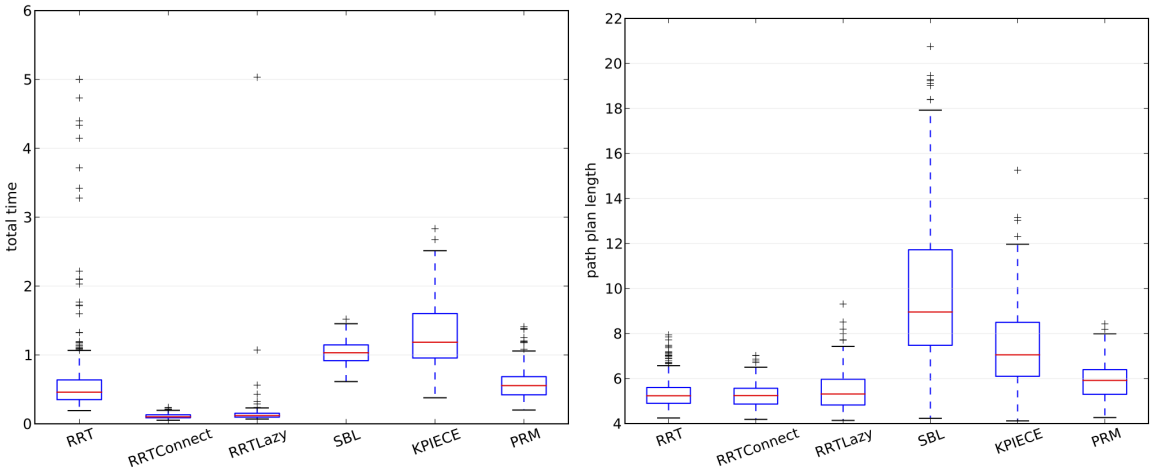
working of each planner. However, in the Matlab Implementation, the comparisons of each planner were presented by varying the step size, such a comparison is not possible in MoveIt Bench marking tools.

Additionally, in MoveIt, the comparison has been performed with not only the RRT Planners and variants, but with all the other modern planners including KPIECE[34], SBL[30] and the first ever class of random time planners, the PRM[14] planner. While discussing the details of KPIECE and SBL are beyond the scope of this research, a short introduction of SBL and KPIECE is given below:

- **The SBL Planner**: The SBL planner is a modified implementation of the Probabilistic Roadmap Planner (PRM). SBL starts by expanding trees from both the start and goal states as the PRM planner does. It delays state validation till a path is found.In this aspect, it resembles the RRT Lazy planner discussed in section 6.4.2. It has been shown to produce good results in real world scenarios [30].

- **KPIECE Planner** The Kinodynamic Piecewise Interior Exterior Cell Exploration algorithm, as the name suggests is designed for both kinematic and dynamic systems [34]. It segments the C_{space} into cells and searches within to find a path. In case a path is not found, it sequentially segments the space into a finer resolution and searches again. KPIECE has also proven to produce better results than RRT variants in many problems [34].

Figure 6.32 shows the comparison between the three RRT variants, RRT-extend, RRT-connect and the Lazy RRT along with other planners. The planners are each run for 300 iterations, with the skeleton based advanced planning problem. The result for each planner is plotted as a box plot. For the comparison between com-



(a) (b)

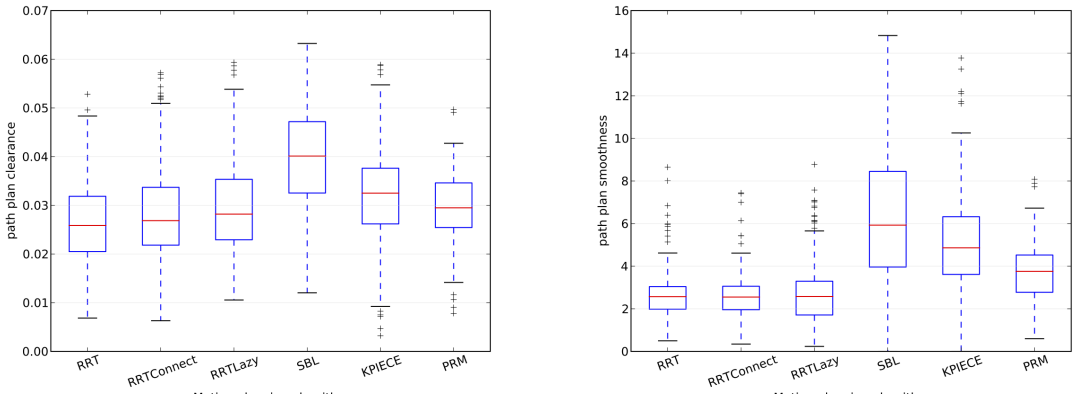
Figure 6.32: Comparison between RRT Extend, RRT Connect, Lazy RRT, SBL, KPIECE and PRM in terms of solution time in seconds (a) and path length (b)

putational times, it can be seen that RRT-connect and RRT-Lazy have taken less than half the amount of time that RRT-extend took to find a solution path. The outcome of the average path lengths between the three RRT variants does not show noticeable or meaningful difference for this problem set.

The comparison has been extended to additional planners available in MoveIt. In figure 6.32, additional planners have been compared. It is interesting to note that, for this problem setting, RRT connect and RRT lazy remain the most efficient in terms of computational time.

The MoveIt bench marking tools collect data about many more parameters than just the path lengths and computational times. These include many comparisons some of which are:

- A clearance factor for path from the obstacles
- The overall smoothness of a path represented.
- The number of states explored



(a) (b)

Figure 6.33: Comparison between RRT Extend, RRT Connect, Lazy RRT, SBL, KPIECE and PRM in terms of path clearance from obstacles and overall path smoothness

- Memory used by the planner for planning

Figure 6.33 shows two additional comparisons in terms of path smoothness and path clearance between all the planners. The results are more favorable towards the RRT variants.

6.6.9 Using Optimal Path Planners

The implementation of dVRK components in MoveIt allows the use of many other planners other than the ones listed above. These planners are Optimal path planners, just like RRT* that was discussed in section 6.4.2. These planners include:

- The RRT*
- PRM*
- Lazy PRM* and many others

The comparison of these planners is not that trivial since these planner run for all the allotted time and try to improve their path continually. Furthermore,

being random time planners, for a given bound of time, they produce very different outcomes, that cannot be simply used to tell which planner is better overall.

6.7 Discussion

This chapter presented the implementation of RRT and its variants in Matlab. This implementation was used to conduct comparisons between different planners. This later led to the development of a motion planning framework for the dVRK using MoveIt and ROS. The movegroups for PSMs and MTMs have been discussed with justification for choosing specific grouping of the joints. In the end various motion planning problems have been experimented in simulations.

A use case for assisitve path planning has been presented, the term assistive path planning, referring the additional aid to the surgeon while planning. The experimental setup has been discussed with the work cycle of getting the pair of start and goal points for motion planning using the existing dVRK hardware and the ease of use of the planning framework.

Chapter 7

Dynamic Simulator for the dVRK and a Matlab Interface

As part of extending the dVRK research platform, additional tools/platforms have been integrated and interfaced with CISST/SAW and ROS. This chapter briefly discusses these tool and platforms.

7.1 Dynamical Simulation of dVRK Manipulators in Gazebo

Gazebo is an research tool for robotics. It's capability of emulating dynamical behaviors of robots and manipulators makes it very valuable for academic research. Gazebo now comes as a standalone package and can be used with ROS for enhancing the feature set of both the platforms. Gazebo is compatible with 4 dynamic solvers as of now. These include the popular ODE Engine and the Bullet Physics Engine, with ODE engine being the default one. The dynamics of each robot is represented internally as a set of differential equations. The constants(masses, inertia's, link

lengths, etc) of the differential equations are provided by the SDF or URDF files that are discussed in the next section.

7.1.1 Setting up the MTM in Gazebo

Gazebo requires a Simulator Description Format (SDF) for robot description. SDF format is similar to the URDF format in some ways and in most ways an evolved version of the URDF. Gazebo can be used to convert a compatible URDF to SDF for visualizing the robot thus saving the effort to develop an SDF format for the dVRK components separately as the URDF files had already been created.

To make the URDF files Gazebo compatible, the description of dynamic properties of each link were required. These dynamic properties include masses, inertia's and also joint properties like damping and friction. Additional properties can also be modeled. Since there was no specification of the such properties, many of the values are mere approximations based on the CAD models. Using approximate values is discussed in 7.1.4.

MTM right has been modeled in Gazebo as shown in figure 7.1. Gazebo simulates the model dynamically and thus any interactions with the manipulator would follow the dynamic behavior of the robot based on the given dynamic parameters. This is in contrast to RViz, which only simulates the robot kinematically. In the URDF files "torque interface" and "sensor" plugins have been added in addition to setting the link and joint properties. These plugins do not affect any other functionality so these have been added as default to all the URDF files.

7.1.2 Controller Plugins for Gazebo

To interact with a robot or a manipulator in Gazebo, plugins have to be developed. For the model shown in figure 7.1, I have developed custom controller plugins that

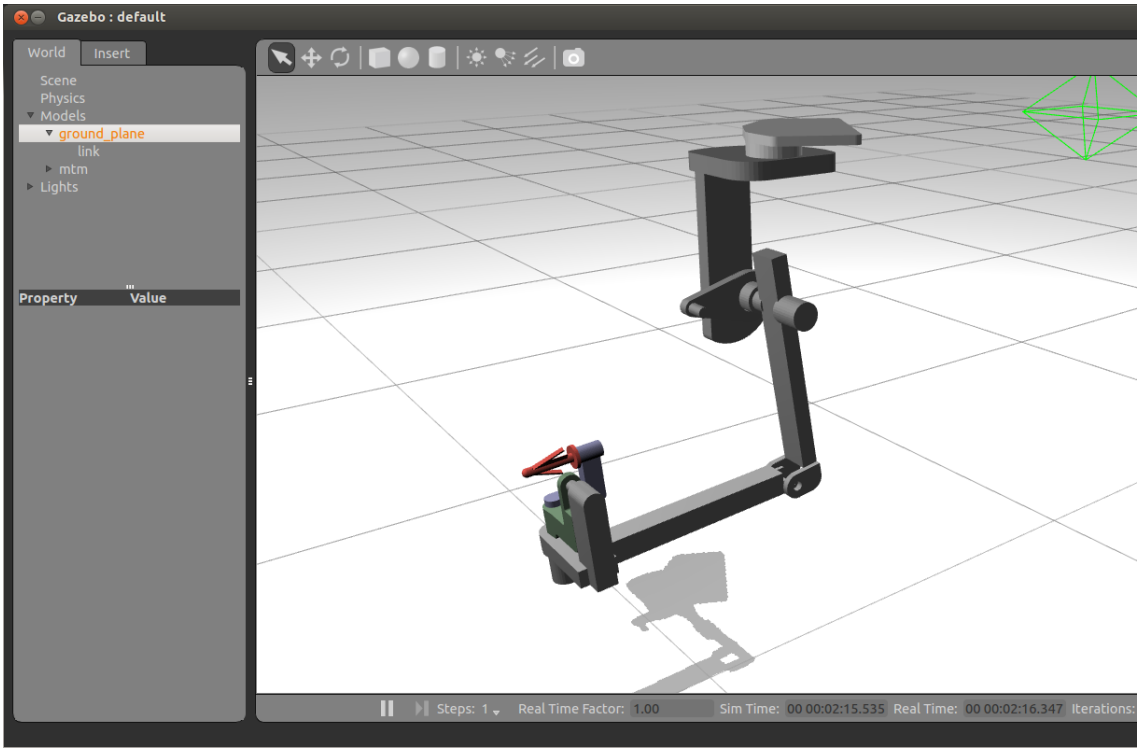


Figure 7.1: MTM modelled in Gazebo

allow for the interaction of the manipulator from ROS. These controller plugins treat each joint separately and model a PID control loop for each. Hence position, velocity, acceleration or torque control can be achieved based on the need. I have used position and torque control for initial experimentation. I have added a GUI with sliders for each joint to test the setting of joint torques. The GUI has been developed using the *rqt_gui* package that aids in the development of publishing and subscribing to topics over ROS-server.

7.1.3 Controller Performance tools for Gazebo

I have extended *rqt_gui* to visualize the controller performances in Gazebo. This package provides many other features which I plan to use in the future. For now, the graph plotting plugin is used to track the input command to the joints in the

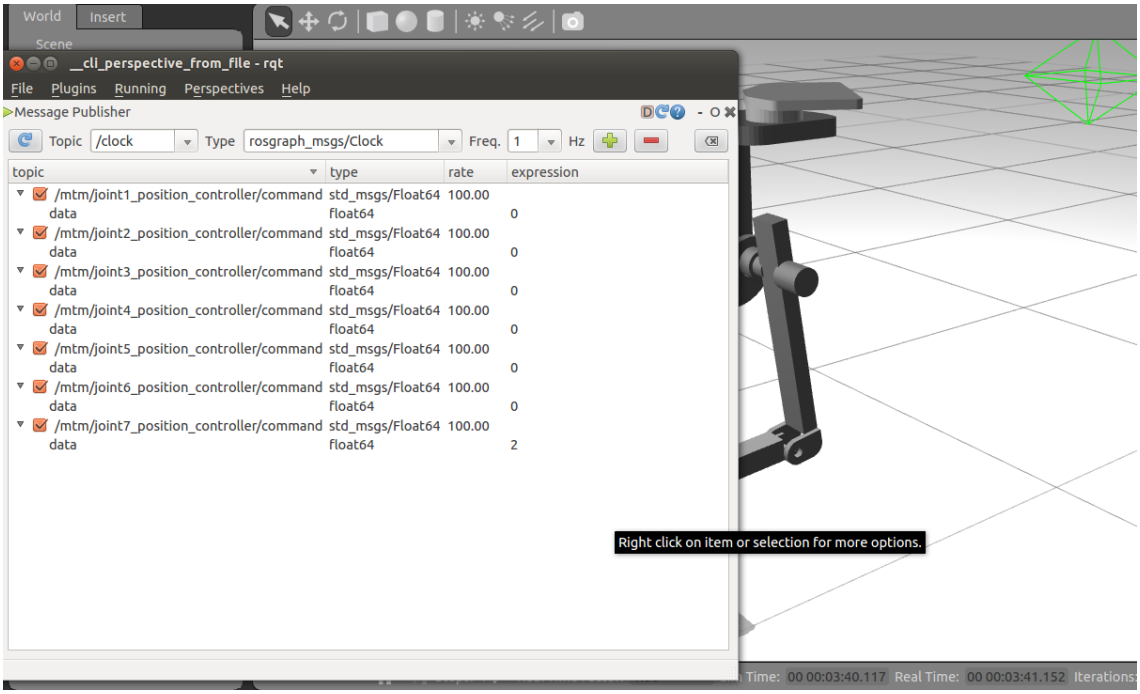


Figure 7.2: Gazebo Controller Plugin for MTM

Gazebo model and the sensor output. The sensor output represents the current position of the arm. The results for controller performance are shown in figure 7.3. Two joints are displayed and are compared to their input commands that are sinosoids of different frequencies and amplitude.

7.1.4 Dynamics of the MTM

The General Dynamics Equation

Any robotics manipulator can be broken down in three indigenous components that comprise its dynamic model. Taking a look at the Dynamical Equation of a generic robot manipulator in 7.1.

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) \tag{7.1}$$

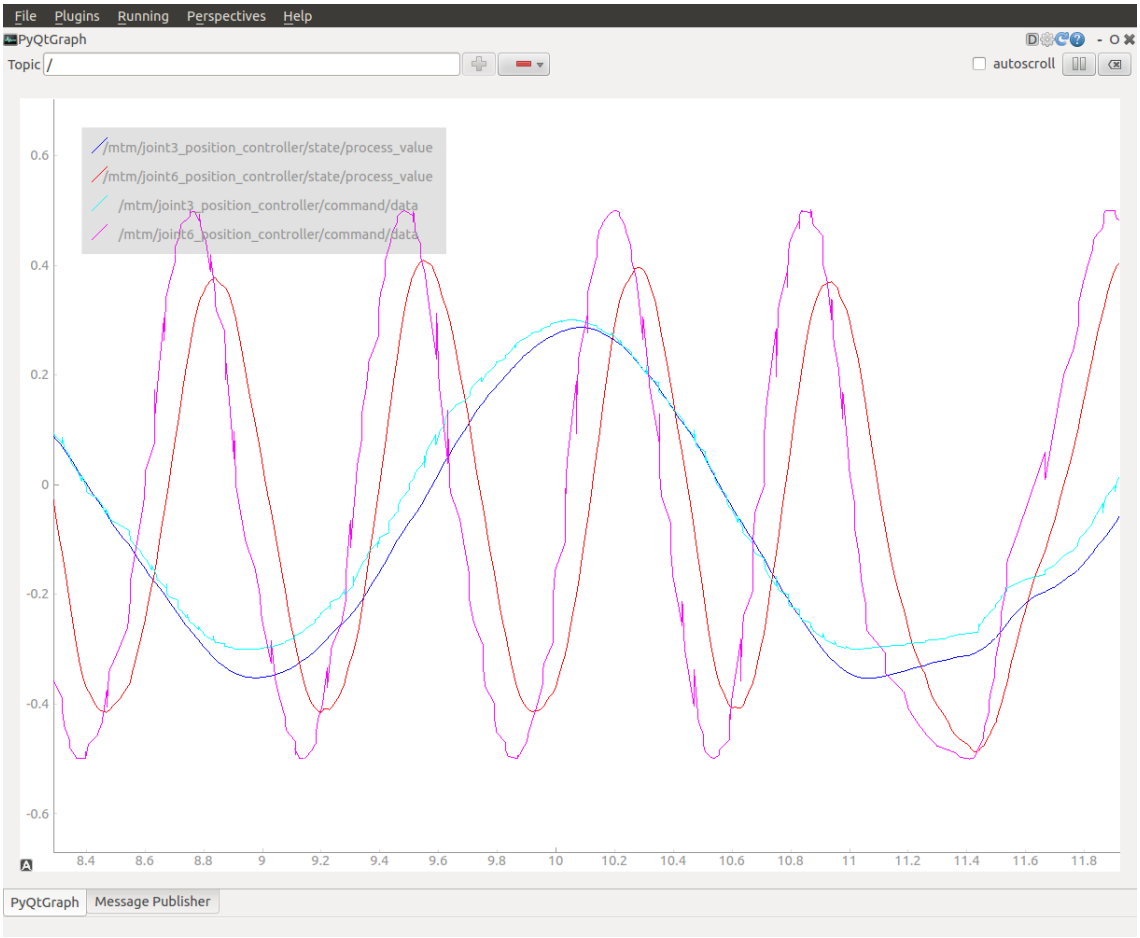


Figure 7.3: Each pair of the two different sinosoids represent the input torque to the joint and the resulting position of joint. In this graph, the 3rd(green and blue sinosoid) and 6th joint(red and purple) of the MTM are used

The three components of the Dynamics are the Inertia matrix M, the Coriolis matrix C, which consists of the Coriolis components of joint velocities and the g matrix which depends solely on the position of the manipulator and the gravity acting upon it. This equation only accounts for the internal forces and torques on the manipulator. The external forces on the manipulator induce torques on the joints that can be calculated using the Jacobian matrix.

Going into a bit more detail, whenever any link of the manipulator experiences

acceleration, the M matrix acts and contribute to forces and moments at different joints. The C matrix constitutes forces and moments on the link when ever their exist non-zero joint velocities. Finally, the g matrix corresponds to the action of gravity on the manipulator. Even if the manipulator is static, the g matrix produces forces and moments on the manipulators links and joints. Additionally, all the three components depend upon the current joint positions. With the Coriolis matrix depending upon the joint velocities as well. In absence of gravity, the g matrix tends to be zero and has no effect on the dynamics of the robot manipulator. This case is specially true in space robots, where the robots have to perform tasks in zero gravity situations.

The determination of the dynamic components is tricky and non-trivial. When the links are being designed in CAD, the calculations of masses and inertia's of links are approximate. Later on, the machining and manufacturing processes render the approximations even more error prone. This is especially true for intertial terms in the M matrix and the Coriolis $C(q, \dot{q})$ which are quite sensitive in the calculation of dynamic model equation 7.1. The g offers relatively greater tolerance due to error in actual parameters. The g matrix also lacks any inertia terms in its compositions, hence the calculation of the gravity component is relatively trivial compared to the Mass and Coriolis Matrix.

Dynamics for the MTM

For the components of the dVRK, the dynamic model has to be calculated from scratch due to lack of any prior information regarding the dynamics. Even the actual CAD models for the robot are not available thus making the task even more

difficult. Even the dynamic properties of the CAD models are inaccurate, since the model is recreated from external inspection of the dVRK manipulators. This makes the CAD model much more simpler and non-realistic as the actual manipulator includes hollow links, cams, pulleys and wires etc that are not accounted for.

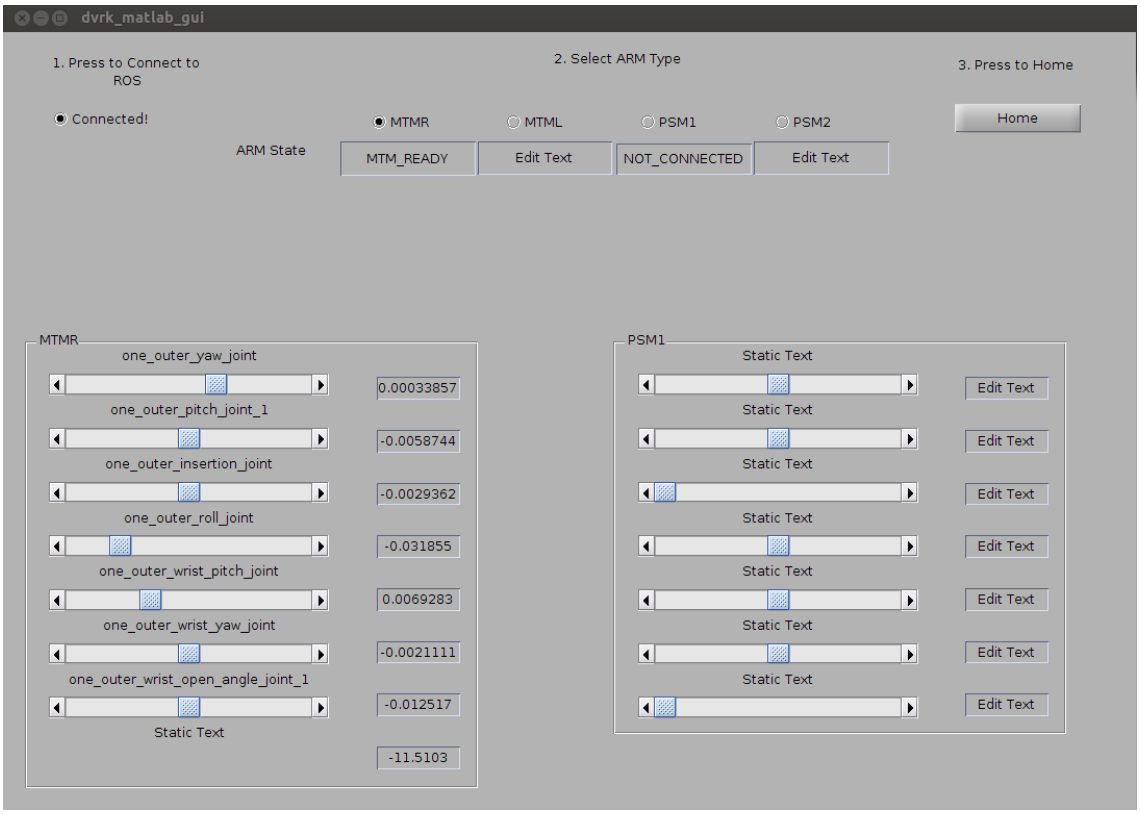


Figure 7.4: Matlab GUI for dVRK. The GUI shows that Matlab is connected to MTM and showing the current joint positions (in rad) next to the slider for controlling each joint

For current implementation, only the terms that constitute the gravitational components for the first three links of the MTM have been approximated using trial and error by JHU (Zihan). The values need improvement and this work is left for the future. Using the same values in URDF for Gazebo is bound to produce different results, as Gazebo would use these values to model the Inertia and Coriolis matrix as well. This would impart additional torques and make the simulated robot astray

from the actual robot as soon as any joint starts to move.

7.1.5 The Matlab ROS I/O

The inclusion of the newly released Matlab ROS IO package extends the research platform even further. The Matlab ROS IO allows for basic functionality for retrieving data from ROS using topics, messages or services. The functionality is only basic and is in no way a replacement for development in ROS. However, this tool allows for researchers to extract the data from ROS and evaluate it for different tasks.

I have developed an extensive Matlab ROS package that can be used for control of the dVRK manipulators through ROS. This package has been designed keeping in mind the need for people less familiar with ROS and advanced programming languages. A very informative GUI has been developed to provide debugging and connection validity with the dVRK manipulator. The GUI is shown in figure 7.4. This package has been programmed with intensive debugging features that allow it make multiple attempts to connect to dVRK and take care of exceptions. The Matlab ROS interface connects to the dVRK manipulator using two bridges. This is shown in figure 7.5.

Figure 7.5: the armBase class connect to ROS using Matlab-ROS IO bridge and ROS connect to CISST/SAW using cisst-ROS Bridge. Thus there are two bridges in between to connect matlab code to dVRK manipulators

Matlab-ROS Class

The GUI provides a quick and easy way for running the dVRK and evaluating simple control and communication. For more advanced algorithms, I have created a Matlab ROS class, called armBase, that has been released as a first version to be tested by Zihan at JHU. This class makes setting up ROS interface very easy. All a user needs is these lines of code in his/here program to set up all the subscribers, publishers and callback assignments.

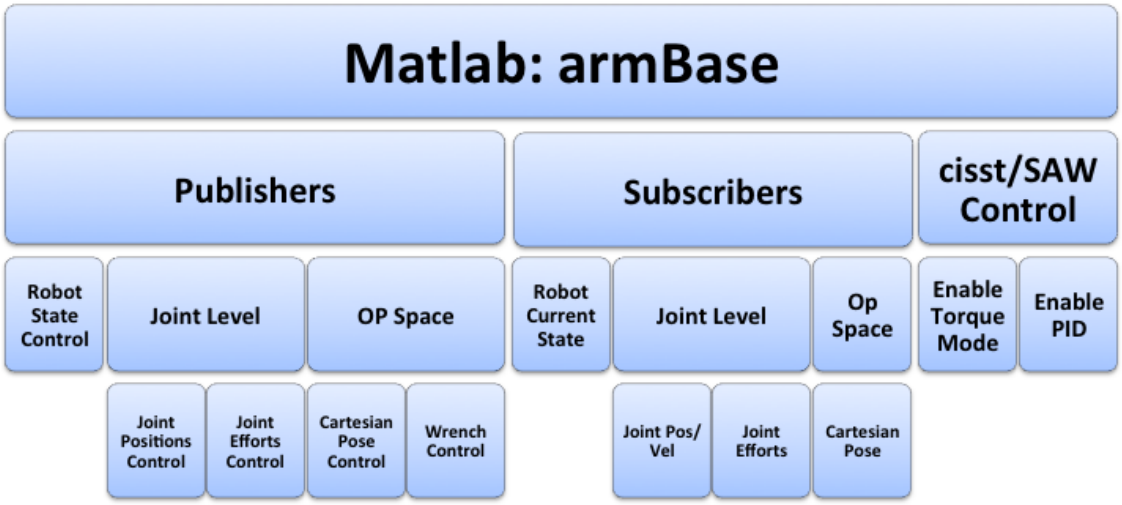


Figure 7.6: The armBase provides the following methods by default for the defined object type (MTM, PSM or ECM)

```
{
node = rosmatlab.node('/my_node')
mtm = armBase(node,'MTM')
mtm.setupArm
/*
 algorithms
```

```
*/
mtm.clean_up
}
```

The Matlab class "armBase" thus creates an object depending upon the string passed to it that could be either PSMs or MTMs. The object has all the methods for setting and retrieving anything from joint positions and torques to manipulator state in very simple to use fashion. Fig 7.6 shows an overview of the functionalities of the armBase class.

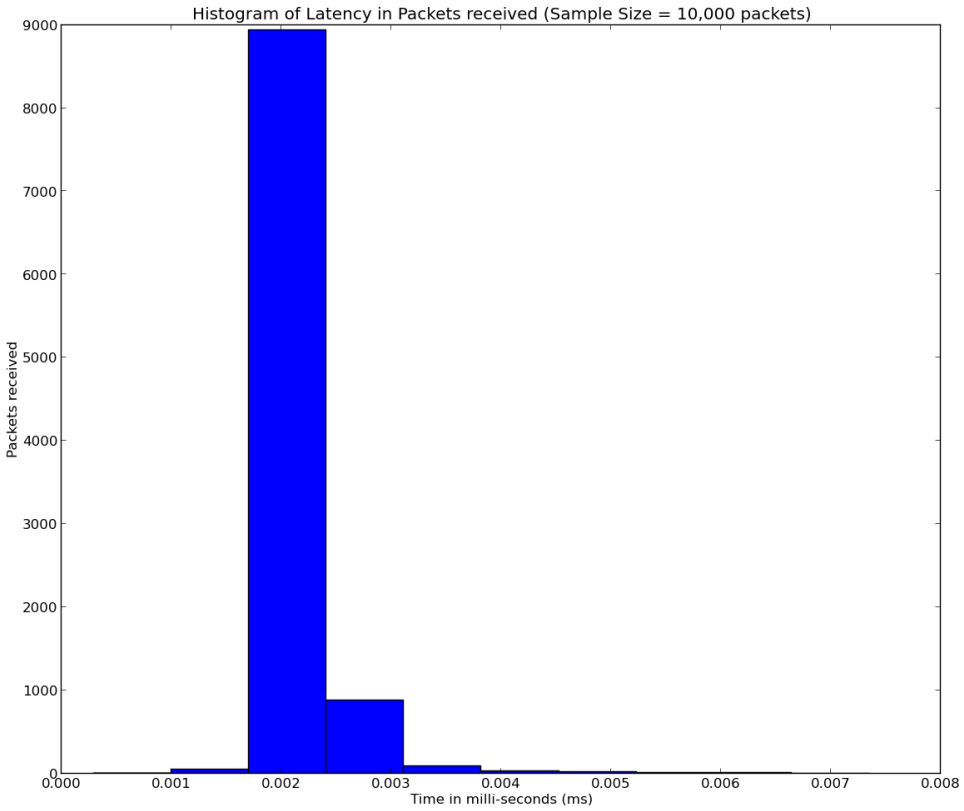


Figure 7.7: Histogram of latency values for 10,000 data packets received from Matlab to ROS. This grap shows that most of the data packets are received in a duration of 1 ms.

The armBase class provides a unified interface for creating an object to any dVRK manipulator (MTMR, MTML, PSM1, PSM2, PSM3, ECM) and so on. This allows less code clutter and only one class. I have relied on string manipulations to achieve this. The class allows adding up additional function callbacks to the subscribers and events so that class can easily be extended for use in GUI design and Simulink. The clean_up function takes care of node and topic termination from ROS, which is very essential for the dVRK interface. It also frees up memory and handles in the process.

I have been able to achieve > 500 Hz of publishing and subscribing speeds. A histogram of latency is shown in figure 7.7. The histogram executable is programmed in Python and checks for each data packets delay with respect to ROS time. This result is very encouraging as ROS-CISST/SAW interface works at the same speed. This would allow total control of dVRK using Matlab for users non-familar with ROS and C++.

Chapter 8

Conclusion and Future Work

I have developed many software interfaces for this research with a brief overview depicted in figure 8.1 which also shows where they fit. To implement the cisst-ROS bridge, shown in the middle of the figure, I had to modify the corresponding to the CISST/SAW libraries, this work was also supported by Zihan at JHU. For right side of the cisst-ROS bridge in figure 8.1, I added applications that were compiled with ROS. This was a milestone in ROS integration since a single application spawned all the required interfaces and made them available at hand.

Going forward, I added the Gazebo interface that allowed for dynamic simulations of the MTMs. I am currently working on adding the same for PSMs. In Gazebo interface, I extended an additional rqt_gui package to implement joint controllers with ease. Shortly after that, I integrated the MoveIt package with the dVRK to be able to conduct motion planning related experiments. Lastly, I have added the Matlab interface that is still to be tested for use by other collaborating researchers.

Motion planning for the dVRK has worked as expected in simulations, solving for problems in a minimal amount of time. For the experimental setup, I used pre and intra-operative planning, with the goals of having an entry path into the body

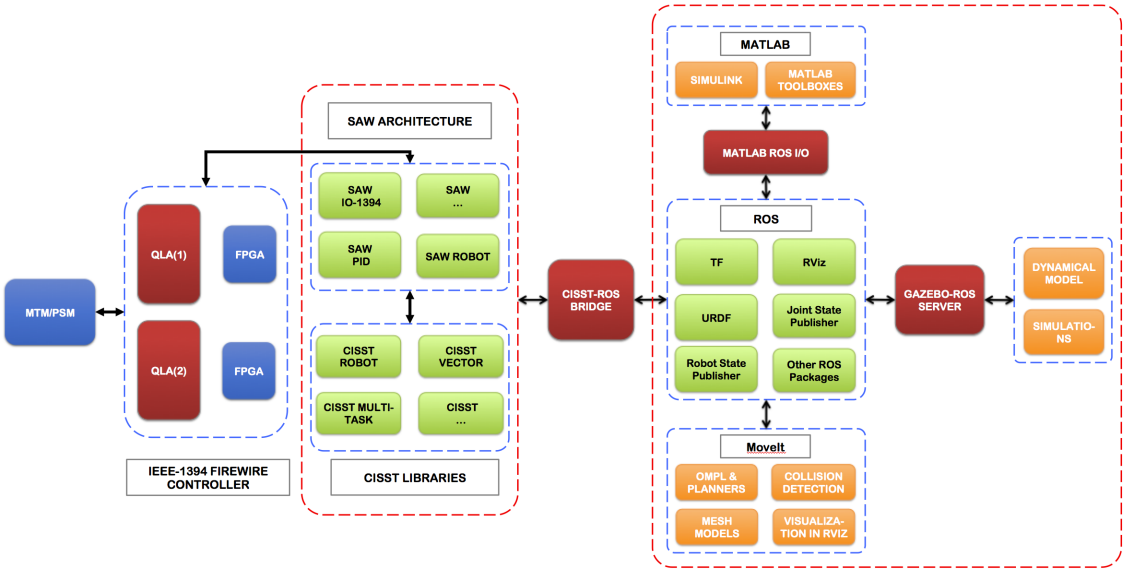


Fig: Showing the Flow Chart of the Extended DVRK Setup including Matlab, Gazebo and MoveIt

Figure 8.1: The Extended System Overview

and also planning inside the body using the single entry point. For the current experiments in simulations, the obstacles environment is registered to the PSM and thus allowing for path planners to perform valid collision and state checking. To evaluate the results on actual problems, accurate registration of the environment with respect to the PSM is required. There are a couple ways to achieve registration, such as pivot calibration using stereo vision or error tracking via optical tracking markers mounted on a camera and the PSM.

Zhixian Zhang (graduated from WPI in 2014) was working on the generation of 3D mesh models of the environment using stereo vision. He was successful in reconstruction >70% of the point clouds from the disparity maps. This work needs to be picked up and taken a step ahead by registering the generated mesh to the PSMs. Once the 3D generated mesh has been registered to the PSMs, the motion planning algorithms can be extended to control the actual PSM.

The dynamic parameters of the dVRK manipulators are unknown. As a future problem, the dynamics need to be estimated. Adaptive control techniques shall be utilized to improve upon the current estimate of the dynamic properties.

8.1 Implementation of Guidance Virtual Fixtures

In the field of Robot Manipulation, Virtual Fixtures, as the name indicates are set of imaginary points that pose as constraints to the movement of the manipulator. Virtual fixtures are used for various purposes and their use case classifies which type of virtual fixtures are needed. Essentially, virtual fixtures can be broadly classified into Forbidden region virtual fixtures (FRVF) and Guidance Virtual Fixtures (GVF).

Forbidden region virtual fixtures create a repulsive potential field around an area, enclosed surface or a volume. This repulsive potential field repels an manipulator as soon as it comes near the region. The stiffness of the potential field can be adjusted to make a hard constrained obstacles or a field whose repulsion gets stronger only as the manipulator get farther and farther inside the field.

Guidance Virtual fixtures are an altogether different approach as compared to FRVFs. In GVFs case, a defined path is used to generate an attractive potential field with a small propagation bias along the path. Based on the usage, the end effector of the manipulator is attracted to the path, with the force of attraction stronger as the manipulator deviates away from the path. Along this path, a force is also applied to the manipulator to move along the path.

8.2 Use Case of GVFs for the dVRK

Once a path has been obtained using the path planning algorithms as covered in section, taking into account the obstacles, the path will be used to generate a guidance virtual fixture. In order to do so, the dynamics of the manipulator must be know. The necessity of accurate dynamical model of the manipulator stems from the fact that the GVFs are used to apply virtual forces at the end effector. These end effector forces are applied by the joints themselves as torques, and are propagated towards the end effector link from the first joint in the base link.

If the dynamic model of the manipulator is accurately know, accurate forces can be applied at the end effector link which helps the surgeon get a feel of the path the manipulator intends to follow.

8.3 Use Case of FRVFs for Assistive Path Planning

As detailed in the section of assistive path planning for aiding the surgeon, a visual tool has been provided that allows the simulated PSMs and the environment to show up collision markers whenever collision happens between the tool and environment, in simulation. Adding FRVFs to the visual feedback will add to the safety and convenience of choosing a pair of start and goal points for assistive path planning. Moreover, the plan to add the FRVFs to the actual PSMs is intended. The FRVFs will be added only for the sensitive organs where the PSMs are not supposed to go in any way.

Bibliography

[1] Is ros for me. http://www.ros.org/is-ros-for-me/.

[2] Zeus robotic surgical system. http://allaboutroboticsurgery.com/zeusrobot.html.

[3] Simon Bann, Mansoor Khan, Juan Hernandez, Yaron Munz, Krishna Moorthy, Vivek Datta, Timothy Rockall, and Ara Darzi. Robotics in surgery. *Journal of the American College of Surgeons*, 196(5):784–795, 2003.

[4] Ryan A Beasley. Medical robots: current systems and research directions. *Journal of Robotics*, 2012, 2012.

[5] Charles L Bennett, Steven J Stryker, M Rosario Ferreira, John Adams, and Robert W Beart. The learning curve for laparoscopic colorectal surgery: preliminary results from a prospective analysis of 1194 laparoscopic-assisted colectomies. *Archives of Surgery*, 132(1):41–44, 1997.

[6] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2383–2388 vol.3, 2002.

[7] Herman Bruyninckx and Peter Soetens. The orocos project. http://people.mech.kuleuven.be/ orocos/pub/documentation/rtt/v1.12.x/doc-xml/orocos-overview.html, Jan 2015.

[8] BL Davies, RD Hibberd, MJ Coptcoat, and JEA Wickham. A surgeon robot prostatectomy-a laboratory evaluation. *Journal of medical engineering & technology*, 13(6):273–277, 1989.

[9] Anton Deguet, Rajesh Kumar, Russell Taylor, and Peter Kazanzides. The cisst libraries for computer assisted intervention systems. In *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions, Midas Journal*, 2008.

[10] Michel Gagner, Eric Begin, Richard Hurteau, and Alfons Pomp. Robotic interactive laparoscopic cholecystectomy. *The Lancet*, 343(8897):596–597, 1994.

[11] M Jung, T Xia, A Deguet, R Kumar, R Taylor, and P Kazanzides. A surgical assistant workstation (saw) application for teleoperated surgical robot system.

The MIDAS Journal-Systems and Architectures for Computer Assisted Interventions, 2009.

[12] S. Karaman, M.R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the rrt*. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483, 2011.

[13] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *CoRR*, abs/1005.0416, 2010.

[14] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[15] P Kazanzides, S DiMaio, A Deguet, B Vagvolgyi, M Balicki, C Schneider, R Kumar, A Jog, B Itkowitz, C Hasser, et al. The surgical assistant workstation (saw) in minimally-invasive surgery and microsurgery. In *MICCAI Workshop on Systems and Arch. for Computer Assisted Interventions*, 2010.

[16] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[17] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995–1001 vol.2, 2000.

[18] Y. Kuwata, G.A. Fiore, J. Teo, E. Frazzoli, and J.P. How. Motion planning for urban driving using rrt. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1681–1686, 2008.

[19] Yik San Kwoh, Joahin Hou, Edmond A Jonckheere, and Samad Hayati. A robot with improved absolute positioning accuracy for ct guided stereotactic brain surgery. *Biomedical Engineering, IEEE Transactions on*, 35(2):153–160, 1988.

[20] S.M. LaValle. In *Planning Algorithms*. Cambridge University Press, May 2006.

[21] David V. Lu. Joint state publisher package. http://wiki.ros.org/joint_state_publisher.

[22] A. Malpani, B. Vagvolgyi, and R. Kumar. Kinematics based safety operation mechanism for robotic surgery extending the jhu saw framework. *The Midas Journal*, 08 2011.

[23] MoveIt. Environment representation. http://moveit.ros.org/wiki/Environment_Representation/C 2013.

[24] MoveIt. Collision checking. http://moveit.ros.org/documentation/concepts/, Jan 2015.

[25] The Economist Online. Surgical robots: The kindness of strangers. http://www.economist.com/blogs/babbage/2012/01/surgical-robots, Jan 2012.

[26] Orocos. Orocos project history. http://www.orocos.org/orocos/history, Jan 2015.

[27] VR Patel, MF Chammas, and S Shah. Robotic assisted laparoscopic radical prostatectomy: a review of the current state of affairs. *International journal of clinical practice*, 61(2):309–314, 2007.

[28] THE ASSOCIATED PRESS. Surgical robot da vinci scrutinized by fda after deaths, other surgical nightmares. http://www.nydailynews.com/life-style/health/surgical-robot-scrutinized-fda-deaths-nightmares-article-1.1311447, April 2013.

[29] Morgan Quigley, Eric Berger, Andrew Y Ng, et al. Stair: Hardware and software architecture. In *AAAI 2007 robotics workshop*, volume 3, page 14, 2007.

[30] Gildardo Sánchez and Jean-Claude Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Robotics Research*, pages 403–417. Springer, 2003.

[31] Richard M Satava. Surgical robotics: the early chronicles: a personal historical perspective. *Surgical Laparoscopy Endoscopy & Percutaneous Techniques*, 12(1):6–16, 2002.

[32] Richard M Satava. Robotic surgery: from past to future—a personal journey. *Surgical Clinics of North America*, 83(6):1491–1500, 2003.

[33] Ioan A. Sucan and Sachin Chitta. Moveit. http://moveit.ros.org.

[34] Ioan A Şucan and Lydia E Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer, 2009.

[35] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`.

[36] Intuitive Surgical. Welcome to the da vinci research kit wiki community. http://research.intusurg.com/dvrkwiki/, Jan 2015.

[37] Foote Tully, Marder-Eppstein Eitan, and Meeussen Wim. Tf. http://wiki.ros.org/tf.

[38] Johns Hopkins University and Intuitive Surgical Inc. Surgical assistant workstation software architecture document. https://www.cisst.org/main/images/5/52/SAW-Architecture-V1.4.pdf, 2007.

[39] Balazs Vagvolgyi, S DiMaio, Anton Deguet, Peter Kazanzides, Rajesh Kumar, Christopher Hasser, and R Taylor. The surgical assistant workstation. In *Proc MICCAI Workshop: Systems and Architectures for Computer Assisted Interventions*, 2008.