Simulation of Teleoperation and Interaction Control of Serial Robot Manipulator Employing ROS Based Modular Approach

By

Abhra Adhikari

Examination Roll No.: M6IAR19011

Registration No.: 137301

Under the Guidance of

Prof. Ranjit Kumar Barai

Thesis submitted in partial fulfillment of the requirement for the award of the Degree of Master of Technology in Intelligent Automation and Robotics under Electronics and Telecommunication Engineering Department

Department of Electronics and Telecommunication Engineering
Jadavpur University
Kolkata – 700032
May, 2019

Faculty of Engineering and Technology
Jadavpur University
Kolkata – 700032

Date:

This is to certify that the thesis entitled "***Simulation of Teleoperation and Interaction Control of Serial Robot Manipulator Employing ROS Based Modular Approach***" has been carried out by Abhra Adhikari (University Registration No.: 137301 of 2016-17) under my guidance & supervision, be accepted in partial fulfillment of the requirement of the degree of Master in Technology in Intelligent Automation and Robotics under Electronics and Telecommunication Engineering Department of Jadavpur University.

Prof. (Dr.) Ranjit Kumar Barai
Supervisor
Electrical Engineering Department
Jadavpur University

Prof. (Dr.) Amit Konar
Course Coordinator
Intelligent Automation & Robotics
Electronics and Telecommunication Department
Jadavpur University

Prof. (Dr.) Sheli Sinha Chaudhari
Head of the Department
Electronics and Telecommunication Department
Jadavpur University

Prof. (Dr.) Chiranjib Bhattacharya
Dean, Faculty Council of Engineering & Technology
Jadavpur University

Faculty of Engineering and Technology
Jadavpur University
Kolkata – 700032

CERTIFICATE OF APPROVAL *

The foregoing thesis is hereby approved as a credible study of engineering subject to warrant its acceptance as a pre-requisite to obtain the degree for which it has been submitted. It is understood by this approval the undersigned do not endorse or approve any statement made, opinion expressed or conclusion drawn therein , but approve the thesis only for the purpose for which it is submitted.

External Examiner

Prof. (Dr.) Ranjit Kumar Barai

Supervisor

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC THESIS

I hereby declare that this thesis titled "Simulation of Teleoperation and Interaction Control of Serial Robot Manipulator Employing ROS Based Modular Approach" contains literature survey and original research work by the undersigned candidate, as part of his Degree of Master of Technology in Intelligent Automation and Robotics.

All information have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: Abhra Adhikari
Examination Roll No: M6IAR19011

Thesis Title: SIMULATION OF TELEOPERATION AND INTERACTION CONTROL OF SERIAL ROBOT MANIPULATOR EMPLOYING ROS BASED MODULAR APPROACH

Date:
Place: Kolkata

Signature of the candidate

ABSTRACT

Despite of the availability of highly sophisticated techniques and increasing computing capabilities, the problems associated with robots interacting with environments remains an open challenge. Despite great advancement in the field of autonomous robotics, there are certain situations where a human in the loop is still needed, especially in the hostile areas and robotic surgery based operations. The very fact lies behind is that current technologies cannot solely & reliably accomplish all piece of work autonomously.

This thesis presents a simulation environment for performing teleoperation of a serial robot manipulator and controlling its interaction with environment in its workspace in a master slave approach where the operator provides a position trajectory using a joystick or even using an interactive designed GUI. The novelty behind this approach lies in the fact we have selected Robot Operating System (ROS) as the underlying platform for all the communication and Gazebo as the simulation environment and ROS being an open source the solutions would be more accessible maintainable & extendable using high-quality code from the active ROS community. Also working with real hardware performing interaction control in hard real time can pose many difficulties in term of programming and cost which can be easily worked on first hand using a realistic and dynamic simulation environment like Gazebo alongside with highly modular software platform like ROS.

Key words: Robot Operating System, Gazebo, Compliant Control, Interaction Control Interactive manipulation.

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement made the work a success. I would like to express deep gratitude and respect to all those people who guided, inspired and helped me for the completion of my project work.

It is a great pleasure to express my deepest gratitude to my project guide **Prof. (Dr.) Ranjit Kumar Barai**, of Electrical Engineering Department, for giving me the opportunity to work on this project. I would like to acknowledge my sincere appreciation for his valuable advice, phenomenal guidance, and constructive suggestions throughout the course of the thesis work. I was at greatest liberty to exercise thoughtful and scientific approach to the problem.

I express my sincere thanks to **Prof. (Dr.) Sheli Sinha Chaudhari** Head of the Department of ETCE for providing me the necessary facilities in the department. I would also like to express my sincere thanks to **Prof. (Dr.) Amit Konar Course Coordinator of Intelligent Automation and Robotics course under** Department ETCE **for his help & guidance all through the course.** I am also thankful to all the staff members of the Department of ETCE and to all my well-wishers for their inspiration and help. I would like to thank my lab mates for providing me moral support towards the fulfillment of this project work.

Finally, and most importantly, I wish to express my deep gratitude to my parents, who always stood by me throughout my life and guided me in my time of crisis giving me inspiration and encouragement in writing this thesis

Date:

Place: Kolkata

Abhra Adhikari
M.Tech. in Intelligent Automation and Robotics
(E.T.C.E)

Examination Roll No: M6IAR19011
Registration No: 137301 of 2016-17

Contents

Contents

List of Figures

List of Figures

List of Acronyms

EEF	End Effector
ROS	Robot Operating System
URDF	Unified Robot Description Format
XML	Extensible Markup Language
GUI	Graphical User Interface
DOF	Degree of Freedom
FT	Force Torque
IK	Inverse Kinematics
ODE	Open Dynamics Engine
IDE	Integrated Development Environment
TF	Transform in ROS
POS	Position & Orientation
UR	Universal Robot
PUI	Physical User Interface
QT	Quick Time
JT	Jacobian Transpose
CM	Controller Manager
RViz	Robot Visualization
API	Application programming interface

Chapter 1

Introduction

"Robot, any automatically operated machine that replaces human effort, though it may not resemble human beings in appearance or perform functions in a humanlike manner."

Encyclopdia Britannica

1.1 Background

In the few decades conventional robotic systems have been employed in various industries. In most cases these systems have consisted of manipulator arms whose function has been to perform desired tasks using their end-effectors. These tasks have involved programming end-effector motion trajectories and then controlling individual joints to produce the desired motion at the end-effector. This has been accomplished by using kinematic relationships between the end-effector pose and the joint angles and by including the dynamic properties of the system in addition to the kinematics. A characteristic of many of these tasks is that they only involve motion in free space. That is, the end-effector and the manipulator links are not in contact with the environment.

At present times robots have started to evolve from their ordered, rigid environment of factories and find their place in close space to humans. Robotic bodies are already capable enough to swim, fly and most recently there have been breakthroughs in the development of bipedal robots. Intelligence in robots is also developing rapidly, but at the same time they are limited by their cognitive capabilities. For most of the industrial applications robots are made to do a certain task mostly in highly ordered environment. At present there is a burring need for human robot collaboration. Instead of putting the intelligent processor on the robot we can simply amend it with human teleoperation. Teleoperation is a form of remote control and is the least version of supervisory control interaction. During teleoperation the robot's movements are determined not just by its environment and inner state, but rather the human operator.

From the different robot control types each having its pros and cons to accomplish a particular task, the one that is relevant for this thesis is controlling a robot arm using teleoperation. The simple way to explain it is by taking the origin of the word *teleoperation*; the word comes from Greek origins where the prefix *tele* means *at a distance* and suffix *operation* means, "operating from a distance" [1]. For practical understanding it is a technique that involves execution by hand from a distant place. The reasons for the use of teleoperation are justified in the sense that the required operation must be performed by a human due the unpredictability of the tasks, and for the same reason are difficult to create a model and pre-program the tasks

A teleoperation system is based in the master-slave architecture; with the master system situated in a control room (local environment), and the slave system situated in the remote environment, both systems require communication and data processing due the distance between them. In the system, the human operator acts upon the master system (local task objects) and is guided with the information feedback (display) in order to execute the task through the remote manipulator [2].

The direct and continuous interaction of the human operator with the control of the master-slave system is commonly referred as *teleoperation*. If the teleoperation system uses a computer as intermediary for the human operator to supervise the task, is refers as telerobotics [3]. Both terms can be used interchangeably, but the term teleoperation encompass the term telerobotics.

The human interface can be as simple as a keyboard, mouse for input; and a computer screen for output. A bit more ergonomic option is to use joysticks or console controllers. These methods are relatively cheap and can be used to accurately control the position trajectories of a slave manipulator.

1.2 Problem Statement & Objective

The problem statement for the current thesis was to design a simulation environment depicting master slave teleoperation of a serial robot manipulator primarily to control the interaction of the robot with the environment.

Building an experimental robotic setup can be tedious, prone to hardware faults and involve large expenses. A common way to overcome some of these problems is to model a part or entire system in simulation environment. However, implementation of teleoperated robotic systems with interaction control manifests additional problems. The human robot interfaces should exist in hardware and this in turn requires simulated system to work in proximity with a real-time. Hence to overcome these difficulties the Robot Operating System (ROS) [4] is being utilized as the underlying platform to carry out the entire simulation setup by using Gazebo which is 3D multi robot simulator. ROS supports a highly distributed and modular design enabling us to choose from a large variety of user written packages that is relevant to our need. More over ROS is open source & free hence appropriate for simulation and testing of any concept prior to be tested in real time scenario. ROS supports a distributed runtime environment that facilitates the working of different executable at the same time thus each one of them runs in parallel manner & can be tested or modified independently without analyzing the entire system. Gazebo supports ODE (Open dynamics engine) and other physics engines for handling the dynamics of the system as closely as a real robot interaction hence simulation in Gazebo environment and its ROS supports enables the design a fully robust and realistic design in our case.

1.3 Motivation

Decades of research and development in robotic manipulation has led to an incredible impact scientifically, socially and economically. Large manipulators move with sub millimeter accuracy at daunting speeds and have since long begun to relieve the human workforce from repetitive, non-ergonomic and dangerous tasks. The cost and the capabilities of robotic manipulators have so far limited their use to large-scale industrial production facilities, e.g. the auto manufacturing industry. Most such applications are characterized by a precise and deterministic task description and accurate position control which aims at rejecting any external forces applied on the robot.

However in contrast, humans are capable of manipulation in uncertain environments, and exhibit physical compliance when subjected to perturbing forces. This compliance is realized by elasticity in the muscles that drive our limbs. We can vary the compliance by co-contraction of opposing muscle pairs. In this way, the compliance can be varied between different tasks and also during tasks. Human studies show that learning impedance variations are an important part of mastering a manipulation task (Burdet et al., 2001; Selen et al., 2009). While traditional robot manipulators are inherently stiff due to their drivetrains, there is currently a strong trend toward torque-controlled, light-weight robots. These robots have been developed for quite some time and have now reached the level of maturity required for real applications. The availability of this class of robots opens a wealth of new possible applications, both in the industrial sector but perhaps even more so in the service robotics sector which involves health care and assistance applications. However, without suitable control these arms are no more useful than their predecessors. Indeed,
for a robot to perform useful work, the controller and the task model is at least as important as the hardware.

Thus for studying interaction control compliant or force control becomes a necessary approach. This thesis emphasizes on compliant manipulation tasks based on cooperative or interactive manipulation.

1.4 Thesis Outline

The objective of this thesis is to design a simulation environment depicting complaint control of a robot arm in contact and controlling the interaction forces generated at the contact point. This is achieved using a master slave based teleoperation approach where the human user controls the robot trajectory and force at the end effector when the robot is making a steady contact. The thesis proceeds as follows:

Chapter 1 *we* start with the introduction to the selected research domain giving an overview of the topic, problem statement and objectives, the underlying principles that inspire us work in this field and the hierchary of our thesis.

Chapter 2 focuses on the literature survey in the selected area of work, by giving an overview of various robot control methods in the related area of interest, recent advancements in the topic and ends with a summary of the survey.

Chapter 3 describes the problem statement of the thesis, challenges in this domain, the limitations under which the entire simulation environment works and the problems faced during work.

Chapter 4 deals with the approach and implementation. It describes the software development process, the software, technical & structural architecture, overall date flow of the system, user interface for the user end.

Chapter 5 deals with design and development process. It starts with the simulation environment design, the overall system architecture, simulation of UR5 robot manipulator, writing a plugin for simulating a 6 axis Force/torque sensor, modeling the compliant environment as a massless spring damper system, designing a virtual joystick interface for the master side & a user interface for force control and testing the working of the simulation in real time.

Chapter 6 presents the results of the experiment under different conditions.

Chapter 7 concludes our thesis by giving a thesis summary and presents the future scope and modifications that can be made with the design.

References

Chapter 2

Literature Review

This literature review discusses about robotic control in the literature, giving attention to position control of a robotic arm in free space and involving interaction with the environment with some relevant applications.

First, a general overview of various manipulator control architectures is discussed before taking the focus to methods incorporated to deal under constrained situations when the robot make a valid contact with the environment. Historical progression from explicit force control to more modern methods of controlling forces and positions simultaneously are presented as well as a look at recent advances in the literature. Although non-contact control is used in conjunction with the framework, the origins of this control will not be discussed in this literature review. This is because the framework uses proven and easily accessible methods to control the manipulator when not in contact with the environment. These methods incorporate kinematic calculations, motion trajectory planning, and collision avoidance and are achieved via the Robot Operating System (ROS) and will be discussed in the approach & implementation chapter. A brief history about Robot Operating System lastly, applications of robotics in the industries are considered and the current state of the art and innovative applications in the industry are reviewed.

2.1 Overview of different manipulator control schemes

The first form of reprogrammable robotic control in an industrial robot was the Unimate robot, patented in 1954 [5], and was used by General Motors for spot welding [6] and other tasks. Although machines capable of completing assembly line tasks existed before the Unimate, they were typically designed for a single task and therefore only such tasks that were feasible for tasks that ran long term. The Unimate, on the other hand, could be "taught" to learn a new task without reconfiguration of the robot. This teaching process did not require a skilled technician to reprogram the Unimate. To teach the Unimate, an operator would move the robot to each position required to complete a task by using a simple control box seen in Figure 2.1

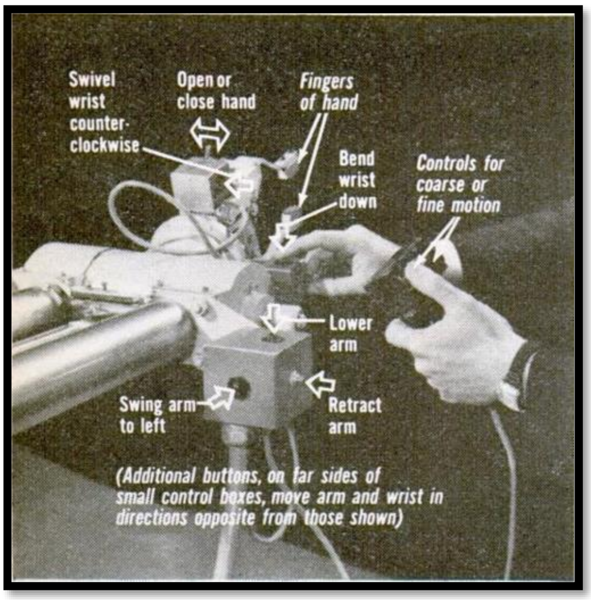

Swivel wrist counter-clockwise
Open or close hand
Fingers of hand
Bend wrist down
Controls for coarse or fine motion
Lower arm
Swing arm to left
Retract arm
(Additional buttons, on far sides of small control boxes, move arm and wrist in directions opposite from those shown)

Figure 2.1 **Operator teaching the Unimate** [7]

This process of "teaching" a robot by recording positions is still used in the industry today, but more advanced position control schemes have also been developed. In modern robotic control schemes, the joint position controllers is typically a path planner that determines a stream of appropriate joint position commands, or a trajectory, that will take the robot from one set of joint positions to another. While generating this trajectory, the path planners also avoid known obstacles in the environment and positions in which control laws break down or behave undesirably, e.g. at travel limits or near singularities.

The Unimate and many other industrial robots solely used position commanding to complete its task and do not take into consideration other important information that can be retrieved via sensors, e.g. cameras or force sensors. For more complex tasks, this sensory information can be used to avoid barriers to motion that have been introduced to the robot's workspace, or to compensate for uncertainty. This lack of sensory information makes pure position commanding useful only for operations where the manipulator has minimal contact with the environment. Once there is contact, slight deviations in a robot's desired position and its actual position, or deviations between the model and actual environment, lead to dangerously high contact forces. For this reason, a robotic system must take into account model inaccuracies by sensing contact forces on the robot. This problem spawned the field of force control which commands the robot based on sensed forces. According to Whitney [8], one of the pioneers of force control, “gross motions”, i.e. open loop position control motions, are useful for “material handling tasks as well as ‘assembly’ tasks such as spot welding in which insertions of one part into another are not necessary”, but fine motions, i.e. closed loop motions based on force feedback, “are required for some types of assembly requiring insertions, push and twist actions, gear meshing, packing, and so on.”.

The simplest way to apply force control is to command robot joint positions only based upon the knowledge of current and desired EEF contact forces. This can be useful in applications where a precise force is desired to be maintained. Such a method was proposed by Whitney and termed linear force feedback strategy. In this method, a force feedback matrix, a matrix of feedback gains, is used to convert sensed forces into desired EEF position deltas which are then converted into new joint position commands [8]. This process is illustrated in Figure.2.2

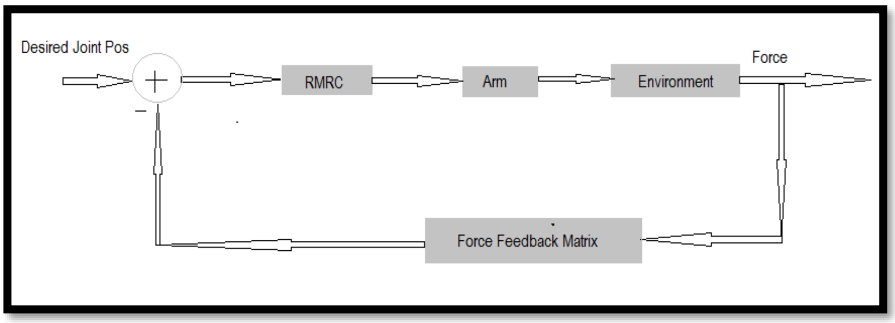


Figure 2.2 **Block diagram of force feedback control method** [8]

This literature review will focus on control algorithms that contain elements of both position and force control methods. This middle section is the area of active compliant control. Active compliant control attempts to track a trajectory, i.e. position control, while maintaining compliance with respect to physical contact (intentional or not).

It is worth mentioning that there is another area of compliant control called passive compliant control. Active compliant control is attained through software whereas; passive compliant control requires that the robot or EEF is inherently mechanically compliant. Passive compliance is achieved using a spring, clutch or other compliant device between the EEF and last link of the robot manipulator [9], in the EEF itself [10], or in each actuator [11] [12]. In passive compliant control, the robot is position controlled, and the compliance of the robot itself allows for the contact forces to be minimized. While variable stiffness actuators have been investigated [13], no physical system was identified that would give a passive controlled robot the range of performance in terms of precision, payload, etc. necessary for the applications proposed by the sponsor. Furthermore, active compliant control techniques can be implemented on proven affordable industrial manipulators, whereas multi-purpose passive compliant control architectures, exemplified by Rethink Robotics' Baxter [14], have only been available on the market for a few years.

Figure 2.3 shows the types of manipulator control. In the domain between explicit position control and explicit force control, there are two main types of control one of them belongs to the category of split control, where the control method attempts to use explicit force and explicit position control separately in different Cartesian directions so that the robot may reach a desired position while also being compliant. The other, falls in the category of relational control where the control method tries to maintain a dynamic relation between the position of the EEF and contact forces in all directions.

In between these two types of control there is hybrid impedance control which attempts to concatenate the benefits of both types of compliant control.

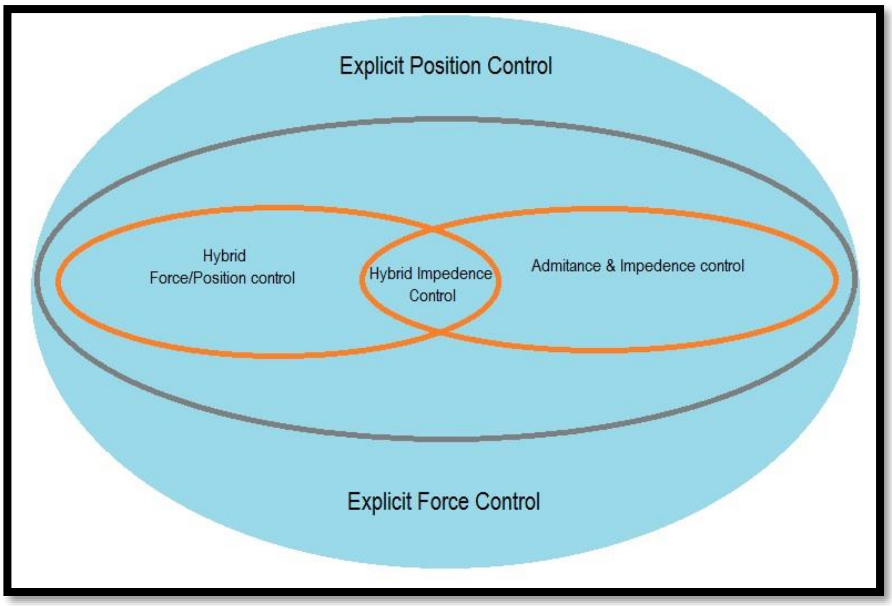


Figure 2.3 **Types of Manipulator Interaction Control**

2.2 Compliant force control

To manipulate objects in the environment or to avoid accidental contact with the environment, robotic control must implement more than just simple position control, it must also control the contact forces between the EEF and the environment. Force control uses sensory force and torque information to adjust the position of an EEF to maintain or reduce forces at the contact with the environment. Explicit force control relies solely on force data to calculate the desired robot movement that will maintain a desired force. This type of control can be very useful in certain applications, but its primary disadvantage is that it pays no attention to a desired position of the EEF. In a lot of cases, the operator would like to maintain a desired force profile while also staying as close as possible to the desired position of the EEF.

2.2.1 Formalizing the Problem of Compliant Control

Mason formalized the problem of controlling contact forces while maintaining a desired position in 1981 as *compliant motion*. According to Mason, "compliant motion occurs when the position of the manipulator is constrained by the task"[15]. Mason focused on active compliance solutions rather than passive compliance.

Mason introduced the concept of a *C-surface* which is "a task configuration space which allows only partial positional freedom" [15]. A C-surface is the intermediate between two extremes, total positional freedom and no positional freedom. While not in contact with the environment, a manipulator has complete positional freedom. On the other extreme, a manipulator rigidly attached to a stiff object has no positional freedom and has complete freedom to control the forces on the EEF. While in contact with a C-surface, a manipulator must consider both position and force control.

Mason went on to develop a method for breaking down the natural constraints of ideal C-surfaces and adding artificial constraints that the operator would like to enforce. The simplest example of this is a manipulator following the surface of a table. A natural constraint is that the velocity of the manipulator in the direction that is normal to the surface of the table must be zero since it cannot move through the table. An artificial constraint constrains the velocity normal to this natural constraint, i.e. the velocity that moves the EEF along the surface of the table, to the desired velocity. Mason's efforts to create this language describing contact tasks was meant to allow future work of synthesizing control strategies that enforce these natural and artificial constraints.

2.2.2 Hybrid Position/Force Control

Active researchers proposed two methods for achieving compliant motion in the early 1980s which are still used today. First, Raibert and Craig developed hybrid position/force control [16]. Raibert and Craig took the direct logical step from Mason's formalized constraints to synthesize a control strategy. They set up a Cartesian frame that described the natural and artificial constraints and used it to pick directions controlled by explicit position control and directions controlled by explicit force control. Then, they developed a method of transforming these control requirements to direct control of each individual robot joint. This method can be seen in equation 2.1.

$$\tau_i = \Sigma^{N}_{j=i} \{\Gamma[s_j \Delta f_j]+\psi_{ij}[(1-s_j)\Delta x_j]\} \text{ ....}(2.1)$$

where τ_i is the torque applied to the *i*th actuator, N is the number force controlled degrees of freedom in the Cartesian reference frame plus the number of position controlled degrees of freedom in the Cartesian reference frame, Δf_j is the force error, Δx_j is the position error, Γ_{ij} is a force compensation function, ψ_{ij} is a position compensation function, and s_j is a binary (0 or 1) vector that indicates which degrees of freedom are force controlled [16]. The equation takes the Cartesian force and position control efforts and maps them to the torque required to be applied to each joint to accomplish the control goal. Raibert and Craig implemented the controller on a 2 axis Scheinman manipulator to show that the controller was feasible and stable. However, it is important to note that Lipkin and Duffy refuted this method in 1988 because it is "based on the metric of elliptic geometry and is thus non invariant" with respect to Euclidean collineations and change of Euclidean unit length [17]. Lipkin and Duffy, along with others proposed new invariant hybrid position/force control methods to attempt to solve the issues reported with Raibert and Craig's version [17] [18] [19].

2.2.3 Impedance Control

The literature usually groups the other method of achieving compliant motion into a category called impedance control. Though there are many different control schemes thrown into this one category, e.g. stiffness control [20] and admittance control [21], the term impedance control comes from Hogan's work in the field [22]. Hogan pointed out that while in general absolute control of EEF position is desired, no controller can make up for the fact that the robot in contact with a physical system must behave according to the physical laws of the combined system. For this reason, the controller should command the desired motion of the manipulator, but also help it to react to disturbances that it encounters. According to Hogan, in impedance control "the controller attempts to implement a dynamic relation between manipulator variables such as end-point position and force rather than just control these variables alone" [22].

In Hogan's implementation of impedance control, he derives the following equation (2.2) for the desired relationship between the contact force and the dynamics of the system.

$$F_{int}=(X_0-X)+B(V_0-V)-MdV/dt \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2.2)$$

where F_{int} is the "interface" force, K is and adjustable stiffness variable or nonlinear function, B is an adjustable damping variable or nonlinear function, (X_0-X) is the difference between the commanded position and the actual position, (V_0-V) is the difference between the commanded velocity and the actual velocity, and M is the inertia tensor of the manipulator. Therefore, according to Hogan, the equations of motion for the manipulator coupled with the environment are seen in equation 2.3.

$$\frac{(M_e+M)dV}{dt} = (X_0-X)+B(V_0-V)+F_{ext} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.3)$$

where *Fext* is the external force, or the force that the manipulator will need to apply to maintain the desired dynamic behavior, and M_e is the inertia tensor of the environment. In order to implement equation 2.3, one must have precise knowledge about the inertia of the environment and robot.

2.2.4 Bridging the Gap between Hybrid Control and Impedance Control

While the initial efforts by Raibert, Craig, Hogan and other early pioneers of compliant control individually gave a few possible solutions to the problem of controlling force and position together, researchers eventually realized that by combining the efforts there might be an even better solution to the problem. In 1988, Anderson and Spong applied these two methods to one control strategy and called it hybrid impedance control [23]. Not only did the control combine impedance and hybrid position/force control, but it also implemented an outer/inner loop of control so that the compliant control may be done separately from the inverse dynamics calculations. Lui and Goldenberg studied this control method further in 1991. They made it more robust by the use of the computed torque technique and a PI control law to compensate for model uncertainties [24].

2.3 RECENT FORCE AND COMPLIANT CONTROL EFFORTS

Since the early 2000's, a dramatic increase in computing power and expanding robot infrastructure have influenced efforts to improve force and compliant control in the literature. With the rise of faster computers, the academic community has renewed its interest in more advanced control methodologies and design that allow for greater robustness and stability, especially in cases of uncertainty. This resurgence has also included the fields of artificial learning, and neural networks. Recent literature in the field of force and compliant control schemes mirrors these advances in the state of the art of control and computing.

One issue with the original idea of impedance control is that there will be uncertainty in the robot model and especially in the model of the environmental stiffness. This issue makes it difficult to perform robust force tracking. Jung proposed a force tracking impedance control scheme that uses an adaptive control philosophy to adjust the velocity profile during motion as a function of the force error [25]. Jung showed that this controller works well in unknown environments and when the environment stiffness is abruptly changed. Researchers have also made similar efforts for adapting to unknown parameters using neural networks. In [26], a neural network is used to adjust an impedance controller for unknown environments. In [27] a neuro-adaptive controller is used to track position and force along a flat surface with non-parametric uncertainties in the models of the robot and environment. Another innovative advancement is the use of model-free reinforcement learning and optimal control to learn variable impedance for a robotic system. Buchli [28] developed a method to allow a robot to learn variable impedance so that the robot may be compliant when able, yet stiff when required. Lee [29] took a biological approach to impedance control. By looking at the way humans interact with objects, Lee developed a control algorithm that adapts the arm stiffness based on the force error and interestingly even allows for negative stiffness.

Researchers have also made many efforts to enhance robustness of impedance control. Jin [30] used time delay estimation and ideal velocity feedback to allow for nonlinearities in robot dynamics without actually modeling them. He showed that the controller improves robustness in cases involving nonlinear friction and allows for relatively simple tuning. Another approach using time delay estimation, [31], attempts to improve robustness without sacrificing accuracy using internal model control. Another example of an effort to enhance robustness can be found in [32], where researchers attempt to improve robustness of task space impedance control on a redundant 7 degree of freedom (DOF) manipulator. Kikuuwe [33] attempted to deal with robustness in cases where the robot actuators become saturated by using a proxy-based sliding mode controller. According to the author, this saturation can result in “undesirable behaviors such as oscillation, repeated overshoots, and instability” [33].

The literature shows that compliant control is a very mature topic dating back to the late 1980’s. Research has been done on both passive and active compliance, and even passively compliant devices that actively change their stiffness. Active compliance has advanced to learn and adjust compliance automatically for specific tasks. The real barrier between the academic research and industrial application is generality. While researchers have studied these learned compliant behaviors in academia and applied them to specific situations, a factory worker, or even a non-expert programmer, cannot program a commercially available robot to behave in this manner. Until roboticist develops general, non-task-specific, applications of active compliant control to be easily adopted into the current state of industrial automation, most industrial automation processes will be limited to the traditional learned position procedures that industrial automation has used since the invention of the Unimate.

2.4 Advent of Robot Operating System

Robot Operation System (ROS) came into development in 2007 at Stanford Artificial Intelligence Laboratory [34] 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 leads 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 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 [35]. The demographic (Figure 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.

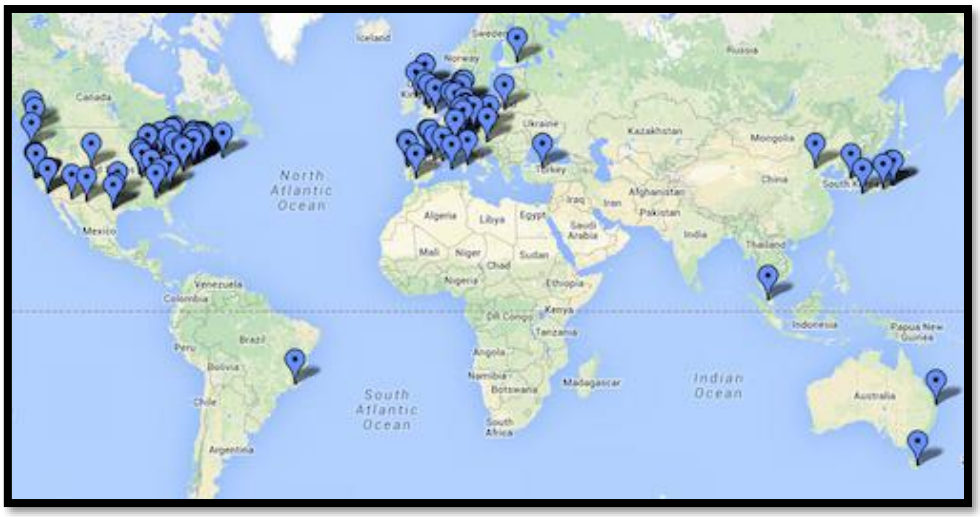


Figure 2.4: **Usage of ROS demographics as of 2011**

2.5 SUMMARY OF LITERATURE REVIEW

Researchers developed the fundamental ideas used in modern applications of compliant control in the early 1980s. Most force and compliant control, even in modern systems, revolves around force/position hybrid control, impedance control, or some combination of the two. Although the initial theories rely highly on setting up the task space and knowing a precise model of the robot dynamics and the environment, later research expanded compliant controllers allowing them to learn stiffness and function in uncertain environments. Rapid advances in computing technology led to resurgence in the fields of control and artificial intelligence. The robotic community used these advances and integrated them into the ideology of compliant control.

Chapter 3

Problem Description & Challenges

3.1 Overview

In this chapter we present to the reader the problem that we try to solve; the scientific importance of it and the difficulties behind it. Robot control is essential in robotics and in the current scenario where there is burying need for cooperative manipulation [] it is essential that the robot motion is compliant as possible. We are focused on a targeted setup problem.

In particular we are interested in designing a simulation environment depicting a master slave teleoperation thus controlling a slave manipulator when interacting with an environment.

With the increasing complexity of tasks that are required of manipulators contact with the environment has become more common whether we are seeing robots working in hostile environments or in medical applications these days interaction control is of significant importance. Additionally, applications for robots have expanded beyond traditional industrial settings into close human environments. There is a large need for Physical interaction between robots & the humans, and hence the environment has to be taken into consideration. Therefore controlling the robots trajectory solely using position control approach will end up in damaging the environment or the manipulator itself. Thus control strategies should be designed that deal with these situations in a safer way & performing the required task.

The motion of a robot when it is in contact with the environment is often referred to as constrained motion as the motion is not free but rather constrained by the environment [54].

For the presence of environmental constraints the control scheme that have been adopted are often referred to as compliant motion control strategies since the robot must be controlled in a manner that is responsive and compliant to the environment [55]. The goal of these control strategies is to successfully perform the desired tasks without compromising the robot or the environment. While being constrained by the environment the robot must control the motion and contact forces simultaneously and responsively.

3.2 Problem Description

The problem statement has been already stated in the chapter 1 under section 1.2

Main objective of my thesis was to incorporate Robot Operating System as the underlying platform for designing a simulation in Gazebo in order to visualize the teleoperation of a serial manipulator and studying its control of interaction when the end effector is making a valid contact.

Thus it is required to assemble a ROS (Robot Operating System) software package for controlling a serial-chain robotic manipulator in Cartesian space using positional data from the robot & force-torque (FT) data from the FT sensor at the wrist of the robot. The software bundle should be capable of reusing existing functionalities (sub-packages) to take the full advantage of the built-in modular approach of ROS framework. To make different software packages work which were originally not meant to work together or with particular robot, can be challenging in many cases. ROS certainly helps to overcome many hurdles but we also need to bear in mind that it is difficult to make fully hardware-independent software but in doing that inside a simulation environment some assumptions & generalizations needed to be made.

In order to combine already developed ROS packages and make them function together, improvements have to be done to existing solutions. Furthermore, profound understanding of how current software packages work is a prerequisite for enhancing and binding different modules. To use some existing packages in my work needed some tweaks accordingly. In addition to these technical modifications, various of functional adjustments has to be done to extend the stability and compatibility of contact control framework that will be discussed in section

3.3 Open Problems in the selected area of work

The goal of Teleoperation is to allow a user to remotely control a slave robot through a master device while maintaining some degree of transparency from the remote environment. Such a system can be of great potential, but the challenging task is to connect master and slave module in a coherent manner. While the master module is run by a human operator, the slave module (manipulator) is often present in an unknown and dynamic environment. The nature of interaction between a robot & the environment has a great influence in the overall system performance.

For successful teleoperation of a robot arm when the arm is acting as slave and control signals are provided by the user from the master end a using joystick or keyboard three main criterions that determine the performance of a teleoperation process. They are *transparency time delay* and *stability*.

As long as the slave manipulator is moving in free space and the motion is not constrained telepresence in term of transparency can be maintained by using visual feedback of the slave side using any vision system. Our interface has that facility as the user can get a view of the world in the slave side and thus using visual feedback change the end effector trajectory accordingly(refer to chapter 5). But as soon as the arm interacts with an environment transparency refers to the capability of the telerobotic system to make the human operator feel as if directly interacting with the remote environment. It is considered that the remote environment is expected to be unknown. Some approximations need to be done in order to model such remote environment and therefore, environment modeling is also an important part in teleoperation based interaction control. Master slave systems are usually kinematic dissimilar which leads to a coupling in the task space. The master device sends pose commands to the end effector of the slave manipulator. This requires a precise solution of the slave IK problem. Force feedback capabilities should be actively present in the system such that the user can feel the environment in a much more interactive manner especially needed for medical applications. This has to be achieved using bilateral teleoperation [ ] where there is force-force analogy between the master and the slave units. There should be local force control on either side of the system. However, in practice these conditions are not easily met especially, when the frictional forces and modeling error are taken into consideration.

Another inherent characteristic of teleoperation systems is the time delay in the communication link. For active force control techniques which fully depends on control loops this problem can be significant.

In order to achieve telepresence transparency of the teleoperation system has to be realized, i.e., accurate position tracking while operating in in free space and force or impedance matching during interaction

One difficulty in controlling robots in contact involves maintaining stability. In particular, instability arises during contact with stiffness environments [6]. Since the system response to contact forces must be fast in this case, sampling time is often a limitation when implementing a stable controller. Also, the environment that is in contact with the robot is typically not easily modeled.

3.4 Challenges faced during our design:

The objective of robot teleoperation is to move the end effector in the Cartesian space (world frame) by using a user interface like joystick or keyboard.

My design uses a virtual joystick interface for controlling the end effector in the world frame. The virtual joystick interface is based on the *ros_qml* package []. However the interface could only publish twist messages as it was originally designed to maneuver a differential drive robot. In order to make that interface to actually move our manipulator's end effector separate ros node is written that takes in account of kinematics of the robot involved uses the ROS's inbuilt IK solver and MoveIt for motion planning to actually move the robot in Cartesian space.

Instead of using a position or a velocity controller for the actuation of the joint of robot model in gazebo, Effort controllers can be used to control the individual joint. In that case gravity compensation techniques need to be implemented in order to drive the individual joints against the gravity acting down.

When a force torque sensor is affixed with the wrist of a simulated robot force sensor will make false readings as it is takes into account of the weight of the last link. In order to compensate for that a virtual link is defined inside the robot description format which is used to compensate for the weight of the last link. A force control loop can be also used to compensate for the weight of the link.

Our design did not have active force feedback capabilities hence to teleopearate the robot arm in Cartesian space using contact force data we had to rely on the real time GUI that ROS has based on *rqt.* If the end effector is pointing down along Z axis and making an interaction with the environment a response of reaction force along the Z axis can be used to identify any sorts of contact with the environment.

To reduce complexity we have only used translational force values for force calculations

Lastly to simulate a virtual mass spring damper system or mass less spring damper system inside gazebo in order to simulate it as a compliant surface for testing interaction control. A custom model is defined in gazebo having its mass damping properties set accordingly. Also the object can be simulated as prismatic joint to visually see the spring like effect when the robot makes a contact with the object.

Chapter 4

Approach & Implementation

4.1 Overview

In this chapter, we describe the approach of this thesis by presenting the main ideas to reach our goals by designing a simulation environment and trying to control a simulated robot arm in a remote location using teleoperation. The robot is made to interact with a virtual object in the environment. The goal was to successfully design a master & slave unit running parallelly in real time taking into account certain design aspects:

- **Realistic simulation** - The realism can be defined by the similarity of sensor data and consequences of robot actions between simulation and reality. The realism of the simulation has many different aspects. The first group of aspects is about the realism of sensor data. Gazebo which is an open source 3D robot simulator enables us to simulate real time sensors in our case a force/torque sensor to get the interaction at the contact point when the robot meets any environment. Also, Gazebo with **ODE** (Open Dynamics Engine [36]) also provides high realism out of the box. Nevertheless, it is important to find good physical parameters for the simulated objects. Examples of those parameters are the mass and coefficients of friction. Another important aspect of simulation-realism is that the simulation should not introduce new problems which do not occur in reality. Such a problem, which occurred in the development of the thesis, is that the simulation ran slower than the system time as the robot software used the system time. This may cause a time delay in real time teleoperation especially when the robot is in a constrained environment.
- **Different levels of simulation**: Sometimes, it is also useful to test with a more abstract and less realistic simulation. This allows testing the optimal case without sensor noise or without using low level components, such as localization, if those

produce new errors or are not finished yet. We use multi-level abstraction for sensing and for actuators. How we use multiple abstraction levels is shown in the next section about the architecture.

Building an experimental robotic setup can be a very difficult, prone to hardware faults and expensive process. A common way to overcome some of these problems is to model a part or entire system in simulation environment. Moreover, it comes handy to model hazardous or inaccessible sites using virtual environments. However, implementation of teleoperated robotic systems with force feedback manifests additional problems. The human robot interfaces should exist in hardware and this in turn requires simulated system to work in proximity with a real-time.

With the objective of our thesis described in chapter 3, we describe a successful implementation of such a system in Gazebo simulator with Robot Operating System. We simulate an experimental setup having a master side which is to be operated by a human controlling a slave side having a serial robot arm that is teleoperated to interact with a virtual object. Since the complete setup runs on Gazebo writing the software plays a major role in my thesis. This chapter presents to the reader the software implementation process that includes software requirements & architecture, technical requirements & architecture, structural architecture. Once such a setup is ready this can serve as a platform for experimental validation of robot interaction related problems.

4.2 Software implementation

4.2.1 Objective

The main objective of implementing software is to make our design usable, configurable and hardware-independent as much possible. Finally, a link between user and robot control software has to be made, meaning graphical and/or physical user interface acts a human robot interface that should be easy to understand and intuitive enough for the user to use in real time communication with robot.

4.2.2 Requirements

This section describes a number of requirements that the software package should fulfill. Identifying the requirements for software system is essential part of the development process

4.2.2.1 Functional requirements

The purpose of the software under development is to make it possible to use compliant control on a serial open-chain manipulator as in our case. Moreover, the human input for the robot should be easily understandable in the form of Graphical User Interface (GUI). The requirements that describe needed functionalities of the software are following:

General movement control- To control the robot in the first place, it needs jogging capabilities, meaning that the software should be able to control robot's EEF Cartesian space relative to known reference frame. That includes capabilities for forward and inverse kinematics solving.

Manual control abilities through some external device (e.g. keyboard, joystick-. This is needed for testing the robot, for cooperation capabilities and other applications, where manual position control of the robot's EEF is needed.

Processing force-torque data- Should be capable of receiving raw data from robot's low-level controller and/or from driver. As various manufacturers implement their robot's FT sensitivity differently, the filtering functionality should be configurable. For our case the sensor is a simulate one so filtering was not done.

Compliant, position/force control rules-The method for converting or generating jog commands (velocity commands) from FT data depending on different predefined control rules. In my case only position based force control is being implemented.

Graphical User Interface (GUI) - The user should be able vary parameters of the object being manipulated. To change the degree of compliance

4.2.2.2 Technical requirements

Technical requirements include the different prerequisites on ability of the software. The requirements are brought out as described below:

ROS support-. In addition to fulfilling general ROS package infrastructure standards [36], interconnection capability with other ROS packages is necessary. Thus, every added module must contain some interface for communicating with it through some available ROS communication pattern, such as topics, services and/or actions [37].

Performance specification- As the compliant position control happens in real time, the performance of the calculations is crucial. Although, the bottleneck of the performance might also be the controlling computer, the best effort should be made to minimize the use of computational resources, e.g. avoiding or limiting the use of thread sleep-functions, memory allocation and exponential complexity. For a robot manipulator to act as expected, the trajectory and forces processing calculations should be made around 50 Hz.

Hardware independent- Hardware independency is one of the most important aspects of software in robotics. The fact that single software can be used with many different manipulators advances is of significant important.

Software configurability- The software should be reconfigurable to facilitate hardware independent Different robot-specific variables can be changed externally modified without recompiling the code.

Safety requirements. Despite the fact that most co-robots have safety functionalities integrated in controller level, another layer of safety features in higher level of code helps to assure that the system is safe to use. Safety features include ability of setting maximum force and torque parameters on individual axes as well as maximum velocity that the EEF of the manipulator could reach. Additionally, revival strategies from erroneous situations need to be defined.

4.2.2.3 Requirements for the robot

Most of the industrial robotic arms are similar to each other in term of their design and usage. For example, consider serial manipulators – all this type of robots share common functionalities and are conceptually same. Therefore, it would be logical that some software module responsible for certain common functionality should work on different robots. It is impossible to achieve ideal hardware independence, although it is possible to define certain rules that a hardware system has to follow, so it would be compatible with the software. The software used in the current thesis can be applied on a robot that meets the following requirements:

1. it is a serial open kinematic robotic manipulator;

2. it has ROS support, meaning that there should be a compatible driver [38] for controlling the robot through well-known standardized ROS messages [39];

3. compatible with ROS MoveIt! [40];

4. A Force/torque sensor is used at the wrist and the driver of the robot should be able to publish wrench values. In our case force/torque sensor is defined as a joint inside the robot description to make the sensor publish the wrench values [41].

4.3 Technical architecture

In current subsection, various lower level architectural decisions, used in the development of the software, are stated and justified.

Robot Operating System (ROS). ROS is widespread, active, and matured framework for developing robot software systems. Considering ROS's advantages, such as various communication patterns between nodes, asynchronous nature and modularity, the decision of choosing ROS was straightforward. Also, enormous community and open source code base encouraged the choice even further. For current project, Kinetic distribution [42] of ROS is used since it is officially recommended release as of May 2018

.

C++ programming language. The stimulus for choosing C++ for fundamental programming language for the project derives from using ROS in the first place, as the framework as well as previous related projects is developed in C++. Furthermore, due to strict real time restrictions, C++ is optimal programming language performance wise. Finally, as C++ is object oriented language, it is intuitive to write and understand for software developer with moderate amount of experience.

Publish-subscribe pattern. Due to robotic systems need for asynchronous operations, the publish-subscribe pattern for exchanging messages between nodes is used in certain cases. The patterns principle is that a node can subscribe or publish messages on previously defined specific topics and the communication takes place asynchronously.

Client-server pattern. Information transfer with a node, which provides return value after performing an action that was initiated by second node, requires client-server pattern (services and actions in ROS). This type of interaction is synchronous.

4.4 Software architecture

Software architecture can be classified into two related, but different, concepts. Architectural structure consist of subsystems and descriptions how the subsystems will interact with each other, whereas technical architecture (or architectural style) specifies different lower level techniques and concepts that underlie a given system. Unfortunately, in many existing robot systems, it is difficult to specify clear boundaries between different approaches and subsystems, meaning that the architectural structure is blurred and different architectural styles are tied together [43]. Consequently, clean and well-conceived architecture of software can have significant influence on the quality of the code.

4.4.1 Software development process

4.4.1.1 Developing principles

It is important to clearly define the requirements of software development process. This helps to discipline the developer to design understandable, maintainable and scalable software and makes sure that every piece of programming could be useful in future and for wider community.

Make use of any related available packages- Before starting to code functionally, it is sensible to look into already written software. Often, there is similar functionality already implemented and it just needs some adaption or generalization. Since ROS is open community and all the code is open-source, it is easy to contribute by improving previously written code.

Maintainability- Newly written code should be maintainable, which means understandable for other users. It is possible to follow some rules of clean code [44], including naming conventions, formatting conventions etc.

Software design rules- Software should be designed modularly, so the modules are open for extension but closed for changes in functionality. This kind of approach ensures that the particular software is scalable.

Documentation- Instructions for how to interface the different modules and using the software is very important for making the software open-community friendly. The purpose is to develop software for larger community and thus help robotics to grow faster in the means of reusability.

4.4.1.2 Development tools

The appliances for software development used by the author of this thesis can be listed as follows:

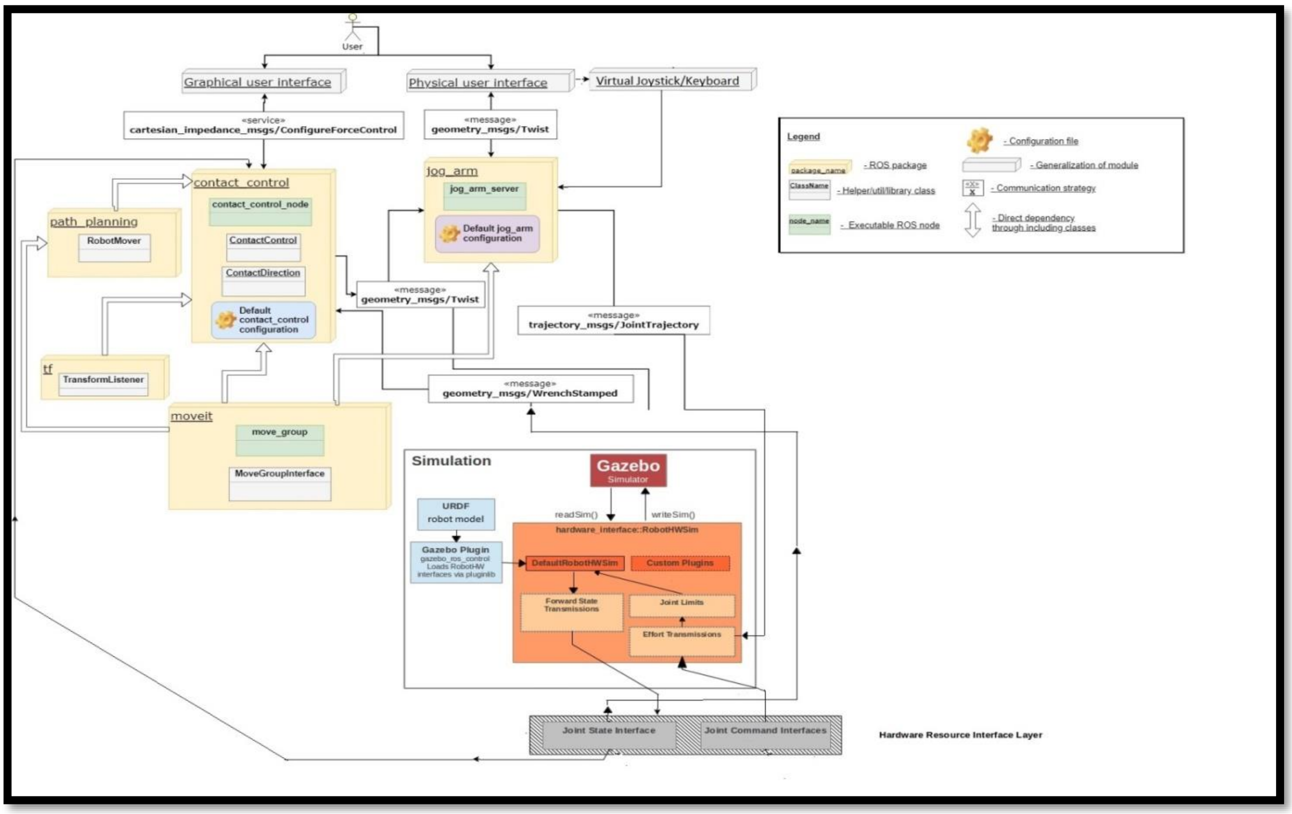
- Framework for developing robotic systems. As described in section (Technical architecture), the software was developed using Robot Operating System (ROS) framework and its Kinetic distribution. In addition to architectural element, ROS can also be used as a development tool. It provides core functionalities, such as package management [45], code compilation tools and infrastructure [46]. Moreover, communication between nodes is standardized and the data transfer is accessible, thus debugging and monitoring of the code at runtime is practically effortless. There are number of tools available that are built for ROS that can be beneficial in system development process, including tools, described in forthcoming points.
- Simulation environment Gazebo [47] was used during the project. Newly written code might be unstable or even dangerous for robots or surrounding environment, thus it is sensible to test the code in simulation environment before launching it on real robot manipulator. Gazebo is excellent fit for this kind of task being compliant with ROS and 3D URDF models are available of both robots used in this project.
- MoveIt! is a set of software packages integrated with ROS. Along other capabilities of MoveIt! there is analytical kinematics solver, which is one key component of current

thesis's software. Additionally, it provides move group interface which is used for sharing information and giving commands about robot's pose and state.

- Operating system (OS) that was used in this project is Linux distribution Ubuntu 16.04 [48]. The OS was used for both, running as well as developing the software for manipulators. The main reason for choosing Ubuntu was ROS's prerequisites, meaning that the Kinetic version of ROS will run only on corresponding OS. Ubuntu is also an ideal environment for writing and managing code, as it supports several of IDEs. Again, it is perfect for controlling robot manipulators, because it is possible to install real-time kernel for low latency communication.

4.5 Structural architecture

A modular approach has been taken when designing current system. In ROS, software is always organized into packages. A ROS package is an entity that contain library can contain library, dataset, configuration files, launch files, makelists and other logically useful set of elements. The aim is to assemble everything into a single package, to maintain scalability and reusability of the packages. Besides, the modularity of the developed software is successfully attained by reusing already available packages and the modular nature of ROS.



User
Graphical user interface
Physical user interface
Virtual Joystick/Keyboard
«service» cartesian_impedance_msgs/ConfigureForceControl
«message» geometry_msgs/Twist
contact_control
contact_control_node
ContactControl
ContactDirection
Default contact_control configuration
path_planning
RobotMover
jog_arm
jog_arm_server
Default jog_arm configuration
«message» geometry_msgs/Twist
«message» trajectory_msgs/JointTrajectory
tf
TransformListener
«message» geometry_msgs/WrenchStamped
moveit
move_group
MoveGroupInterface
Legend
Configuration file
package_name - ROS package
Generalization of module
ClassName - Helper/util/library class
Communication strategy
node_name - Executable ROS node
Direct dependency through including classes
Simulation
Gazebo Simulator
URDF robot model
readSim()
writeSim()
hardware_interface::RobotHWSim
Gazebo Plugin gazebo_ros_control Loads RobotHW interfaces via pluginlib
DefaultRobotHWSim
Custom Plugins
Forward State Transmissions
Joint Limits
Effort Transmissions
Joint State Interface
Joint Command Interfaces
Hardware Resource Interface Layer

Fig 4.5 **Structural architecture scheme of the software package**

The schematic of the structural architecture of the software bundle has been shown in Figure 4.5. It shows the interaction and dependencies between different modules involved in the system. In order to clarify the functionalities, main purpose of each component is now described. Contact control package holds contact control node, which processes force control and impedance configuration data from user and passes it on to *ContactControl* class. Contact control node acts as a middleware between the GUI and *ContactControl* class. It helps to change the format of the data & acts as a bridge between the two. Contact control package includes *ContactControl* class, which receives the force/torque (FT) data from robot's force/torque sensor using *ft_sensor* topic and sends velocity commands to *jog_arm* package which then to the controllers of manipulator. The *jog_arm* package includes *jog_arm_server* node that handles kinematic calculations, in order to move manipulator's EEF in Cartesian space and continuously publishes joint trajectory messages to robot's positional controller. The path_planning package is helper package that can be used for path planning of the robot. MoveIt! package is used as an interface between manipulator's driver and the rest of the functionality. Essentially, MoveIt! package provides other components with *MoveGroupInterface* that communicates over ROS topics, services and actions to the *move_group* node which then provides other modules with kinematic data, such as robots current pose. The TF package [49] is used for retrieving and computing transformation between robot individual frames, such as fixed base frame and EEF frame. It ensures the functionality even when different robot controllers expect velocity data in different coordinate frames. Final part of the scheme is robot driver package which is always robot-specific and contains the hardware driver, MoveIt! configuration and relevant configuration files that override default configuration.

4.5.1 Moving manipulators' EEF in Cartesian space

Controlling the manipulator's EEF in Cartesian space is not an easy task because it requires real-time kinematic calculations to convert between joint space and Cartesian space of the manipulator. Initially, joint level pose is received from manipulator joint's states using Joint_State Package; forward kinematic calculations are to be performed to get the POS of the EEF in Cartesian space. All the velocity calculations are then done on the EEF. In order to translate the new EEF velocity back to joint space velocity inverse kinematic calculations are needed.

However, there are manipulators that do have this kind of functionality already implemented at controller level, so next level of code just has to provide Cartesian space EEF velocities. Otherwise, joint-level velocities need to be calculated on a higher level. The *jog_arm package* [50] provides such higher level joint velocity calculation. It includes *jog_arm_server* node that subscribes to Cartesian velocity data and publishes joint space velocity data. The package uses MoveIt! for real-time kinematic calculations.
During current thesis, the *jog_arm* package was used for controlling UR5 manipulator inside the simulation environment.
`

4.5.2 User interfaces

In order to communicate with manipulators' controllers at runtime, a user interface makes things lot easier. The user interface can be divided into graphical (GUI) and physical (PUI) interfaces. During current thesis project, both approaches have been used and worked upon
PUI was mainly used for testing manipulator EEF movements in Cartesian space and manually controlling the manipulator joints & torques. Two options for moving the manipulator through PUI are kept – joystick and keyboard. Joystick interface functionality was already available in *jog_arm* package but for simulation we have used a virtual joystick interface. The virtual joystick was designed on QT platform and a part of *ros_qml* [51] package was used here for communication with other ROS nodes. Along with a virtual joystick, an interface through keyboard had to developed, thus keyboard publisher package [52] was developed. The *key_to_twist* node in this package translates key presses into velocity commands. The above mentioned node was also used inside the virtual joystick package.
GUI is designed to vary certain parameters and control rules to corresponding node at runtime. For this purpose, *manipulator_control_gui* package was designed. The GUI is developed using rqt, which is ROS a wrapper for QT toolkit [53]. The package was written to provide intuitive graphical interface for applying different control laws and vary certain parameters. It has been designed such that full customization for every parameter is possible. All the communication between GUI and *contact_c_node* is done using ROS service calls. The GUI is still in the developing phase and not fully functional.

The next chapter presents in details about the design and shows a framework of the developed design.

Chapter 5

Design & Development

The previous chapter describes the approach taken and ways to implement the proposed simulation to design a master slave teleoperation of a serial robot manipulator and control its interaction with a virtual object. The current chapter presents in details about design of the simulation components.

5.1 Simulation Environment:

With real-world applications in our mind and the fact that it is necessary to integrate program interfaces with a real hardware we chose the Robot Operating System (ROS) as underlying platform and used Gazebo for designing our simulation system. As we know ROS is not an operating system in true sense but rather a software framework that brings/facilitates the various hardware components, such as sensors and actuators, by means of very well defined structured communication layer above the hosting operating systems [48].

In the following list we will highlight some of the concepts used by ROS.

• Nodes are the run-time processes. Node can be a pure computation process, e.g. calculating inverse kinematics of a robot, or a driver for a sensor or actuator. Usual robot setups consist of many nodes.

• Messages are the data structures comprising typed fields. Nodes communicate with each other by means of messages. A message can consists both of basic data types (integer, floating point, boolean etc.), arrays and data structures themselves.

• Topics are named and strictly typed message buses. Any node can publish/subscribe to a topic and send/receive messages as long as they are of a right data type.

• Packages are the main unit for organizing software in ROS on the level of file system. A package may contain nodes, plugins (shared libraries), datasets, configuration files, or anything else that is usefully organized together.

• Stacks are collections of packages that provide an aggregated functionality, for instance, navigation or image processing algorithms and routines.

5.2 Components of the Simulation

Our implementation of the system (figure 5.2) in Gazebo and ROS consists of the following components:

• *Contact force node*: provides interface to display the contact force generated during interaction.

• *Teleoperation node*: provides the mapping of the local coordinate frame of a virtual joystick to a global frame of the virtual environment. This node is in charge of the motion scaling and indexing as well as switching between the modes of teleoperation, e.g. 'Direct Force Control' & 'Position Control') of the virtual slave manipulator.

• *Virtual Slave node*: presents the slave manipulator as a single slave to the master site. It provides control decomposition routines.

• *GazeboRosControllerManager plugin*: provides CM interface for JT Cartesian Controller in Gazebo.

• *JT Cartesian Controller*: calculates inverse dynamics of the robot

• *FT Sensor plugin*: provides ROS interface for forces and torques calculated by physics engine in simulation.

• *Camera plugin*: simulates the cameras located in a virtual environment and provides ROS interface for image visualization tools.

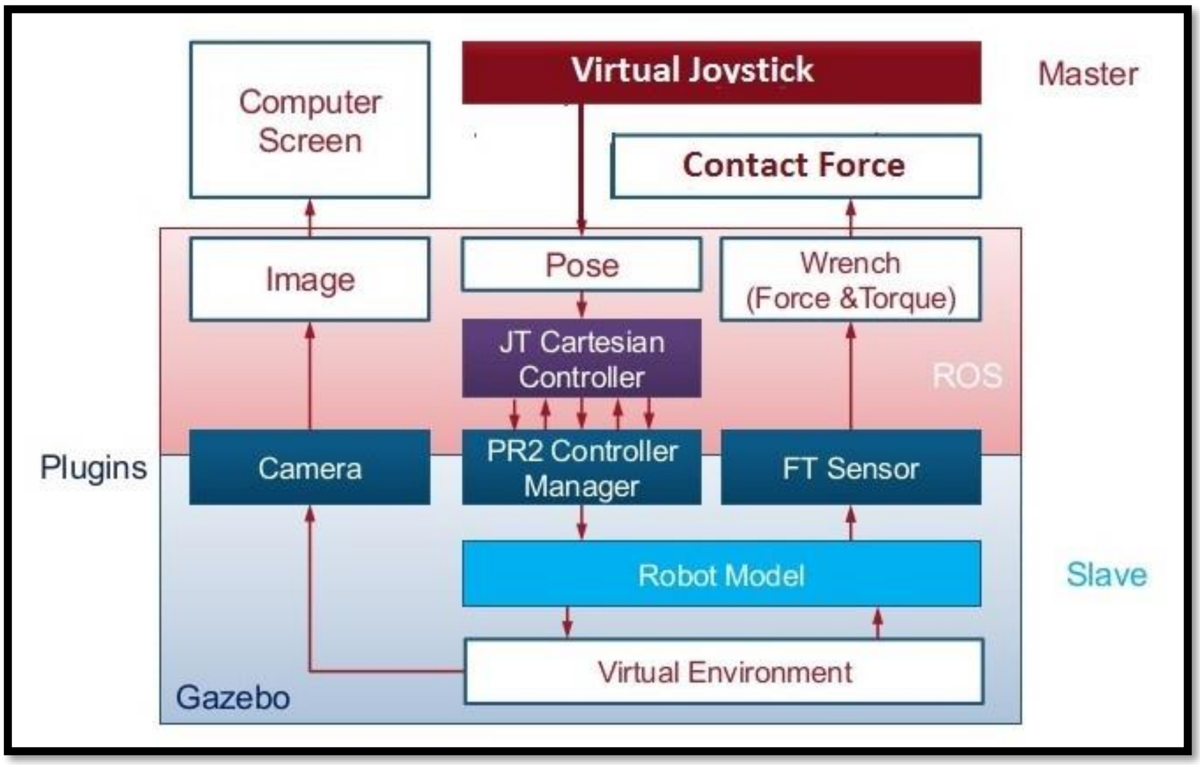


Figure 5.2 **Intercommunication of numerous components of the system is outlined**

Brief description of all the components:

Contact force node: The role of contact force node is to accept force and torque data from the FT sensor at the wrist of the robot and determine the appropriate velocity commands that reduce undesired contact forces on the robot and the manipulated object. This is based on the GCCF

Teleoperation node: provides the mapping of the local coordinate frame of a virtual joystick to a global frame of the virtual environment. This node is in charge of the motion scaling and indexing as well as switching between the modes of teleoperation, e.g. ‘Direct Force Control’ & ‘Position Control’) of the virtual slave manipulator.

Virtual Slave node- loads the robot URDF file into the ROS server and spawns the respective joint controllers using the defined ROS transmissions to actuate the arm . Also it loads the trajectory controller for the manipulator which is used to take position & orientation values from the user end.

FT Sensor plugin: A simulated force/torque sensor is used at the wrist of the robot to measure the contact forces at the point of interaction. The plugin is written in c++ &provides ROS interface for forces and torques calculated by physics engine in simulation. This plugin can get the force/torque data, manipulate it, and output it to destination ros topic in our case sends it to the contact_force node.

5.3 System Architecture:

Robotic systems are often based on an ROS (Robot Operating System), which is widely accepted
as the standard framework for robotic system development. ROS helps to develop a robot easily by adding many ROS packages to a software stack. ROS itself is one example for a modular system, as different ROS packages can be combined in a modular manner.

Our simulated system consists of two main components broadly. One is a master devise working as Node in ROS environment and a virtual slave robot working as a different node both are connected to ROS server & can communicate over topics or can interact with each other using ROS actions/services. Master devices are represented by a virtual joystick and GUI for communication with the slave manipulator. The virtual joystick is used to give commands to the slave robot in terms of position trajectory whereas the GUI is used for varying the environmental (virtual object's) parameters like stiffness or damning used during the interaction control.

As a result of a high popularity of ROS in the robotics research community and due to its modular nature most of the components that are necessary to implement in our system are available in the repository. For instance, one of the key components in our simulation – a model of UR5 manipulator is available from *universal_robots* stack under ROS-Industrial program and was adopted with some modifications. Our system requires measuring the contact forces when the robot EEF interacts with an environment. In order to do that a force/torque sensor is used at the wrist of the robot. Gazebo permits to write custom plugins in order to incorporate a force/torque sensor working in real time. A detail description about modeling the sensor has been discussed later on. ROS understands a robot model specified only in URDF which is written in XML language. To use a force/torque sensor it has to be included inside the main robot URDF file also and defined as child link to the last link of our robot which is wrist_3. This is the conventional process of placing a sensor with the wrist of a manipulator. The F/T sensor is purposely defined as joint in the URDF file to facilitate ROS inbuilt *joint_state_publisher* package to publish the wrench values at the point of contact.

However, Gazebo model itself is just a set of rigid links connected by joints. In order to actually use them as actively controlled actuators an appropriate controller has to be designed Here, it is very easy to see the demarcating line between ‘bare hardware’ (Gazebo) and controlling software (ROS). The model of manipulator presents a passive articulated object with a set of dynamic properties such as inertia, mass etc., which is unable to move by itself. In general a real manipulator would have the same properties and therefore control algorithms developed on the base of simulation can be transferred to the real hardware without major changes.

To simplify development of the robot controllers ROS provides a special interface by means of PR2 Controller Manager (CM). Despite its name CM can be used with any robot that is possible to define with Unified Robot Description Format (URDF).PR2 Controller Manager is a vital component of the system. It builds abstraction layer over hardware actuators whether it is a real actuator or a simulated one. While the pr2 controller manager package provides the infrastructure to run controllers in a hard real time with real robots the GazeboRosControllerManager provides the same interface for controllers in our simulation. The Gazebo ROS Control web page should be referred for the more details.

The actual work flow for our system has been shown in the previous chapter under structural architecture column.

5.4 Design of Master and Slave nodes in Gazebo

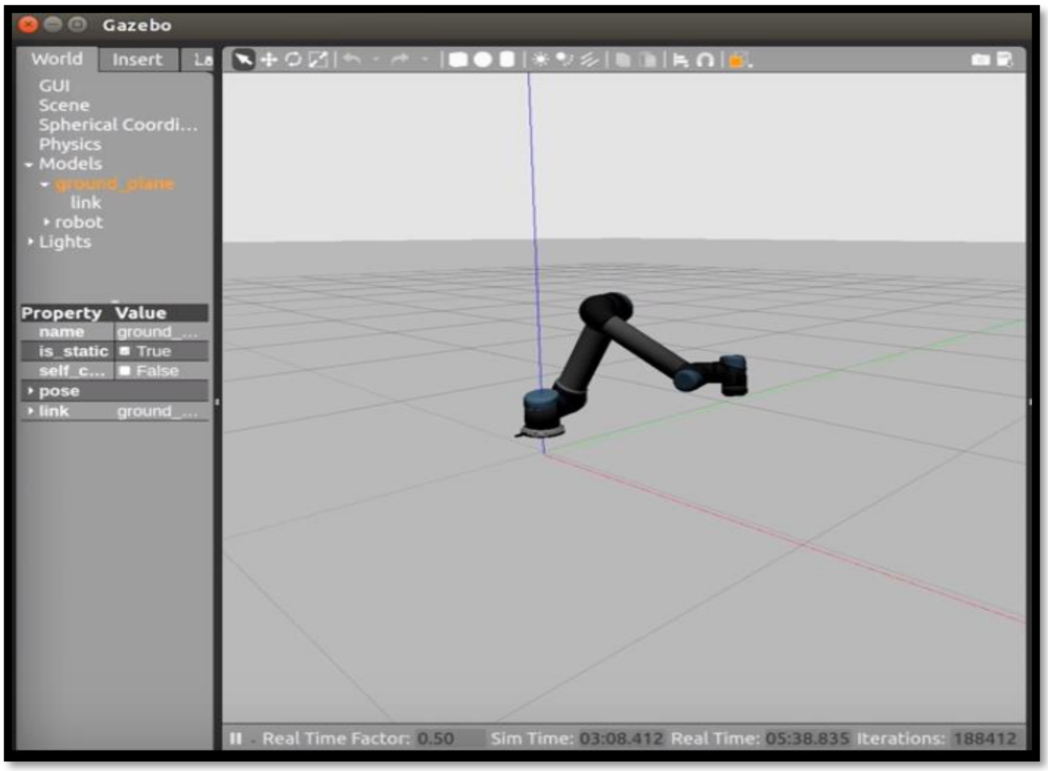


Figure 5.4 **Screenshot of Slave Manipulator with the controllers loaded inside the Gazebo world**

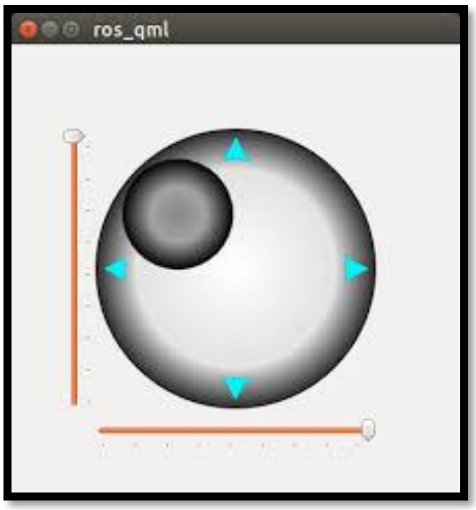
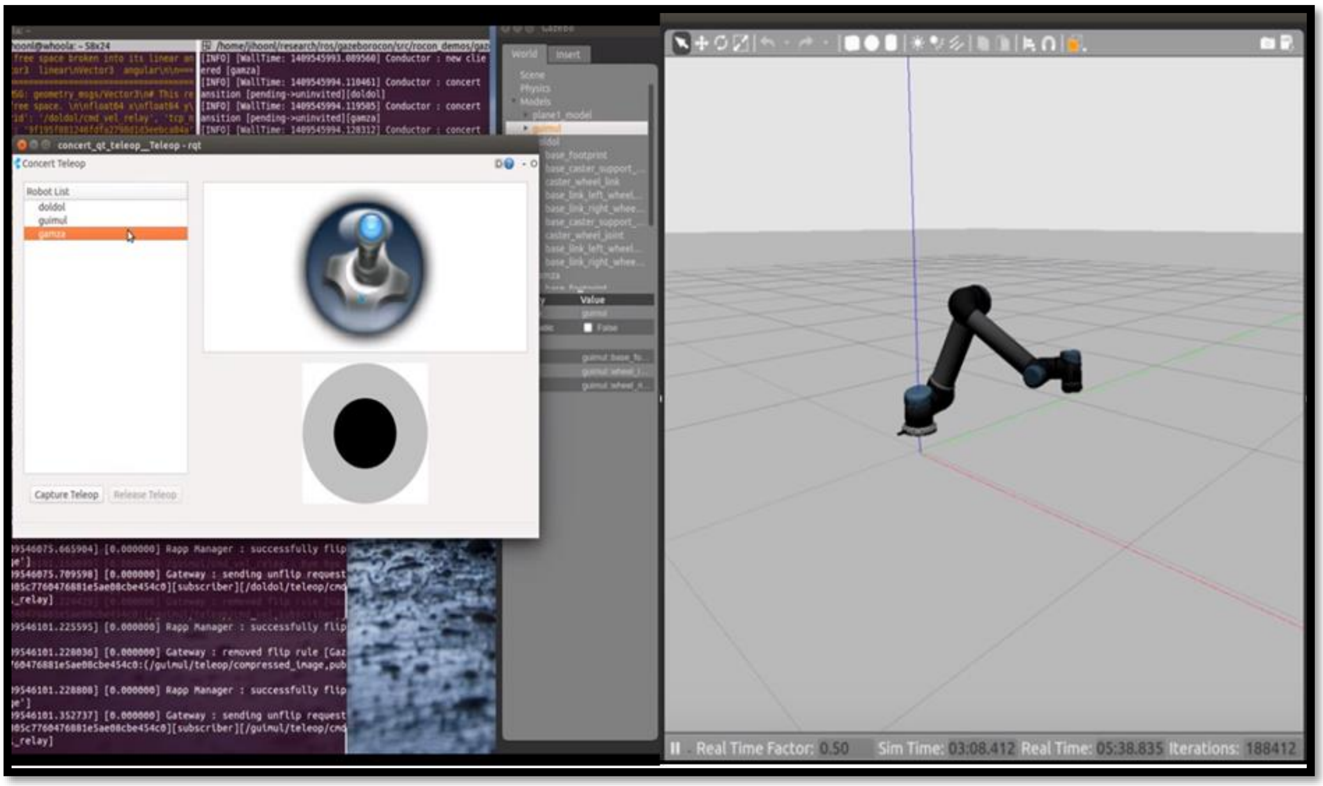


Figure 5.5 **Shows Virtual Joystick user interface**



```
hoonl@whoola: ~ 58x24
free space broken into its linear an
or3  linear\nVector3  angular\n\n===
========================================
G: geometry_msgs/Vector3\n# This re
free space. \n\nfloat64 x\nfloat64 y\
id': '/doldol/cmd_vel_relay', 'tcp n
 : '9f195f881246fdfa2790d1d3eebca84a'
```

```
/home/jihoonl/research/ros/gazeborocon/src/rocon_demos/gaz
[INFO] [WallTime: 1409545993.089560] Conductor : new clie
ered [gamza]
[INFO] [WallTime: 1409545994.110461] Conductor : concert
ansition [pending->uninvited][doldol]
[INFO] [WallTime: 1409545994.119505] Conductor : concert
ansition [pending->uninvited][gamza]
[INFO] [WallTime: 1409545994.128312] Conductor : concert
```

```
09546075.665904] [0.000000] Rapp Manager : successfully flip
e']
09546075.709598] [0.000000] Gateway : sending unflip request
05c7760476881e5ae08cbe454c0][subscriber][/doldol/teleop/cmd
_relay]

09546101.225595] [0.000000] Rapp Manager : successfully flip

09546101.228036] [0.000000] Gateway : removed flip rule [Gaz
7604768881e5ae08cbe454c0:(/guimul/teleop/compressed_image,pub

09546101.228808] [0.000000] Rapp Manager : successfully flip
e']
09546101.352737] [0.000000] Gateway : sending unflip request
05c7760476881e5ae08cbe454c0][subscriber][/guimul/teleop/cmd
_relay]
```

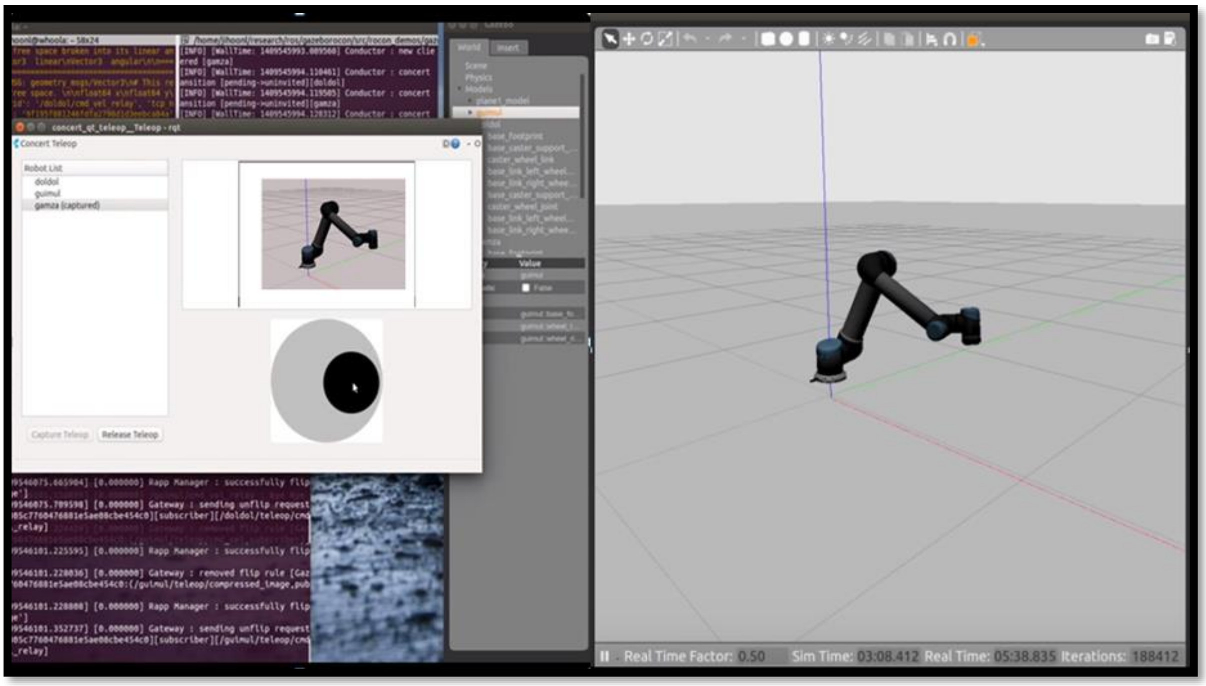
Figure 5.6a **Master Node** Figure 5.6b **Slave Node**

5.4.1 Master Node: It is the user side view of the system. Thanks to parallel processing architecture of ROS the two nodes communicate over ROS topics using internet connection in real time which acts as the communication medium in this case. To make things easier a virtual joystick interface is made based on the ros_qml package that publish twist messages to control the robot end effector tip in Cartesian space. A separate set of program is written in C++ that takes into account of all the kinematic calculations to map the Cartesian space position & orientation into joint angles. It uses the inverse kinematics and differential kinematics as the end effector velocity is to controlled based on the individual joint velocities. Next it uses the MoveIt planning tool to move the robot end effector in real time. Using MoveIt facilitates all the inverse kinematics calculations due to its inbuilt topologies for handling those calculations. Also a separate user interface in form of GUI has been designed to vary the stiffness/damping of the virtual simulated environment in real time to vary the responses.

5.4.2 Virtual joystick Node: Figure shows the virtual joystick interface. It has been designed using ros_qml package.The purpose of this interface is to publish twist messages to maneuver the end effector in the Cartesian space. For our simplicity the orientation of the last link is kept constant is kept unchanged to omit the torque measurements from our design. The sliders control scaling factors geometry_msgs/Twist message is published while the thumb position changes. Once the mouse button released the thumb goes back to center. It is based on the QT platform
The ros_qml package provides Qt 5.4 QML plugin for integration with ROS. The plugin is written in Python and thus depends on PyQt 5.4. (A minimum version of Python 2.7 is required for compiling).

5.4.3 Keyboard teleop node: Figure 5.10 shows the screen shot of keyboard teleop node. The teleop node is an open source code ROS package that takes input as keystroke from the keyboard and output the equivalent position & orientation of the end effector tip in the Cartesian space. It outputs the desired velocities to move the end effector accordingly. Also there is provision kept to change the control modes between joint space and Cartesian coordinate space.

5.4.4 Slave Node: It is the working side of our system that has the simulated robot manipulator with a fixed base and the virtually simulated environment. The manipulator selected is UR5 from Universal Robotics. Universal_robotics stack under ros-industrial provides all the necessary tools for interfacing the robot in the Gazebo environment. It contains packages that provide nodes for communication with Universal's industrial robot controllers, URDF models for various robot arms and the associated MoveIt packages. In order to use the above package to our use we have made certain modifications in the robot URDF file. Our design has a simulated force/torque sensor affixed with the wrist of the robot in our case it is the 6th link for the UR5 manipulator. Hence, the model for the sensor is added in the URDF file to make the FT link acting a child to the last link of the robot. Also the gazebo plugin for the force/torque is initialized for the sensor to actually work. The FT sensor is purposely defined in form of a joint to make use of ROS inbuilt capabilities to publish the force/torque data which is later subscribed by the contact_force node for viewing the interaction forces. The sensor modeling part is discussed in this chapter later on. The slave node also has the virtual object that is defined as a straight line fixed to the default plane in the gazebo world. It represents a mass less spring damper system that acts as a complaint surface for robot manipulation.



Master Node **Slave Node**

Figure 5.7 **Slave side view from Master Unit/Node**

For a master slave teleoperation maintaining some degree of telepresence of the slave unit to the master side is highly demanded feature. Since bilateral force feedback is absent which could have improved the telepresence actively visual telepresence is provided to the user in the master side that so that the user can guide the tool tip independently from the master node. A camera is used in the gazebo world that streams the real time image when the system is running Figure show the screenshot of the above mentioned idea.

In order to start communication with the UR5 in ROS, arm’s controller needed to be initialized in ROS. To do so, *roslaunch ur_5_gazebo.launch*. It immediately creates a node that starts to publish information about robot’s state. UR driver also subscribes to the *follow_joint_trajectory* topic to receive command from another ROS node. RQT graph for robotic arm is shown in Figure 5.8

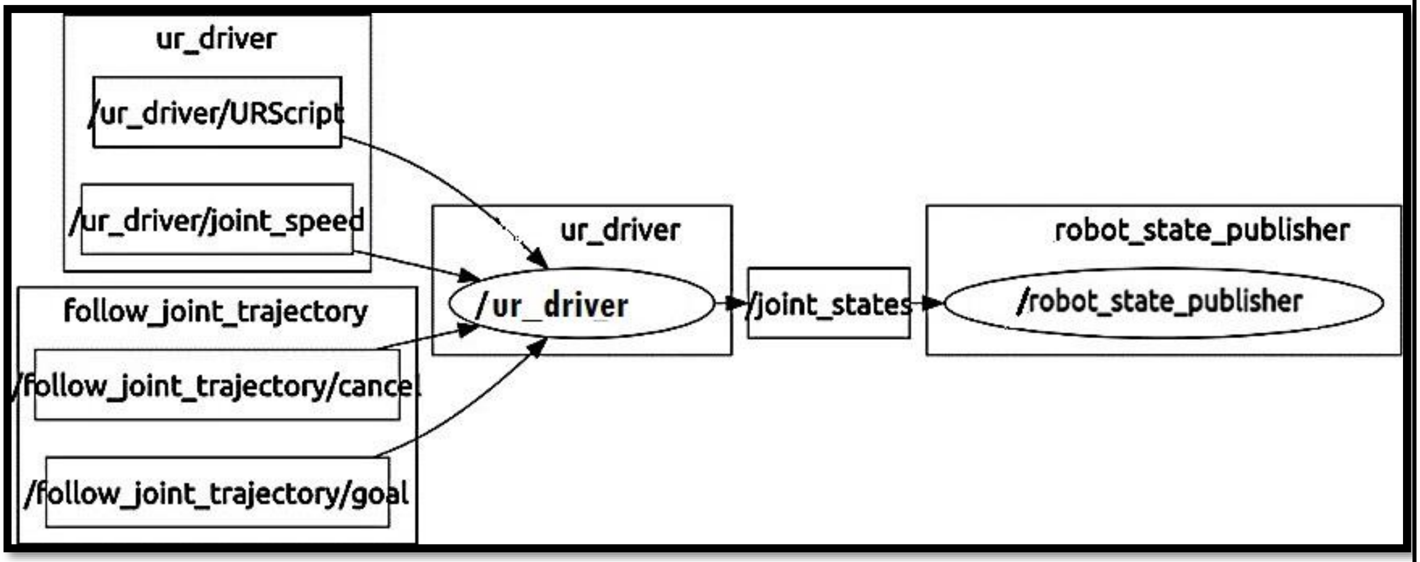


Figure 5.8 **UR5 in ROS computational graph**

ROS UR5 library was modified to remotely move the robotic arm into desired position. This program takes input as joint angles that the robot has to follow. The robot's end effector was manually moved in the Cartesian space flowing inverse kinematics using the Fast IK solver package with the virtual joystick. In order to move the manipulator in joint space another .The robot can be also controlled via another node apart from the virtual joystick using *keyboard_teleop* (figure 5.10) which has two modes either the motion could be in the joint space or with respect to the Cartesian space. Incorporating the IK solver package with the following teleoperater nodes that takes into account of all the transformations motion of the end effector in the world space is achieved. To do that the file mentioned publishes the value of joint angles into *follow_joint_trajectory* topic that UR driver listens to. Graph for this system is presented on Figure 5.9 when the robot changes its state its gets reflected in the topic *robot_state_publisher*.

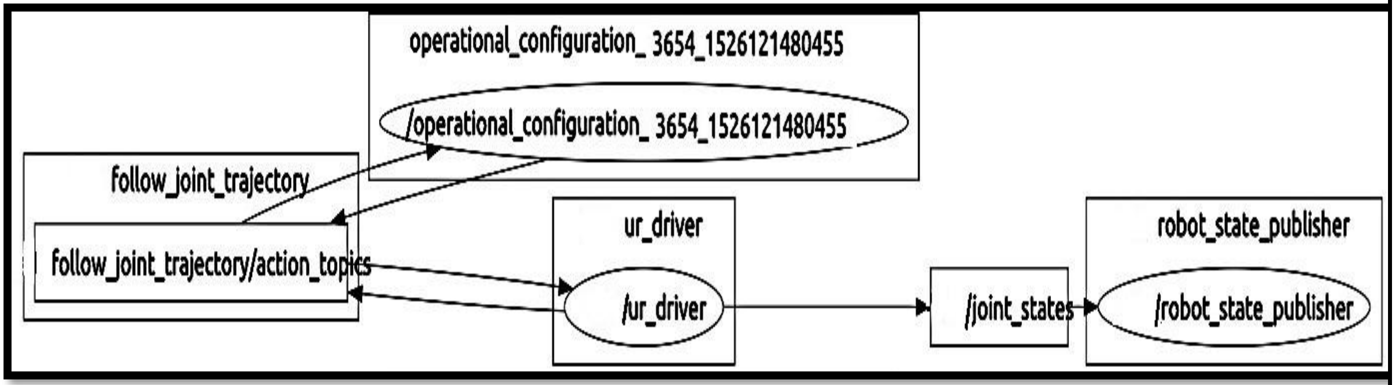
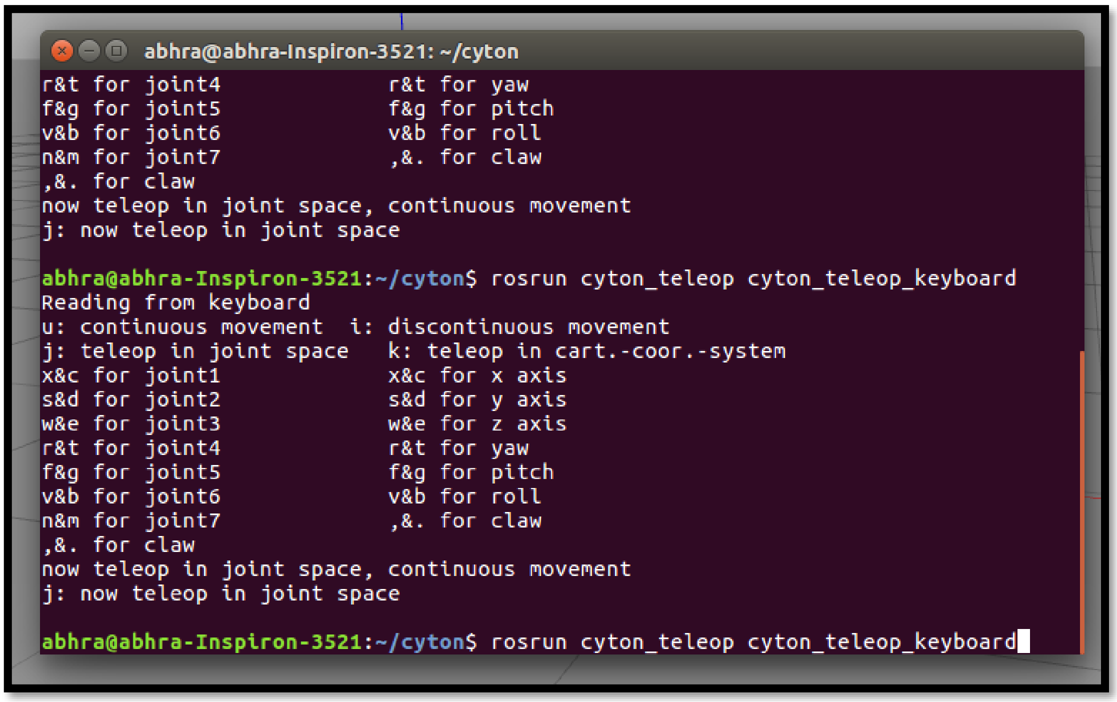

operational_configuration_3654_1526121480455
/operational_configuration_3654_1526121480455
follow_joint_trajectory
follow_joint_trajectory/action_topics
ur_driver
/ur_driver
/joint_states
robot_state_publisher
/robot_state_publisher

Figure 5.9 **Computational graph for the arm following position received from master side**



```
r&t for joint4              r&t for yaw
f&g for joint5              f&g for pitch
v&b for joint6              v&b for roll
n&m for joint7              ,&. for claw
,&. for claw
now teleop in joint space, continuous movement
j: now teleop in joint space

abhra@abhra-Inspiron-3521:~/cyton$ rosrun cyton_teleop cyton_teleop_keyboard
Reading from keyboard
u: continuous movement  i: discontinuous movement
j: teleop in joint space   k: teleop in cart.-coor.-system
x&c for joint1              x&c for x axis
s&d for joint2              s&d for y axis
w&e for joint3              w&e for z axis
r&t for joint4              r&t for yaw
f&g for joint5              f&g for pitch
v&b for joint6              v&b for roll
n&m for joint7              ,&. for claw
,&. for claw
now teleop in joint space, continuous movement
j: now teleop in joint space

abhra@abhra-Inspiron-3521:~/cyton$ rosrun cyton_teleop cyton_teleop_keyboard
```

Figure 5.10 **Keyboard teleop Node**

5.5 Design of force sensor in Gazebo

Force feedback implies using devices with the robot arm generally at the robot wrist placed in between the wrist & tool tip that are able to detect the forces at the point of contact. In our case the entire setup is simulated in Gazebo hence a 6 axis Force/torque sensor has to be also designed inside the simulation environment. Gazebo facilitates the simulation of a force/torque sensor by writing a plugin and incorporating that into the environment.

Understanding the Force/Torque Sensor

Generic properties

5.5.1 Force/Torque specific parameters

A force/torque sensor is created by adding <sensor> tag with the attribute type set to force torque. There are two additional parameters that can be set.

```
<sensor name="force_sensor" type="force_torque">
  <force_torque>
    <frame>child</frame>
    <measure_direction>child_to_parent</measure_direction>
  </force_torque>
</sensor>
```

<frame>

The value of this element may be one of: child, parent, or sensor. It is the frame in which the forces and torques should be expressed. The values parent and child refer to the parent or child links of the joint. The value sensor means the measurement is rotated by the rotation component of the <pose> of this sensor. The translation component of the pose has no effect on the measurement.

Regardless of this setting, the torque component is always expressed about the origin of the joint frame.

<measure_direction>

This is the direction of the measurement.

Adding a force/torque sensor to a link

While the SDF schema allows a <sensor> tag to be placed on either a link or a joint, the force/torque sensor only works on joints.

Modeling a Real Force/Torque Sensor

The above example places a force/torque sensor on a revolute joint. However, real force/torque sensors are typically rigidly mounted to another rigid body. A real sensor could not measure the force and torque exactly at the revolute joint origin. Modeling this way is reasonable if the real sensor is close enough to the joint that the error from the offset is negligible.

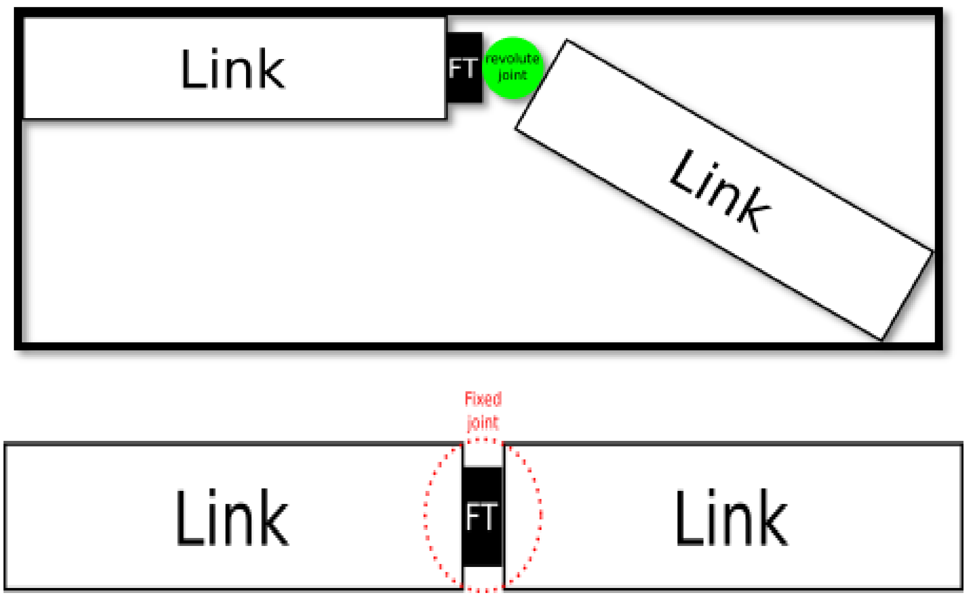


Figure 5.11 **Connection of sensor to the link of robot**

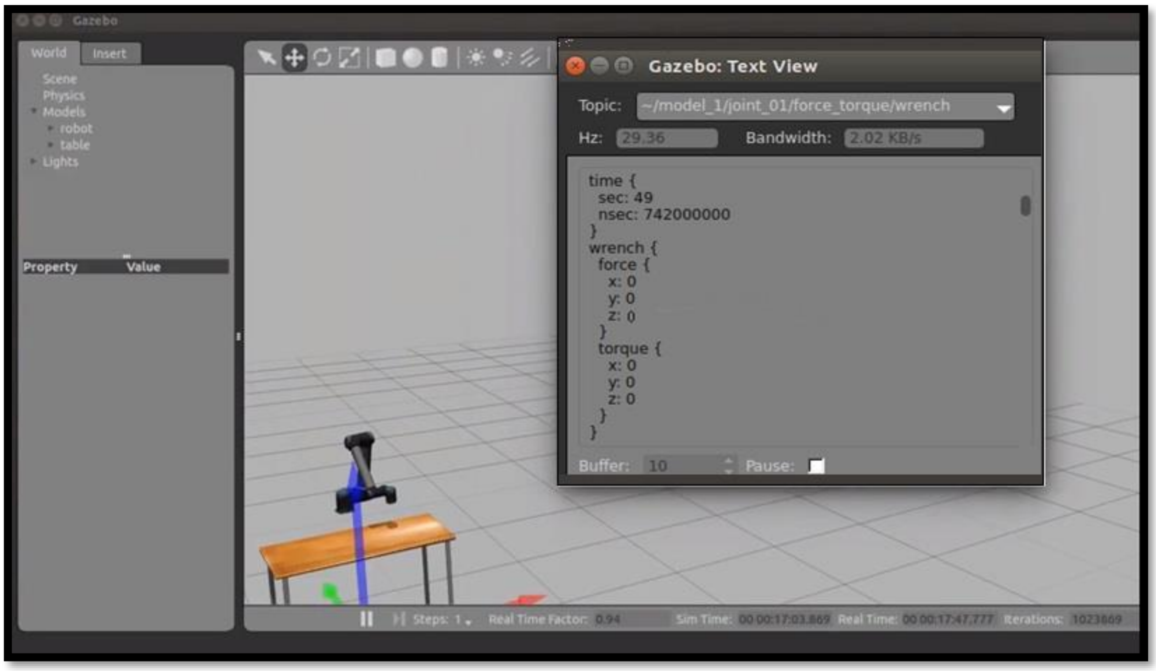
If this error is not negligible, the rigid body can be split into two links with a fixed joint at the location of the real sensor.

5.5.2 Preparing sensor for work

The simulated Force and Torque sensor is able to detect forces in three directions, as well as torques along corresponding axes. Maximam force that sensor can detect is 100 N, therefore, the maximum force that the arm can ensure at the end effector has to be set at 100 N. This means that when this force is exceeded, robotic arm will stop executing the program and a message will be displayed on the terminal screen saying that force limits have been exceeded. This is done as a safety feature in the simulation. If a real arm was being used instead it would go into STOP condition. This can be also included as a security system that helps to protect both the robot & humans around it.

Before setting up the force feedback system certain design aspects have to be taken into account to make work easy and realistic as possible.
First, it is obvious that the sensor to consider mass properties of the payload (in this case – the last link). For our case we have considered the mass to be negligible inside the simulation.
Second, the sensor needs to automatically take gravity into account. This is very important feature, because it completely removes need to adapt program code to the gravity, which can include complex calculations indeed. If mass properties are considered it has to be compensated in the control loop.

To view that the simulated sensor is actually publishing the force/torque data, arm's end effector is moved in the 3 orthogonal directions X, Y, Z sensor axis are pointing downwards in the Cartesian space Figure 5.12



Topic: ~/model_1/joint_01/force_torque/wrench
Hz: 29.36 Bandwidth: 2.02 KB/s

```
time {
  sec: 49
  nsec: 742000000
}
wrench {
  force {
    x: 0
    y: 0
    z: 0
  }
  torque {
    x: 0
    y: 0
    z: 0
  }
}
```

Buffer: 10 Pause:

Figure 5.12 **Screenshot showing force/torque value in free space (contact force/moments= 0)**

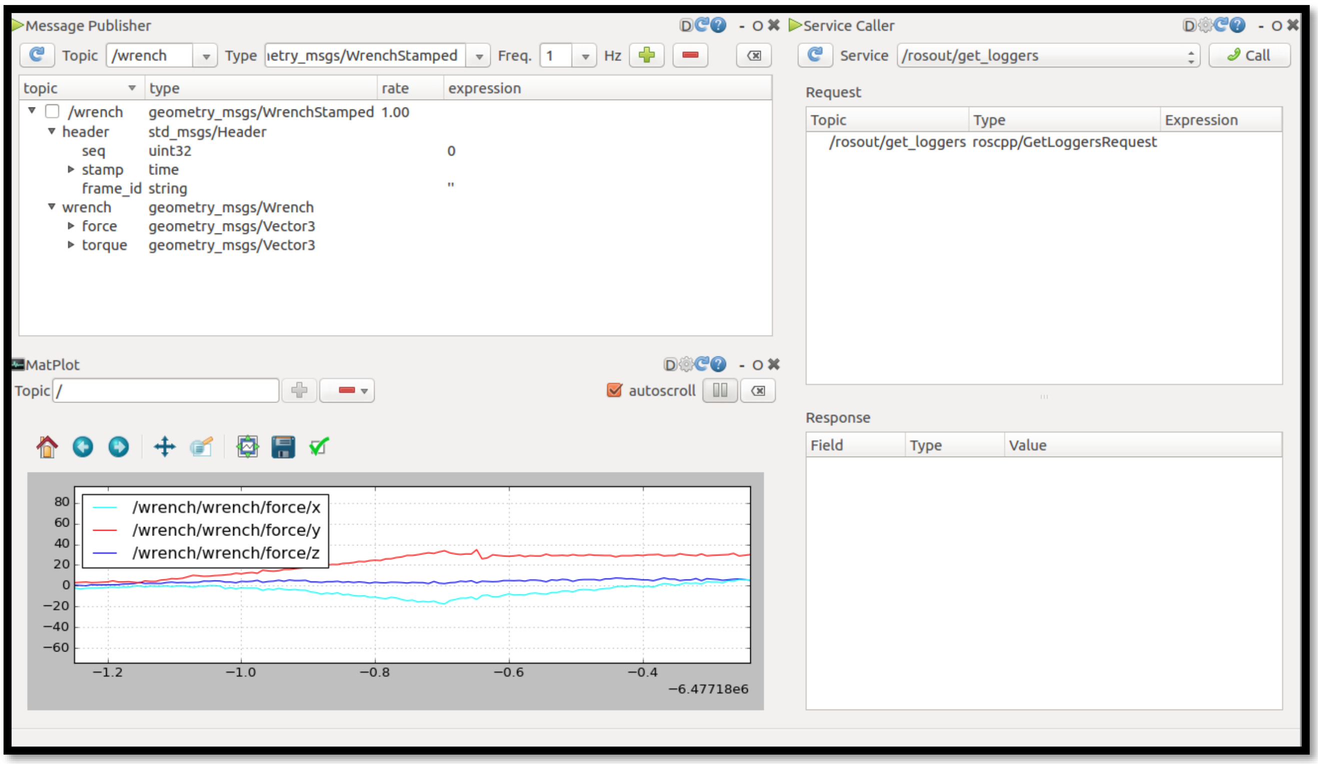
Initially, force readings in 3 axes were equal to zero, because sensor readings were made to take gravity into account. After applying external forces, readings started change accordingly. The external forces & moments in our case are applied manually using ROS in built *rqt* GUI tool. Figure 5.13 shows a screenshot of rqt application with three plugins loaded, message publisher plugin, plot plugin and service caller.

5.6 GUI based on ROS *rqt*

rqt is a Qt-based framework for GUI development for ROS. It consists of parts/metapackages .

rqt metapackage provides a widget rqt_gui that enables multiple `rqt` widgets to be docked in a single window.

rqt is a software framework of ROS that implements the various GUI tools in the form of plugins. One can run all the existing GUI tools as dockable windows within rqt! The tools can still run in a traditional standalone method, but rqt makes it easier to manage all the various windows on the screen at one moment.


Message Publisher

Topic /wrench Type ıetry_msgs/WrenchStamped Freq. 1 Hz

topic	type	rate	expression
/wrench	geometry_msgs/WrenchStamped	1.00	
header	std_msgs/Header		
seq	uint32		0
stamp	time		
frame_id	string		''
wrench	geometry_msgs/Wrench		
force	geometry_msgs/Vector3		
torque	geometry_msgs/Vector3		

Service Caller

Service /rosout/get_loggers Call

Request

Topic	Type	Expression
/rosout/get_loggers	roscpp/GetLoggersRequest	

Response

Field	Type	Value

MatPlot

Topic /

autoscroll

Figure 5.13 **rqt gui showing change in force values on manually publishing the force values**

Force and torque sensor works as a plugin that publishes readings for listener on the main station. When the robot URDF is loaded into the world the sensor starts to publish its values upon when made in contact. For our simplicity all the files are written in a single launch file as per ROS build tools. So launching a single file will start all the respective nodes accordingly.

Sensor publishes data into */robotiq_force_torque_sensor* node that */contact force* node, which reads from and process it into positional values (Orientation is kept unchanged) to maintain the required force at the contact of the robotic arm, the motion is kept along the surface. Hence the contact force is generated along the Z axis which is the normal direction in this case. Only force readings are considered during this work, therefore, *Mx*, *My* and *Mz* torques are ignored, as no change in orientation is considered.

5.7 Simulation of the virtual object for Interaction

Massless Spring-Damper Object Environment

We develop a virtual object being manipulated by the manipulator by applying forces on the end effector. These forces are generated by the difference between the length of the object vs the initial length of the object when the simulation started:

$$F_i = \frac{v_{0,I}}{||v_{0,i}||} \left[K_e(X_{e,i}(t) - X_o) + C_e(X_{e,i}(t) - X_{e,i}(t-1))\right] \qquad (5.1)$$

where F_i is the force on link i, K_e the stiffness of the environment/object, C_e the damping of the environment/object, $v_{o,i}$ the vector representing the object and its norm, $||X_{e,i}(t)||$ the Cartesian position of link i at time t, $X_{e,i}(t-1)$ the Cartesian position of link i at time t-1, and xo the initial position of link i at the start of the simulation. A rosnode is created that listens to the end effector positions, calculates the applied force (5.1), and applies the force on the two manipulators.

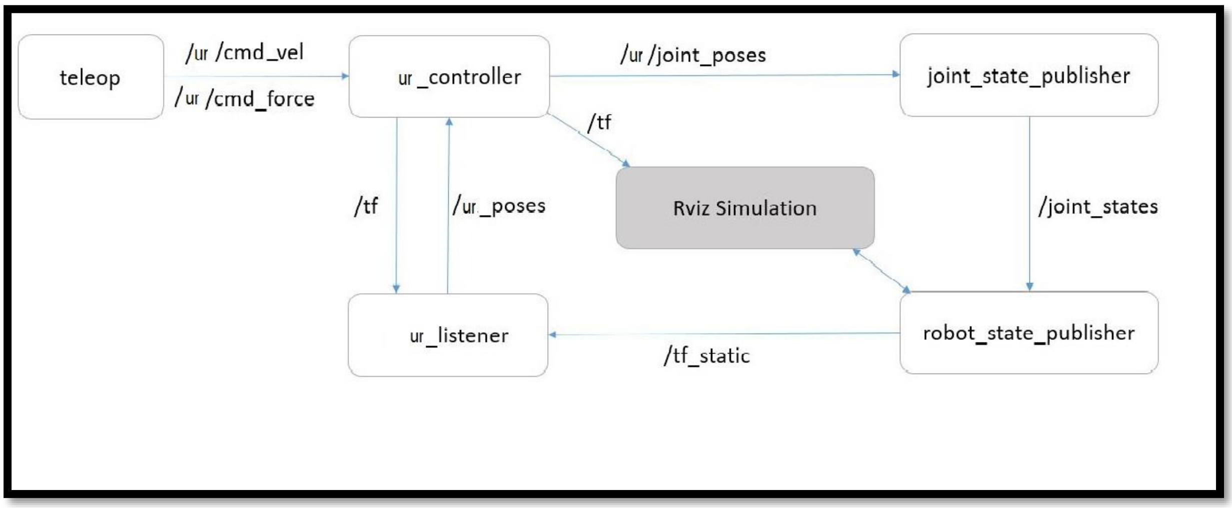


Figure 5.14 **ROS controller architecture with RViz simulation. Squares represent rosnodes and arrows represent rostopics for communication.**

5.8 ROS Controller Architecture

The controller was developed to work for both the RViz and Gazebo simulations. The ROS architecture of this controller interaction with RViz (Figure 5.14) and Gazebo (Figure 5.15) is shown below. The ROS software environment consists of rosnodes depicted by squares. Each rosnode is an individual process that performs computation where each node communicates with another node using ROS TCP/IP like communication called rostopics (arrows). This type of architecture reduces code complexity by reducing exposed API to other nodes. Node ur_controller represents the main controller which receives the Cartesian positions of the robot from the topic /ur_poses, calculates the desired Cartesian positions, computes inverse kinematics, and outputs the desired joint angles with the topic /joint poses

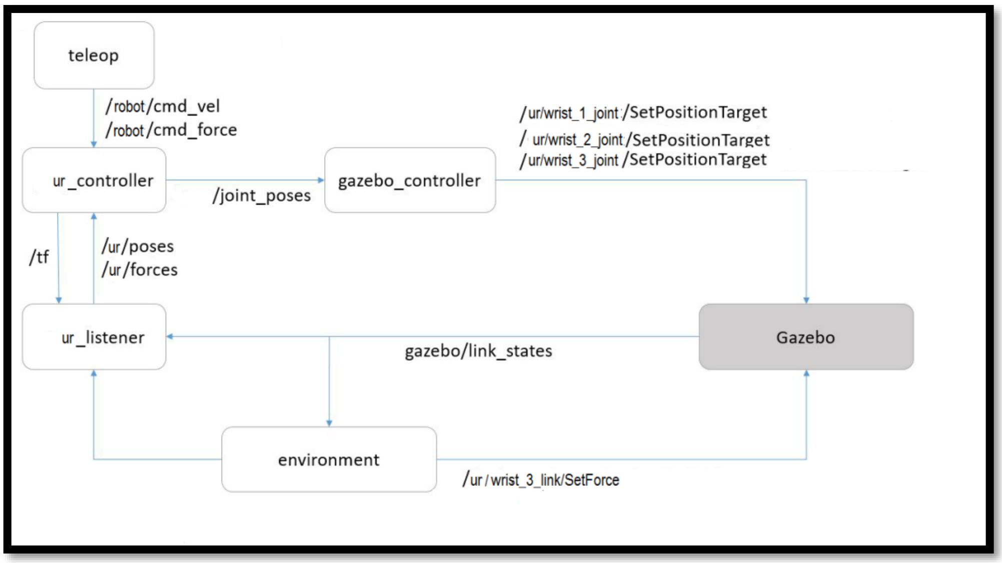


Figure 5.15 **ROS controller architecture with Gazebo simulation.**

.

The teleop node is an open source code ROS package [ ] that takes keystroke data and in this work, is modified to output x,y, or z desired Cartesian increments and x, y, z Euler angles increments with topic /robot/cmd vel. It also outputs an increment set force with the topic /robot/cmd force. A data collection node called ur listener receives link position and rotation data from the RViz or Gazebo simulation and publishes the data to ur controller. In RViz, it reads link positions and rotations using the tf library. In Gazebo, the topic gazebo/link states publish link positions and rotations.

Other supplementary nodes that launch with the RViz simulation are the *joint state publisher* which is a GUI to change the robot joint angles via joint states. Here this node is used as it was programmed to receive joint commands from other nodes. *Robot state publisher* is a node that contains the robot kinematic data from the URDF, receives desired

joint angle data from joint state publisher and publishes the correct position and rotation of each link of the robot to the RViz simulation.
is a node that contains the robot kinematic data from the URDF, receives desired joint angle data from joint state publisher and publishes the correct position and rotation of each link of the robot to the RViz simulation.
Other nodes developed to communicate with Gazebo are the *gazebo controller* node. This node is just complementary converting the rostopic /joint poses to rostopics that gazebo reads (top right of Figure 6.5).

The environment node is the virtual object for Gazebo that simulates an elastic massless spring-damper that reads link Cartesian positions via gazebo/link states

At the end of this chapter it should be clear to the reader about the various components of our simulation and how each one interacts with other in real time to perform the act of teleoperation of a slave manipulator for interaction with a virtually simulated environment.

Chapter 6

Testing &Results

6.1 Initial testing

In order to successfully test the software and its working a rough testing plan was prepared. The plan includes some test cases following by. The test plan is designed to cover all needed functionality while iteratively getting closer to real use scenario, where real robot is using data from the wrist sensor and uses a local force control at the slave side
A number of different *rqt* plugins (Figure 5.13 chapter 5) [58] were used during testing:

- ✓ **Plot** – for plotting and visualizing sensor and velocity data on graph;
- ✓ **Message publisher** – for manually publishing various of data for testing purposes (e.g. FT data and EEF velocities);
- ✓ **Topic monitor** – for monitoring published messages in real time;
- ✓ **Service caller** – for manually communicating with nodes that use services;
- ✓ **Node graph** – for getting overview of interconnections between ROS nodes;
- ✓ **TF tree** – for viewing robots' transformation tree to get the transforms for all the links

The testing steps are following:
1. *Test master & slave communication in fully simulated environment*. That means, a simulated robot is loaded into Gazebo and the FT data is published manually using *rqt* plugin. The purpose of this test is to confirm that GCCF functionality works.

2. *Test jogger on the simulated robot manipulator*. That is to check by manually publishing velocity messages using keyboard interface. & twist messages using the virtual joystick node.

The aim of this test is to assure that the controller can move its EEF linearly in Cartesian space.

3. *Test contact control package on simulated robot.* Purpose of this test is to see how the robot may respond on the FT data coming from simulated FT sensor.

4. *Try contact control package on the manipulator while publishing FT data manually*. Purpose of this test is to see, how the robot responds to velocity commands published by the package.

The primary objective of my thesis was to control a manipulator when it is interacting with an environment maintaining some degree of compliance. Since the entire system is designed in simulation I have considered a virtual object (Figure 6.1**) inside Gazebo in form of a straight line which is fixed with the base plane. For testing the end effector tip is made to follow that virtual object for some pre specified distance keeping contact and trying to maintain a desired interaction (contact) force.**

After performing the initial tests listed under section 6.1, interactive teleoperation of the manipulator with the virtual object is carried on. The end effector is made to follow a position trajectory along Y axis inside the simulation world.

Gazebo is a dynamic simulator wrapper for ROS based on the Open dynamic engine. It allows dynamic simulations of robot manipulators communicating through ROS. The gazebo simulations used parameters from real robot because parameter identification of the real hardware resulted in base parameters for the simplified kinematic model.

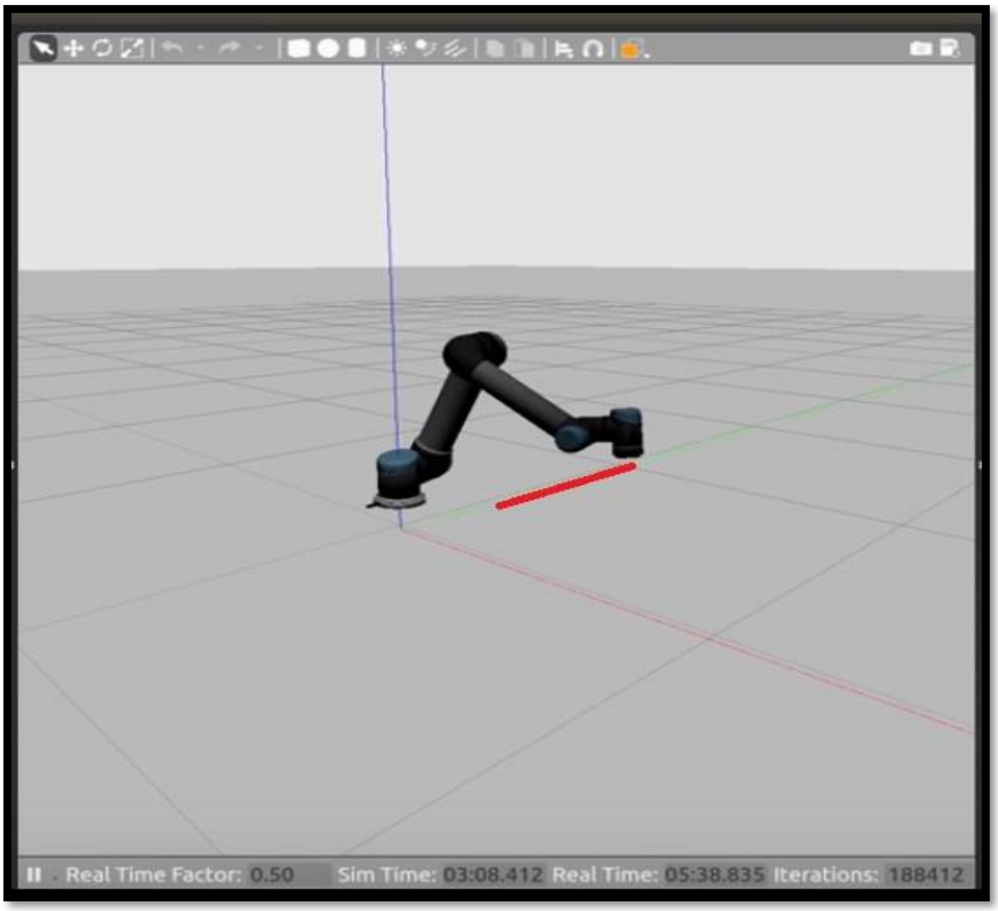


Figure 6.1 **Screenshot of Control simulation in Gazebo with force tracking on object (red line)**

6.2 Results

To validate the effectiveness of our simulation design & the performance of the implemented control strategies during interaction with the environment certain experimental results are presented in the following section.

During tests, ROS inbuilt debugging tools are being used. The basic *rqt* is a software framework of ROS that utilizes different GUI implementations in the form of plugins. We have used *rqt_plot* which provides a GUI plugin for visualizing 2D plot using different plotting methods.

6.2.1 Position Control in Free Space Tracking Results

At first in order to test whether simulation of teleoperation of a serial robot manipulator is working properly the jogging of the arm has been tested. The position control of the arm in free space is tested. Hence no interaction is considered. The jogging of the arm is performed in order check whether the serial robot manipulator is seamlessly following the joystick commands from the user end.

In this experiment, the arm was controlled by *virtual joystick*, and position of the end effector in Y axis was considered. It is visible that arm is able to reach position of the joystick; however, inverse kinematics adds delay that affects its overall responsiveness. Figure shows the response.

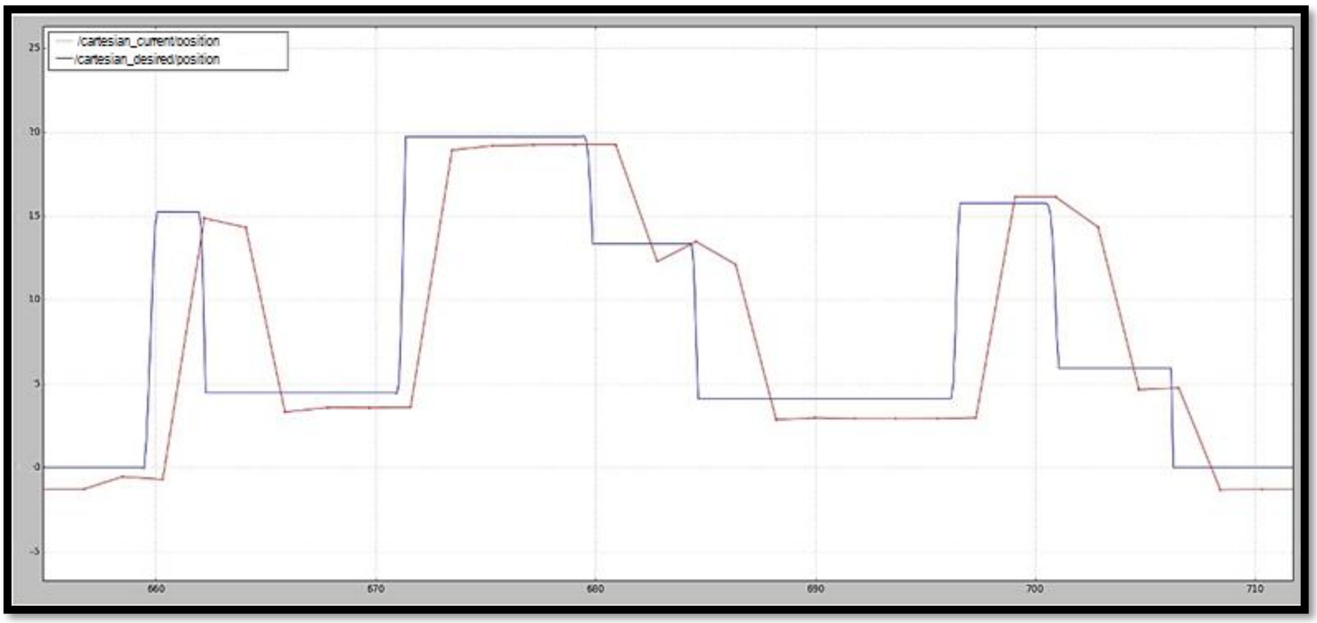


Figure 6.2 **Position of joystick and UR5 end effector in Y axis.**

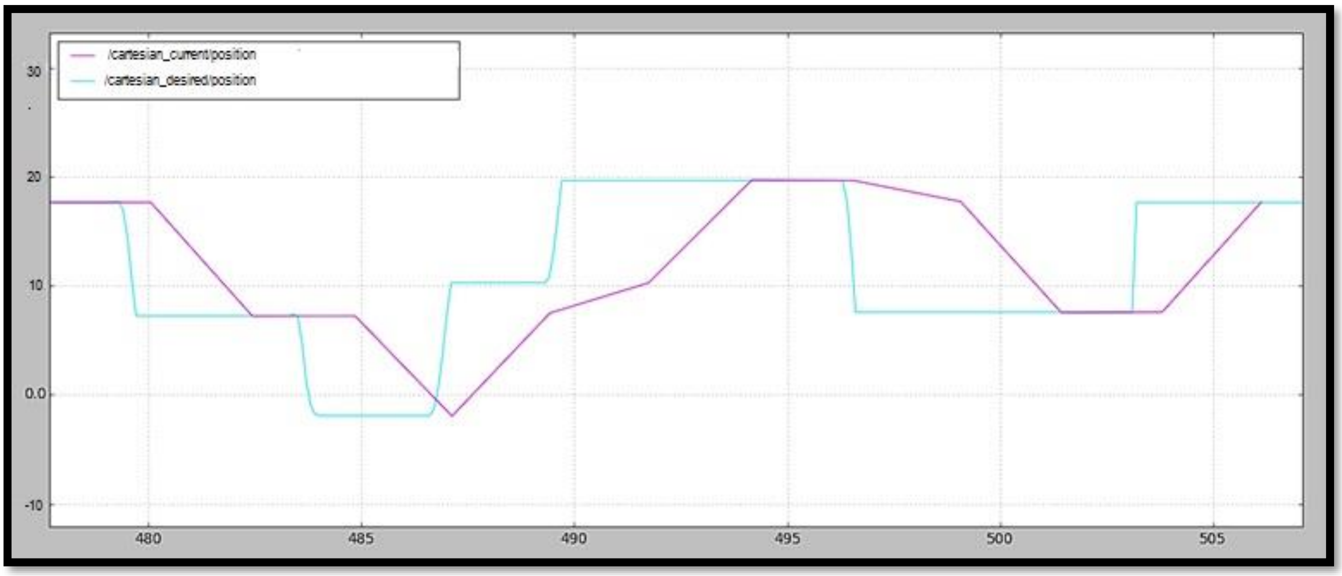


Figure 6.3 **Position of joystick and UR5 end effector in X axis**

6.2.2 Position Control based Force Tracking Results

For simulation of interaction control of a serial robot manipulator the manipulator needs to interact with the environment. On interaction with the environment contact forces will be generated at the point of interaction.
To measure that force a 6 axis force/torque sensor has been affixed with the wrist of our UR5 manipulator. Details of implementation have been covered in the previous chapter. A virtual object in form of massless spring damper is used as a manipulation object

Figure shows the simulation in gazebo for the interaction control with end effector moving 20 mm in the y-direction while tracking a desired tensile force of 5 N on the object (red). For performing this test, the stiffness for equation (5.1) under section 5.7 selected is K_e = 1000 [N/m]. The controller is running at 450 Hz. Figure shows relevant data of the y-position of the end effector (above) and force acting on the object (below). The end effector moves in the y direction and results in the satisfactory tracking of the Force.

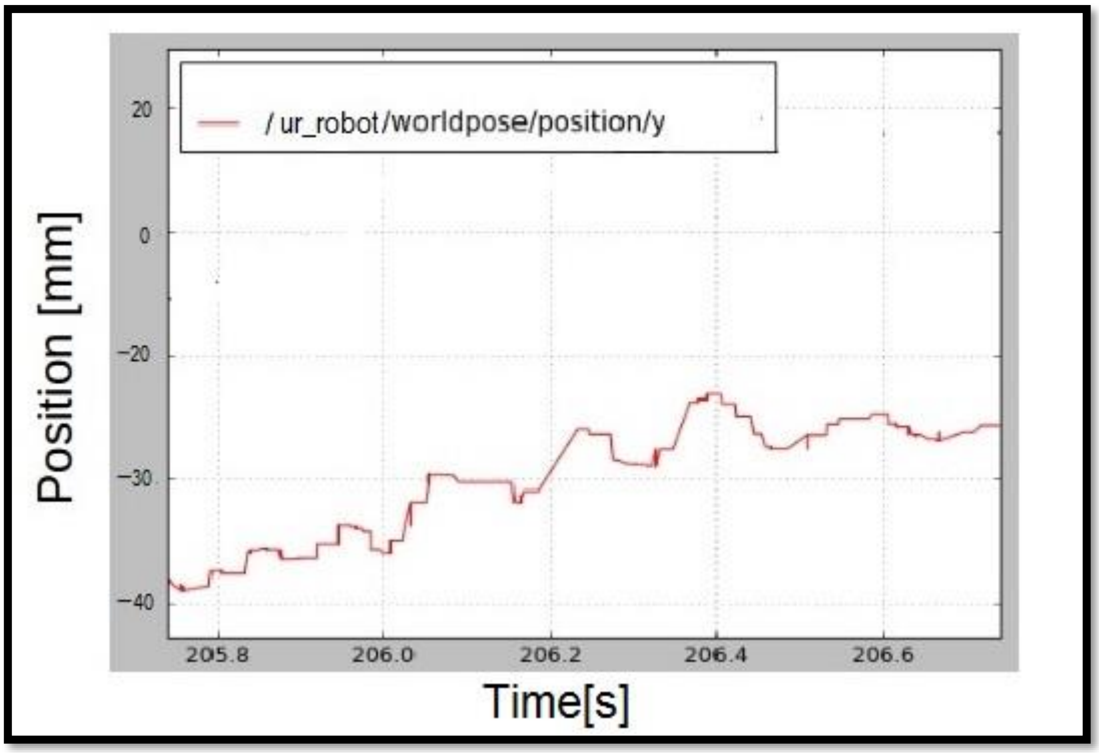


Figure 6.4 **Response of the controller for tracking a desired position trajectory**

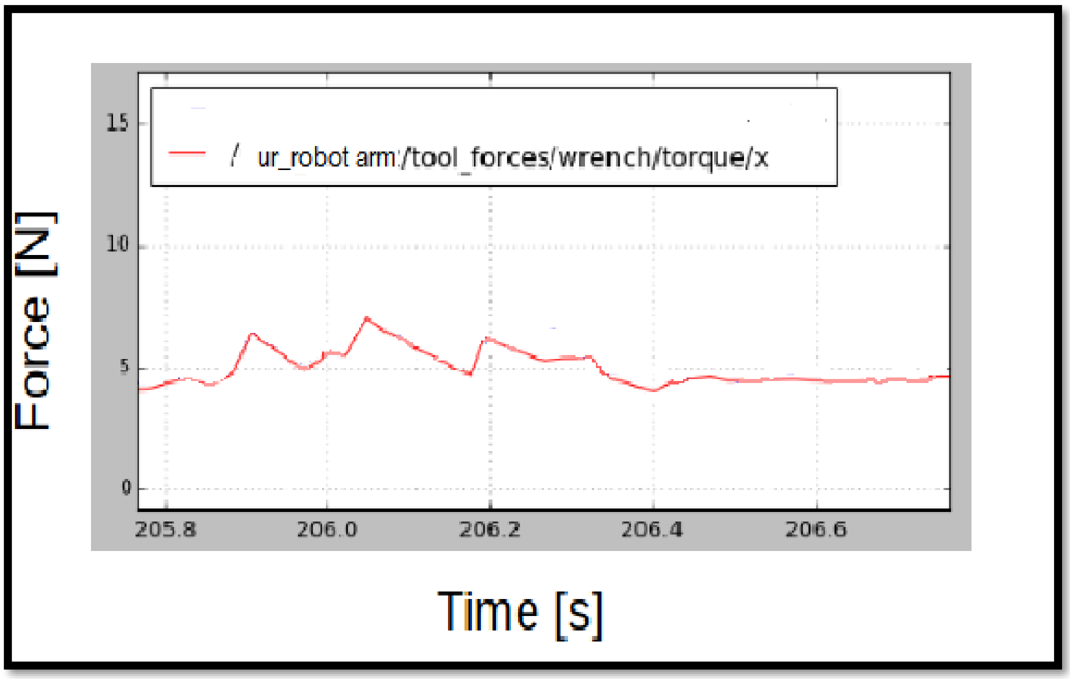


Figure 6.5 **Response of the controller for tracking a desired vertical force of 5N when the end effector is moved along y direction for 20 mm horizontally.**

6.2.3 Position Based Force Tracking

In order to check validity and closeness of the robot kinematics calculations, the motion of the slave side manipulator is being controlled using a standard Cartesian position controller. The slave manipulator is made to move to compensate the force that is acting on the end effector tip which is measured using a wrist force sensor. It is desired to maintain a force of 5N vertically. When the force deadband is 0.2N and the controller is running at 500 Hz Figure shows the result for this test having the first three curves from top showing the desired position and actual position [m] along x, y and z directions. And the curve in the bottom shows the force output of the force sensor. The horizontal axis is selected as the inbuilt timer of the ROS working platform. It is clear from the figure that, force tracking results are not stable enough .

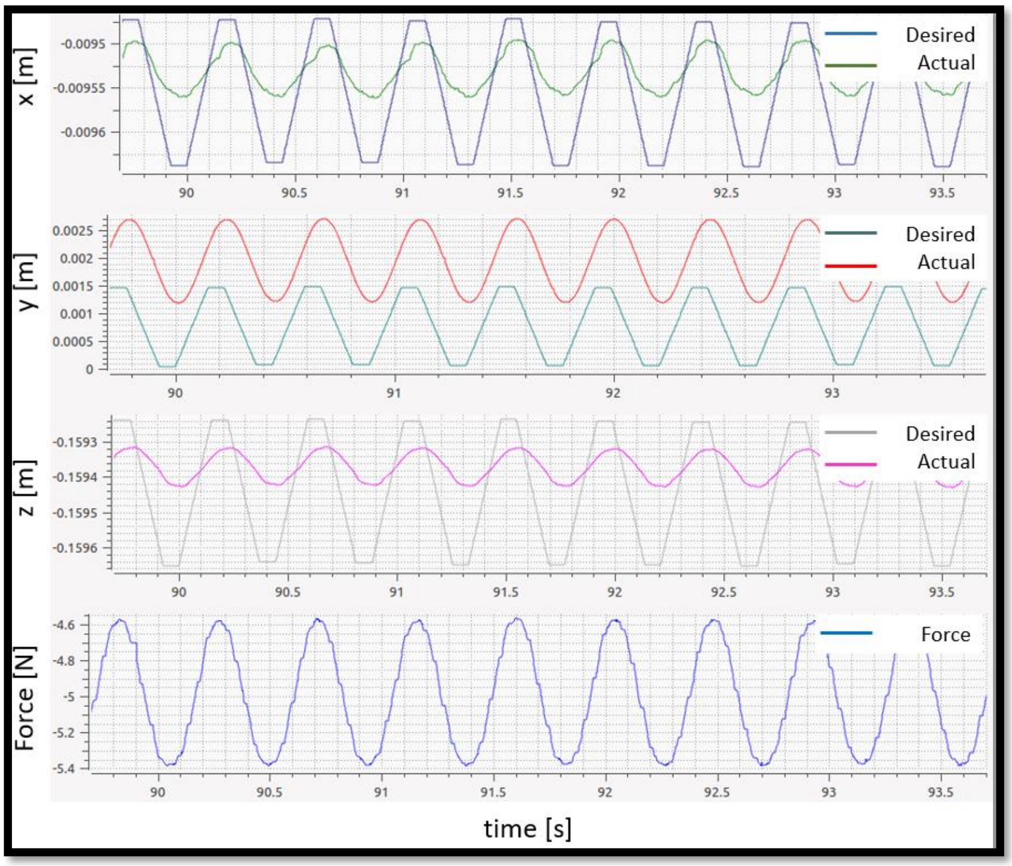


Figure 6.6 **Position based Cartesian force tracking result (Not stable)**

When the deadband is changed to 0.4N (Figure 6.7) the force tracking is more stable when the EEF is moved for 5 cm this time along the x-direction. This result shows that that a position based Cartesian force tracking is dependent on environment stiffness and the gains needed to be suitably adjusted.

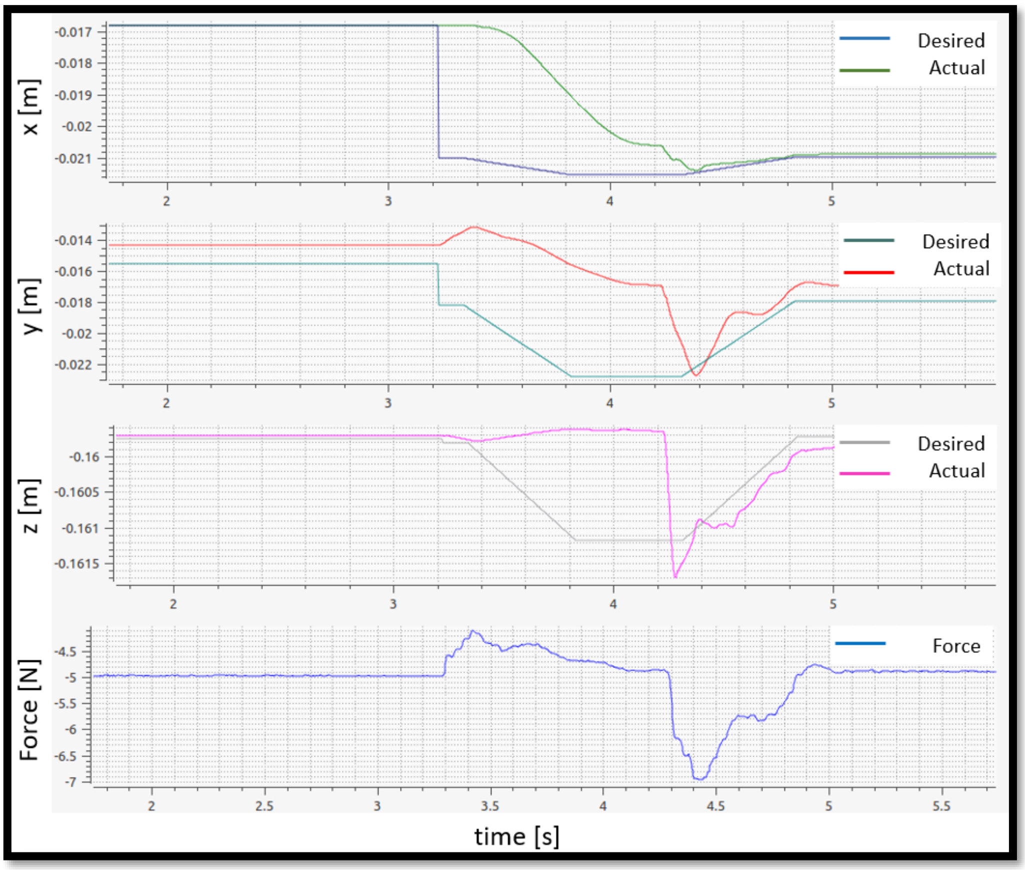


Figure 6.7 **Position based Cartesian force tracking result (More stable)**

Chapter 7

Conclusion & Future Work

Although there have been remarkable advancements in robotic capabilities in research labs and particular industry applications since the first years of reprogrammable robotic control have arrived, many industries, still limit their use of robotics to simple pick and place and non-contact tasks. And in cases when contact-tasks are considered, overall costs are typically much higher. Also most of the commercially available robot arms have no access to their individual joint actuator currents hence the only option is to append a wrist torque sensor at the tool tip to make the force measurements at the contact points. And there are certain manipulators like KUKA LBR iiwa which are light weight & due to its joint torque sensors, they can detect contact immediately and reduces its level of force and speed instantly. Its position and compliance control enables it to handle delicate components easily without exceeding the permissible force. But they come at the cost of their high price. Hence designing a realistic simulation and simulating the environment, manipulators & sensors gives an easy hassle free working platform from the research point of view. Thus it can be used to validate test algorithms in a virtual environment. As complex robots are usually managed by robotics frameworks, framework-simulator integration is a powerful tool, which allows design and verification of algorithms implemented straight into frameworks. The Gazebo simulator is already integrated with the Robot Operating System (ROS) hence has its importance to the robotics community. To export simulation resources, framework components are instantiated and synchronized inside a Gazebo system plug in. Each component is a C++ class implemented separately from others classes.

The work presented in this thesis creates a test bed for finding interaction control issues, for controlling the manipulator trajectories during interaction with environment or by varying the contact force information maintaining a desired force profile permissible both for manipulator & the environment depending majorly upon the task specified. However the current scenario with Gazebo simulator has a limitation that it only works with rigid bodies that move through articulation and not by deformation. Gazebo supports 3 types physics engine namely ODE (Open Dynamics Engine), Bullet Physics Engine & DART (Dynamic Animation & Robotics Toolkit). By default Gazebo uses ODE for all the simulations and Open Dynamics Engine (ODE) is a free, industrial quality library for simulating articulated rigid body dynamics. And because of its support for ROS, it is limited by the fact that it cannot yet simulate deformable objects.

Gazebo has two main parts, the server and the client. The server processes the physics of the environment, which involves calculating the full state of each body involved in the simulation taking into account the forces and velocities on each object as well as any additional data inputted from external controllers. The client receives pose data in messages from the server and displays the environment in a neat graphical user interface that adds convenient functionality and visualization. The server allows for the use of several different physics engines, each of which is optimized for different kinds of simulations.

However to simulate **deformable objects** the Bullet physics engine can be used. While Gazebo supports Bullet, it does not yet support Bullet's soft body libraries specially. Gazebo does, however, support third party plugins that allow for hard coded customization. These plugins needed to be written separately to add Bullet soft body functionality to Gazebo.

Bullet simulates soft bodies as collections of nodes, links, and faces. Nodes contain the pose, mass, force, and velocity information of each vertex in the soft body. The links act as springs between the nodes and allows for customization of deformation constants like bending and stretching. The faces give the soft body surface area allowing it to interact with fluid forces like wind. Collision shapes are commonly used in simulations to allow the user to simplify the simulated object in order to cut back on computational strain. Bullet processes rigid body collisions by determining the proximity of each body's collision shape. However, this shape is determined at compilation time, which is problematic for soft objects because their shape changes during the simulation. Bullet handles collisions between a soft body and another body by iterating through the soft body's nodes any checking if any of them is in contact with the other body's collision object.

If so then compliant motion can be more realistically studied as the environment will in that case provide varying reactance (nonlinear) that can be used to simulate human like body or even very high precision working environment. The interaction forces will vary dynamically hence adaptive control paradigms could be easily worked on in the simulation environment.

whereas now a model for the environment (contact point) has to be taken into consideration for testing while working on a constrained surface.

Use of Active Force feedback joysticks can accurately provide the reaction forces at the contact point of interaction back to user side using virtual spring law. This will enable successful bilateral teleoperation simulations in real-time.

For example **Geo magic Touch** is a motorized device that applies force feedback on the user's hand, allowing them to *feel* the objects and producing true-to-life touch sensations as user manipulates any 3D objects. With their use in research areas it becomes so intuitive to actually get a presence of the environment or the interaction on the master side accurately.

Use of force estimation methods like **Active observers design** (AOB) (AOBs; Cortesão 2002) in the control loop can improve the teleoperation performance by reducing the time delay.

However in our work we have considered the contact model as a massless spring damper system for simplicity and bring the effect of compliance on the environment. In real scenario for a compliant surface like a human body the reaction forces are nonlinear hence other contact models can be used to verify the stability of the results.

We have also considered that the robot is making contact at a single point which is ideally good at the initial testing but consideration of multiple contact points will bring more realistic situations that can be worked upon.

6.2 Conclusion

The **novelty** of my work lies in the fact that the entire system has been designed and operated in Gazebo Simulator and Robot Operating System **(**ROS) platform. ROS being an open source platform along with the 3D multi robot simulator Gazebo the implementation should be universally acceptable. Also using ROS as a design platform features a high degree of modularity in terms of software.

In this work, we have implemented a system which allows to teleoperate, in real time, an industrial robot in an intuitive way using simulated virtual joystick. The implemented software packages exploit the potentialities and the modularity of ROS, allowing .the developed software framework to work with other robots family compatible with ROS architecture. The task space of an industrial robot could be greatly increased in these fields by expanding control to include contact task procedures using a controller that is easily accessible and reconfigurable

In general experimental setup based on Gazebo and ROS has shown good results. However one of the major problems we faced was the significant drop of performance (in terms of coherence with wall-clock time) during dynamic interaction with other object.

Although the divergence of wall-clock and simulated clock time may be treated as a time-delay (a property of any real teleoperation system) it poses additional difficulties which we were trying to avoid by using simulation.

To summarize, following are the pros and cons of the developed system are formulated:

Pros

Free of charge software. Most packages in ROS are available under permissive BSD-licence, while Gazebo is distributed under Apache Licence v2.0. Thus the system can be used in commercial applications.

Rapid prototyping. The large number of very generic ROS packages allows to rapidly build a prototype of a system. Particular subsystems later can be replaced with more sophisticated versions without changing the interface. Compatibility with real-world robots. Since the system utilizes conventional ROS interfaces the simulated environment can be replaced with the real teleoperation system with very little effort.

Cons

Not real-time. Although certain ROS packages, e.g. pr2 controller manager, are able to work in the hard real-time loops, the simulation subsystem, i.e. Gazebo, is not designed for it. That introduces a variable time-delay, which is also depends on the complexity of simulated environment.

Difficult to tune physics engine parameters. The version of Gazebo that we used in our tests still utilize legacy.Open Dynamic Engine (ODE). The main problem of ODE is that the parameters are not very intuitive to tune and the engine itself is not actively developed. The newer versions of Gazebo are promised to support Bullet physics engine, which should help to solve some of the issues and also bring new features such as soft body contacts and GPU-accelerated calculations. Nevertheless, developed system possesses the great potential that mitigates most of the problems we mentioned above.

6.3 Future Work

Some future aspect includes:

- Incorporation of deformable objects in place of rigid bodies in Gazebo environment can have more realistic compliant control simulations as then varying reaction forces will be taken into account.
- Using active force feedback devices as the master device can feel the working environment seamlessly.
- Inclusion of force estimation techniques in the control loop can improve the time delay.
- For achieving human like compliance contact model should be highly nonlinear.

References

1. Peter F. Hokayem and Mark W, Spong. "*Bilateral teleoperation: An historical survey*". Coordinated Science Laboratory, University of Illinois. 2006.
2. Sheridan T.B. and W.R. Ferrell. "*Human Control of Remote Computer Manipulators*". Massachusetts Institute of Technology. 1967.
3. Thomas B. Sheridan. "*Telerobotics, Automation, and Human Supervisory Control*". Massachusetts Institute of Technology. 1992.
4. Robot Operating System, Available at http://wiki.ros.org/ROS/Introduction.
5. J. G. C. Devol, "*Programmed article transfer*," US2988237 A, 13-Jun-1961.
6. S. Y. Nof, Handbook of Industrial Robotics, vol. 1. John Wiley & Sons, 1999
7. A. Armagnac, "*New Factory Worker: Teachable Robot Can Remember 200 Commands*," Popular Science, vol. 181, no. 2, pp. 79–81, Aug-1962.
8. D. E. Whitney, "*Force Feedback Control of Manipulator Fine Motions*," J. Dyn. Syst. Meas. Control, vol. 99, no. 2, pp. 91–97, Jun. 1977..
9. "*ATI Industrial Automation: Compliance Devices*." [Online]. Available: http://www.ati-ia.com/Products/compliance/compensator_main.aspx. [Accessed: 30-Mar- 2016].
10. J. De Schutter, D. Torfs, and H. Bruyninckx, "*Robot force control with an actively damped flexible end effector*," Robot. Auton. Syst., vol. 19, no. 2, pp. 205–214, Dec. 1996.
11. G. A. Pratt and M. M. Williamson, "*Series elastic actuators*," in 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95. "Human Robot Interaction and Cooperative Robots", Proceedings, 1995, vol. 1, pp. 399–406 vol.1.
12. M. Quigley, A. Asbeck, and A. Ng, "*A low-cost compliant 7-DOF robotic manipulator*," in 2011 IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 6051–6058.
13. G. Tonietti, R. Schiavi, and A. Bicchi, "*Design and Control of a Variable Stiffness Actuator for Safe and Fast Physical Human/Robot Interaction*," in Proceedings of the

2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005, 2005, pp. 526–531.

14. E. G. Evan Ackerman, "*How Rethink Robotics Built Its New Baxter Robot Worker,*" 18-Sep-2012.[Online]. Available: http://spectrum.ieee.org/robotics/industrial-robots/rethink-robotics-baxter-robot-factory-worker. [Accessed: 30-Mar-2016].
15. M. T. Mason, "*Compliance and Force Control for Computer Controlled Manipulators*," IEEE Trans. Syst. Man Cybern., vol. 11, no. 6, pp. 418–432, Jun. 1981.
16. M. H. Raibert and J. J. Craig, "*Hybrid Position/Force Control of Manipulators*," J. Dyn. Syst. Meas. Control, vol. 103, no. 2, pp. 126–133, Jun. 1981.
17. H. Lipkin and J. Duffy, "*Hybrid Twist and Wrench Control for a Robotic Manipulator*," J. Mech. Transm. Autom. Des., vol. 110, no. 2, pp. 138–144, Jun. 1988.
18. J. M. Selig and P. R. McAree, "*A simple approach to invariant hybrid control*," in, 1996 IEEE International Conference on Robotics and Automation, 1996. Proceedings, 1996, vol. 3, pp. 2238–2245 vol.3.
19. K. P. Jankowski and H. A. ElMaraghy, "*Constraint Formulation for Invariant Hybrid Position/Force Control of Robots*," J. Dyn. Syst. Meas. Control, vol. 118, no. 2, pp. 290–299, Jun. 1996.
20. J. K. Salisbury, "*Active stiffness control of a manipulator in cartesian coordinates*," in 1980 19th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes, 1980, pp. 95–100.
21. H. Seraji, "*Adaptive admittance control: an approach to explicit force control in compliant motion*," in, 1994 IEEE International Conference on Robotics and Automation, 1994. Proceedings, 1994, pp. 2705–2712 vol.4.
22. N. Hogan, "*Impedance Control: An Approach to Manipulation: Part II—Implementation*," J. Dyn. Syst. Meas. Control, vol. 107, no. 1, pp. 8–16, Mar. 1985.
23. R. Anderson and M. W. Spong, "*Hybrid impedance control of robotic manipulators*," IEEE J. Robot. Autom., vol. 4, no. 5, pp. 549–556, Oct. 1988.
24. G. J. Liu and A. A. Goldenberg, "*Robust hybrid impedance control of robot manipulators*," in , 1991 IEEE International Conference on Robotics and Automation, 1991. Proceedings, 1991, pp. 287–292 vol.1.

25. S. Jung, T. C. Hsia, and R. G. Bonitz, "*Force tracking impedance control of robot manipulators under unknown environment*," IEEE Trans. Control Syst. Technol., vol. 12, no. 3, pp. 474–483, May 2004.
26. V. Mallapragada, D. Erol, and N. Sarkar, "*A New Method of Force Control for Unknown Environments*," in 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 4509–4514.
27. Y. Karayiannidis, G. Rovithakis, and Z. Doulgeri, "*Force/position tracking for a robotic manipulator in compliant contact with a surface using neuro-adaptive control*," Automatica, vol. 43, no. 7, pp. 1281–1288, Jul. 2007.
28. J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, "*Learning variable impedance control*," Int. J. Robot. Res., vol. 30, no. 7, pp. 820–833, Jun. 2011.
29. L. Kwang-Kyu, "*Force Tracking Impedance Control with Variable Target Stiffness*," 2008, pp. 6751–6756.
30. M. Jin, S. H. Kang, and P. H. Chang, "*Robust Compliant Motion Control of Robot With Nonlinear Friction Using Time-Delay Estimation*," IEEE Trans. Ind. Electron., vol. 55, no. 1, pp. 258–269, Jan. 2008.
31. S. H. Kang, M. Jin, and P. H. Chang, "*A Solution to the Accuracy/Robustness Dilemma in Impedance Control*," IEEEASME Trans. Mechatron., vol. 14, no. 3, pp. 282– 294, Jun. 2009.
32. S. H. Kang, M. Jin, and P. H. Chang, "*A Solution to the Accuracy/Robustness Dilemma in Impedance Control*," IEEEASME Trans. Mechatron., vol. 14, no. 3, pp. 282– 294, Jun. 2009.
33. R. Kikuuwe, "*A Sliding-Mode-Like Position Controller for Admittance Control With Bounded Actuator Force*," IEEEASME Trans. Mechatron., vol. 19, no. 5, pp. 1489–1500, Oct. 2014.
34. Morgan Quigley, Eric Berger, Andrew Y Ng, et al. Stair: Hardware and software architecture. In AAAI 2007 robotics workshop, volume 3, page 14, 2007.
35. Is ros for me. http://www.ros.org/is-ros-for-me/.
36. "ROS package," 2015. [Online]. Available: http://wiki.ros.org/Packages. [Accessed: 30-Apr-2019].
37. "ROS communication patterns," 2014. [Online]. Available: http://wiki.ros.org/ROS/Patterns/Communication. [Accessed: 30-Apr-2019].
38. "ROS-Industrial Robot Driver Specification," 2017. [Online]. Available: http://wiki.ros.org/Industrial/Industrial_Robot_Driver_Spec. [Accessed: 01-Apr-2019].

39. “ROS Messages,” 2016. [Online]. Available: http://wiki.ros.org/Messages. [Accessed: 09-Apr-2019].

40. S. Chitta, I. Sucan, and S. Cousins, “MoveIt! [ROS Topics],” IEEE Robot. Autom. Mag., vol. 19, no. 1, pp. 18–19, 2012.

41. “Wrench Message,” 2017. [Online]. Available: http://docs.ros.org/indigo/api/geometry_msgs/html/msg/Wrench.html. [Accessed: 09-Apr-2019].

42. “ROS Kinetic.” [Online]. Available: http://wiki.ros.org/kinetic. [Accessed: 15-May-2019].

43. D. Kortenkamp, R. Simmons, and D. Brugali, “*Robotic Systems Architectures and Programming*,” in *Springer Handbook of Robotics*, 2nd ed., Berlin: Springer-Verlag, 2016, pp. 283–302.

44. L. Prechelt, G. B. Uml, M. Feathers, R. Jeffries, and W. C. Xp, “*The essence of “Clean Code* ,’” 2014

45. “ROS package,” 2015. [Online]. Available: http://wiki.ros.org/Packages. [Accessed: 30-Apr-2019].

46. “Catkin,” 2017. [Online]. Available: http://wiki.ros.org/catkin. [Accessed: 06-May-2019].

47. “Gazebo,” 2018. [Online]. Available: http://gazebosim.org/. [Accessed: 07-May-2019]

48. “Ubuntu 16.04,” 2018. [Online]. Available: http://releases.ubuntu.com/16.04/. [Accessed: 06-May-2019].

49. “ROS TF,” 2017. [Online]. Available: http://wiki.ros.org/tf. [Accessed: 01-May-2019].

50. “Jog arm repository,” 2018. [Online]. Available: https://github.com/ut-ims [Accessed: 12-May-2019].

51. “ros_qml repository,” 2015.[Online]. Available: https://github.com/bgromov/ros_qml [Accessed: 10-May-2019].

52. M. Appo, “Keyboard publisher package,” 2018. [Online]. Available: https://github.com/ut-ims-robotics/keyboard_publisher. [Accessed: 12-May-2019].

53. “Rqt,” 2016. [Online]. Available: http://wiki.ros.org/rqt/Plugins. [Accessed: 13-May-2019].

54. S. Chiaverini and L. Sciavicco. “*The parallel approach to force/position control of robotic manipulators”*. IEEE Transactions on Robotics and Automation, 9(4):361–373, August 1993.

55. M. Mason. Compliance and force control for computer controlled manipula- tors. IEEE Transactions on Systems, Man, and Cybernetics, SMC-11(6):418– 432, June 1981