Simulation, Control and Testing of a Custom 5-DOF Robotic Manipulator System

by

ROOPAK M. KARULKAR

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2018

To my family and friends, for their constant motivation and support.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Kamesh Subbarao for the knowledge, motivation and guidance given to me as well as opportunity to work with a wonderful and diverse team at the Aerospace Systems Laboratory.

I would also like to thank my committee members, Dr. Alan Bowling and Dr. David Hullender whose tutelage has been extremely beneficial for my academic development.

I would like to thank my friends and colleagues at the Aerospace Systems Laboratory, Pengkai, Ameya, Denish, Paul, Kelvin, Murali, Karan, Rajnish, Mitchell, Tracie, Alok, Ziad, Pavan, Diganta, Abel and Abhishek for making the past two years enjoyable and contributing to creating an environment conducive to learning. I am thankful for the way they challenged me to explore different avenues in relation to my academic pursuits.

May 3, 2018

ABSTRACT

Simulation, Control and Testing of a Custom 5-DOF Robotic Manipulator System

ROOPAK M. KARULKAR, M.S

The University of Texas at Arlington, 2018

Supervising Professor: Kamesh Subbarao

Open chain manipulators are a well posed problem, however it was necessary to build a system customized to meet the needs of the Unmanned Ground Vehicle developed in the Aerospace Systems Laboratory. Considering that there are multiple such small ground vehicles, the main design constraint on the system was to be modular and lightwieght to enable quick swapping from both, a hardware and a software point of view.

The purpose of this thesis is design, simulation and control a modular 5-Degree-of-Freedom manipulator with versatility in end effector configuration. This was achieved through the Robot Operating System which brings a high degree of cross platform flexibility with minimal code modification. The manipulator was simulated and tested to execute a cartesian-space trajectory using a singularity robust inverse kinematics algorithm. The experimental setup has a cyber-physical architecture to allow for the necessary intensive computations to be offloaded to a more powerful ground station.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Manipulators add extended functionality to any robot. Manipulator designs may be varied to suit the desired application. Consequently, the Unmanned Ground Vehicle, ASL Gremlin, developed in the Aerospace Systems Laboratory was to be augmented with a manipulator to extend its capabilities. The manipulator is meant to be customizable, low-cost, easy to manufacture and custom fit to the rover. A varying degree of dexterity can be added to a manipulator depending on the desired application by adding more degrees of freedom. Manipulators with 6 to 8 degrees of freedom (DOF) are common in the industrial robotics. However, addition of degrees of freedom increases the complexity of the control system. The position and orientation of the end-effector, the attachment at the end of the manipulator, is described by a pose vector. A pose vector is a six-element vector containing the Cartesian coordinates (X-Y-Z) and the Euler angles (Φ, Θ, Ψ) of the end effector. Since the pose vector has 6 elements, any manipulator with more or less degrees of freedom presents a problem while being controlled.

Manipulator trajectories can be specified in the joint-space or cartesian space, ultimately the joint angles need to be calculated to control the robot. Cartesian space trajectories are specifed in terms of the pose variables of the end-effector and joint-space trajectories are specified in terms of the joint angles of the manipulator. Trajectories specified in the cartesian space need to be converted to the joint-space, i.e the necessary joint angles need to be computed. This gets computationally intensive,

but is a more intuitive way to specify trajectories. Since trajectories generated in the cartesian space have little to no information of the joint-space constraints like joint limits, special algorithms need to be implemented while solving for the joint angles to avoid joint singularities.

1.2 Previous Work

The manipulator end effector pose is described with 6 variables, while the manipulator has 5 joints. This results in a non-invertible Jacobian matrix. A Damped Least Squares pseudo-inverse was used as shown by Wampler [1] and Nakamura [2]. This addressed the problem of inverting a non-singular matrix as well as adding singularity robustness to the system. These methods do not address the effects of joint limits on inverse kinematics which is addressed in a modified and improved version of the Damped Least Squares algorithm by Na [3]. Another approach of selectively damping joints using the Damped Least Squares Method was suggested by Buss [4] which recommends clamping the length of the error vector between the current position and the target to a maximum distance. Siciliano provides another inverse kinematics approach where the error dynamics are taken into consideration so the joint velocity error is also taken into account [5]. Inverse kinematic solutions can be obtained analytically as well. One such analytical solution for a KUKA youBot oufitted with a manipulator is presented by Sharma [6]. Minimum jerk trajectory algorithms for manipulators have been previously used using trigonometric splines [7] and cubic splines [8]. However, the algorithm developed by Godbole [9] which uses a quintic polynomial and minimizes jerk was used for this application. This is the same algorithm used for the trajectory generated for the rover. This was done with a view towards integrating these two systems and make shareable code. Wang outlines

a cyber-physical system network of interacting agents [10] performing colaborative work.

1.3 Problem Description

This thesis presents a solution to add additional functionality to the ASL Gremlin Unmanned Ground Vehicle platform. This thesis addresses singularity robust trajectory following for the custom manipulator with a non-invertible Jacobian matrix. This thesis builds on the use of a cyber-physical system architecture for Unmanned Vehicle research.

1.4 Thesis outline

The thesis outline is as follows: The kinematic and dynamic modeling of the manipulator is performed in Chapter 2. Chapter 3 details the simulation of the manipulator in MATLAB and ROS for multiple cases for 2D and 3D trajectories. Chapter 4 contains the details of the hardware and experimental setup. Chapter 5 contains the experimental results. Finally, chapter 6 states some concluding remarks.

CHAPTER 2

MODELING OF THE MANIPULATOR

2.1 Kinematics

Kinematics of the system study the motion of the bodies in the system with out taking into account the forces and the torques acting on it. The two types of kinematics involved in motion control are forward and inverse kinematics. Forward kinematics is the calculation of the end-effector position and orientation when the joint angles are known. This is a fairly simple and intuitive process due to the manner in which the kinematic equations are set up. Inverse kinematics are used to calculate the joint angles necessary to achieve a desired end-effector pose. This is a more complex problem, because depending on the manipulator configuration, there may be multiple solutions that achieve the same end effector position. The manipulator was designed as a 5-DOF system as rotating the wrist joint was not a design requirement. All joints are revolute so it is a 5R manipulator.

2.1.1 Reference Frames

As the manipulator is a multi-body system, each body, each link, is assigned a reference frame. This is because the way a vector is expressed may differ from the way it's defined in the frame of interest. The kinematics are modeled under the assumption of a static inertial reference frame that is common to the observer and the robot. All end-effector vector quantities are related back to this inertial reference frame. This is done via a 3×3 rotation matrix ${}^{A}_{B}\boldsymbol{R}$ which gives the orientation of Frame B with respect to Frame A.

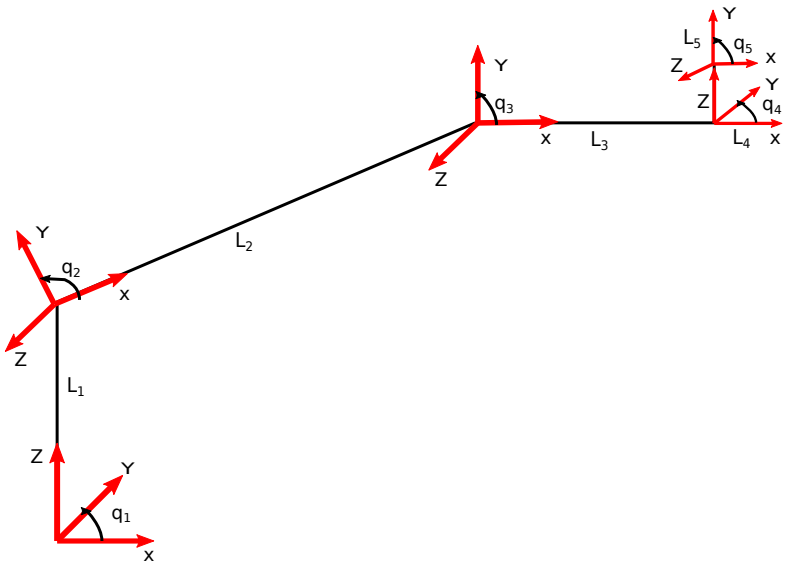

Figure 2.1: Kinematic model of the manipulator

2.1.2 Frame Assignment and Forward Kinematics

For each frame i, the z-axis, $\hat{Z}_i$ is coincident with the joint axis. The axis $\hat{X}_i$ is the common normal to $\hat{Z}_i$ and $\hat{Z}_{i+1}$ and points from joint j_i to j_{i+1}. The right hand rule is used to find $\hat{Y}_i$ and complete the frame. The frame assignment for the manipulator is seen in Figure 2.2 where $\hat{X}_i$ is shown in red, $\hat{Y}_i$ is shown in green and $\hat{Z}_i$ is shown in blue.

The kinematics were modeled using Modified Denavit-Hartenberg (DH) Convention [11]. The relationship between two links can be described by four DH Parameters as illustrated in Fig 2.3 [12]. The parameters for the fabricated manipulator are listed in Table 2.1.

The parameters are defined as follows

α_i = Angle from $\hat{Z}_i$ to $\hat{Z}_{i+1}$ about $\hat{X}_i$

a_i = Distance from $\hat{Z}_i$ to $\hat{Z}_{i+1}$ along $\hat{X}_i$

d_i = Distance from $\hat{X}_{i-1}$ to $\hat{X}_i$ along $\hat{Z}_i$

θ_i = Angle from $\hat{X}_{i-1}$ to $\hat{X}_i$ along $\hat{Z}_i$

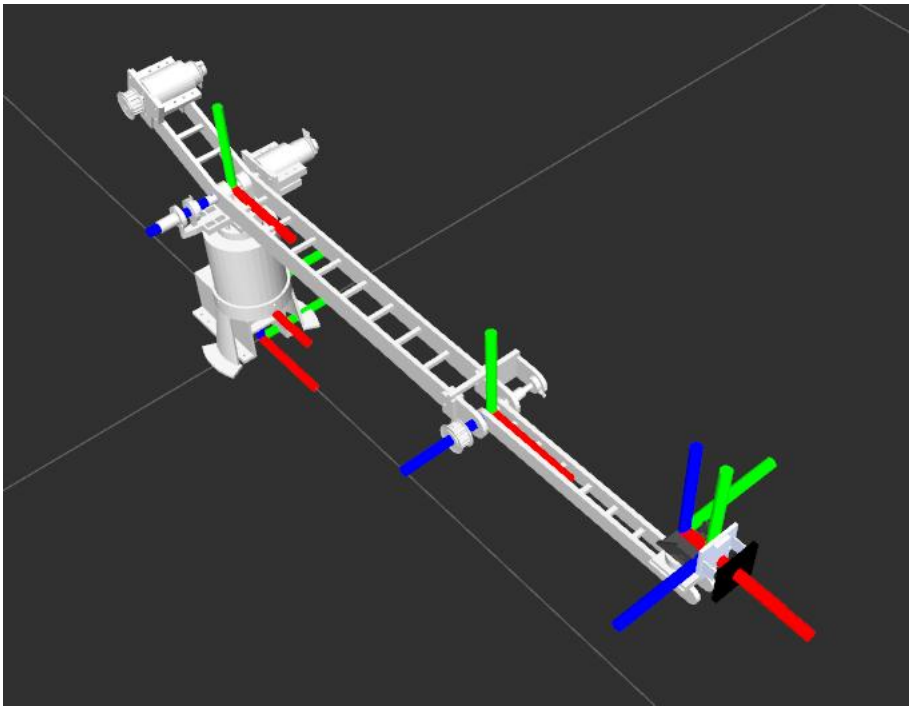

Figure 2.2: Body-attached frames as visualized in RViz

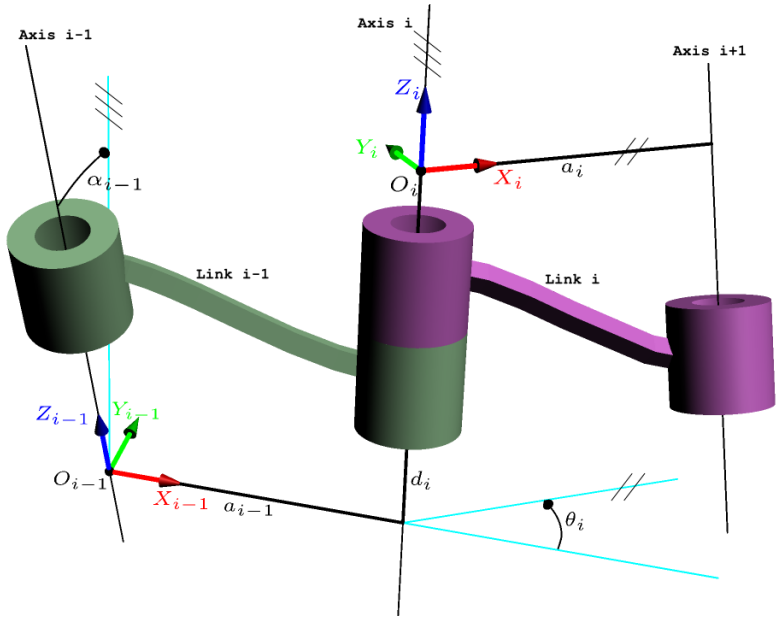

Figure 2.3: D-H Parameters

θ_i are the joint angles of the manipulator.

Table 2.1: DH Parameters of the Manipulator

	α_{i-1}	a_{i-1} (mm)	d_i (mm)	θ_i
1	0	0	218	q_1
2	90	0	0	q_2
3	0	393.7	0	q_3
4	-90	225.35	58.53	q_4
5	90	27.5	0	q_5

The DH Parameters were used to create a transformation matrix for every body-attached frame [11] T such that

$$ {}^{i-1}_{i}\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.1} $$

where $\boldsymbol{R}$ is the rotation matrix between two frames and $\mathbf{p}$ is the position vector to the frame i from frame $i-1$.

$$ {}^{i-1}_{i}\mathbf{T} = \begin{bmatrix} c(\theta)_i & -s(\theta)_i & 0 & a_{i-1} \\ s(\theta)_i c(\alpha)_{i-1} & c(\theta)_i c(\alpha)_{i-1} & -s(\alpha)_{i-1} & -s(\alpha)_{i-1} d_i \\ s(\theta)_i s(\alpha)_{i-1} & c(\theta)_i s(\alpha)_{i-1} & c(\alpha)_{i-1} & c(\alpha)_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2} $$

The generalized formula for the transformation matrix is given in Eq. 2.2 where $c(\cdot) = \cos(\cdot), \quad s(\cdot) = \sin(\cdot)$ The end-effector frame or the tool frame can be related to the inertial reference frame by multiplying all of the relevant transformation matrices.

$$ {}^{0}_{5}\mathbf{T} = {}^{0}_{1}\mathbf{T}\,{}^{1}_{2}\mathbf{T}\,{}^{2}_{3}\mathbf{T}\,{}^{3}_{4}\mathbf{T}\,{}^{4}_{5}\mathbf{T} \tag{2.3} $$

The pose vector, $\mathbf{x}$, of the end effector is used to describe its position and orientation $\mathbf{x} = [x\ y\ z\ \Phi\ \Theta\ \Psi]^T$ where $x,\ y,\ z$ are the cartesian coordinates of the end-effector and $\Phi,\ \Theta,\ \Psi$ are the Z-Y-X Euler angles as shown in Fig 2.4 [11].

$$x = T[1,4] \tag{2.4}$$

$$y = T[2,4] \tag{2.5}$$

$$z = T[3,4] \tag{2.6}$$

$$\Phi = \text{atan2}\left(T(2,1), T(1,1)\right) \tag{2.7}$$

$$\Theta = \text{atan2}\left(-T(3,1), \sqrt{T(1,1)^2 + T(2,1)^2}\right) \tag{2.8}$$

$$\Psi = \text{atan2}\left(T(3,2), T(3,3)\right) \tag{2.9}$$

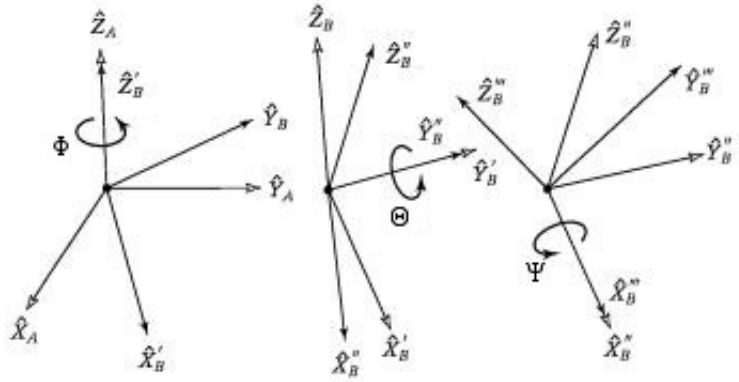

Figure 2.4: Z-Y-X Euler Angle representation

The end effector tool for this manipulator is designed to be modular and can be switched out. To relate the new tool frame to the inertial reference frame, post-multiply Eq. 2.3 with a transformation matrix relatng Frame 5 to the new tool frame. In its current configuration, the manipulator has 5 joints and no constraints so it is a 5 degree-of-freedom system.

2.2 Dynamics

The equations of motion were derived using the iterative Newton-Euler Method due to ease of computation. The manipulator was a fixed base case so the initial conditions for the base are zero. The equations are generated in two stages. The first stage involves computing the angular and linear velocities and accelerations, and the forces and torques on all links starting from the first link to the last link. The second stage computes the net force and torque acting on the links starting from the last link to the first link [11]. The manipulator is a multi-body chain made of all the links so the velocity of link [i] and the velocity of the joint [i+1] make up the velocity of link [i+1]. The angular velocity and accelerations of the links are computed as follows where $i : 0 \rightarrow l-1$ and l is the number of links.

$$
\begin{aligned}
{}^{i+1}\boldsymbol{\omega}_{i+1} &= {}^{i+1}_{i}\mathbf{R}\ {}^{i}\boldsymbol{\omega}_{i} + \dot{q}_{i+1}\ {}^{i+1}\hat{Z}_{i+1} && (2.10)\\
{}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} &= {}^{i+1}_{i}\mathbf{R}\ {}^{i}\dot{\boldsymbol{\omega}}_{i} + {}^{i+1}_{i}\mathbf{R}\ {}^{i}\boldsymbol{\omega}_{i} \times \dot{q}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} + \ddot{q}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} && (2.11)
\end{aligned}
$$

The translational accelerations of the links are then computed as,

$$
{}^{i+1}\dot{\boldsymbol{v}}_{i+1} = {}^{i+1}_{i}\mathbf{R}\left({}^{i}\dot{\boldsymbol{\omega}}_{i} \times {}^{i}\boldsymbol{p}_{i+1} + {}^{i}\boldsymbol{\omega}_{i} \times \left({}^{i}\boldsymbol{\omega}_{i} \times {}^{i}\boldsymbol{p}_{i+1}\right) + {}^{i}\dot{\boldsymbol{v}}_{i}\right) \qquad (2.12)
$$

The translational accelerations of the centers of mass of the links are also necessary to compute the inertial for and moment acting on the center of mass of each link. These accelerations are calculated as,

$$
{}^{i+1}\dot{\boldsymbol{v}}_{C_{i+1}} = {}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} \times {}^{i+1}\boldsymbol{p}_{C_{i+1}} + {}^{i+1}\boldsymbol{\omega}_{i+1} \times \left({}^{i+1}\boldsymbol{\omega}_{i+1} \times {}^{i+1}\boldsymbol{p}_{C_{i+1}}\right) + {}^{i+1}\dot{\boldsymbol{v}}_{i+1} \qquad (2.13)
$$

where ${}^{i+1}\boldsymbol{p}_{C_{i+1}}$ is the location of the center of mass of link [i+1] expressed in frame {i+1}.

The inertial force and moment acting on the link [i+1] which cause the motion are computed as,

$$^{i+1}\boldsymbol{F}_{i+1} = m_{i+1}\ ^{i+1}\dot{\boldsymbol{v}}_{C_{i+1}} \tag{2.14}$$

$$^{i+1}\boldsymbol{N}_{i+1} = {}^{C_{i+1}}\boldsymbol{I}_{i+1}\,^{i+1}\dot{\boldsymbol{\omega}}_{i+1} + {}^{i+1}\boldsymbol{\omega}_{i+1} \times {}^{C_{i+1}}\boldsymbol{I}_{i+1}\,^{i+1}\boldsymbol{\omega}_{i+1} \tag{2.15}$$

${}^{C}I$ is the inertia tensor such that C_i is located at the center of mass of the body and is oriented like frame {i}.

The forces and moments are then calculated using inward iterations such that $from\ i : l-1 \rightarrow 0$. These are the forces exerted on links by their neighboring links.

$$^{i}\boldsymbol{f}_i = {}^{i}_{i+1}\boldsymbol{R}\ ^{i+1}\boldsymbol{f}_{i+1} + {}^{i}\boldsymbol{F}_i \tag{2.16}$$

$$^{i}\boldsymbol{n}_i = {}^{i}\boldsymbol{N}_i + {}^{i}_{i+1}\boldsymbol{R}\ ^{i+1}\boldsymbol{n}_{i+1} + {}^{i}\boldsymbol{p}_{C_i} \times {}^{i}\boldsymbol{F}_i + {}^{i}\boldsymbol{p}_{i+1} \times {}^{i}_{i+1}\boldsymbol{R}\ ^{i+1}\boldsymbol{f}_{i+1} \tag{2.17}$$

$\boldsymbol{f_i}$ is the force exerted on link [i] by link [i+1]. $\boldsymbol{n_i}$ is the torque exerted on link [i] by link [i+1].

The required joint torque $\boldsymbol{\tau_i}$ is the Z component of the torque ${}^{\boldsymbol{i}}\boldsymbol{n_i}$. The torque on the end effector $\boldsymbol{\tau_{l-1}}$ would be zero if the end effector is free to move in space but non-zero if there are interactions with objects because the force $\boldsymbol{f_{l-1}}$ would be non-zero.

$$\boldsymbol{\tau_i} = {}^{i}\boldsymbol{n}_i^T\ {}^{i}\hat{Z}_i \tag{2.18}$$

The intertia tensors ${}^{C}\boldsymbol{I}$ were obtained from a high fidelity CAD model. The acceleration due to gravity is applied to ${}^{0}\dot{\boldsymbol{v}}_0$ upward. The equations of motion are as follows

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q})g \tag{2.19}$$

The mass, gravity and coriolis matrices can be obtained from symbolically solving Eq. 2.18 such that

$$\mathbf{M}(\mathbf{q}) = \frac{\partial \boldsymbol{\tau}}{\partial \ddot{\mathbf{q}}} \tag{2.20}$$

$$\mathbf{G}(\mathbf{q}) = \frac{\partial \boldsymbol{\tau}}{\partial g} \tag{2.21}$$

$$\mathbf{B}(\mathbf{q}) = \boldsymbol{\tau} - \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} - \mathbf{G}(\mathbf{q})g \tag{2.22}$$

where $\mathbf{M}(q) \in \mathcal{R}^{6\times5},\ \mathbf{G}(q) \in \mathcal{R}^{6\times1},\ \mathbf{B}(q,\dot{q}) \in \mathcal{R}^{6\times1}$

In order to be valid, the mass matrix, $\mathbf{M}(q)$, needs to be full rank, positive definite and symmetric. The mass matrix for this manipulator met these conditions.

CHAPTER 3

SIMULATION OF THE MANIPULATOR

3.1 Workspace Analysis

It is essential to know the manipulator workspace before planning end-effector trajectories. The workspace consists of all the points reachable by the end effector given the joint limits of each angle. The joint limits for the manipulator are listed in Table 3.1

Table 3.1: Joint Angle limits (degrees)

Joint	Lower limit	Upper limit
1	0	360
2	-7	63
3	-90	90
4	-90	90
5	-90	90

These joint angle ranges result in over 1.4×10^{11} possible joint configurations. Generating points for that would be computationally wasteful. To overcome this, the joint angles for joints 4 and 5 were held at 0. This does not affect the workspace representation much because these two joints are mainly used for orientation control. The remaining 3 joints were incremented by 5 degrees and the end effector position was calculated. This method reduced the number of points to 36,288 and made it feasible to calculate the workspace.

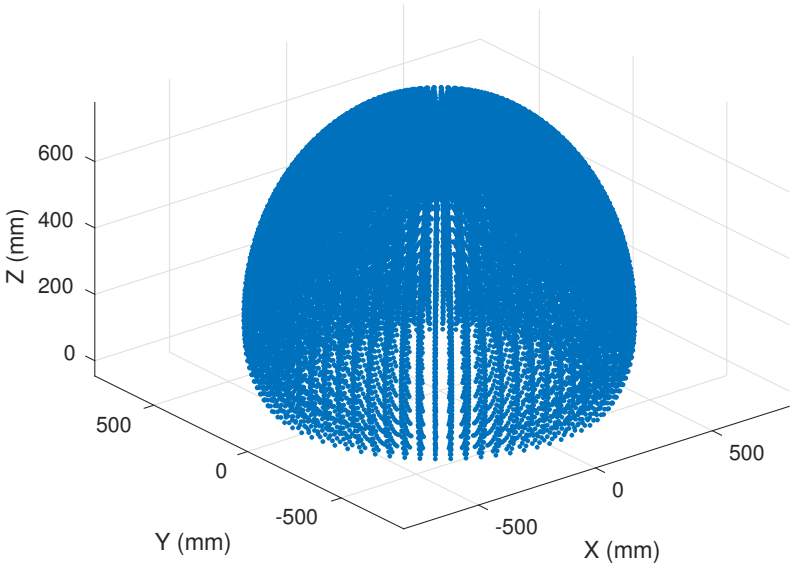

Figure 3.1: Manipulator Workspace

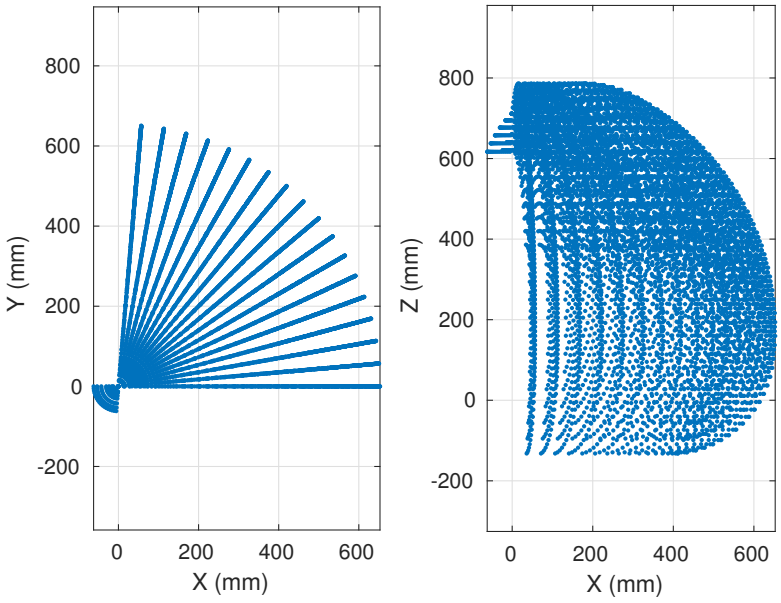

Figure 3.2: Manipulator Workspace - Quarter Section

3.2 End-Effector Trajectory

There are two ways of generating trajectories, namely, joint-space trajectories and operational-space trajectories.Operational-space trajectories are defined in terms of operation-space variables like the position of the end effector in cartesian coordinates and the orientation, which in this case is expressed using Euler angles however, quaternions may be used as well. Joint-space trajectories are defined in terms of the joint angles of the manipulator. Joint-space trajectories are the most convenient from a control point of view because singularities are easily avoided and there is no Jacobian inversion involved. Generating just a joint-space trajectories is effective if only the initial and final pose are of consequence and the path traced by the end effector to reach the final pose has no restrictions on it. A joint-space trajectory was generated by solving for the angles corresponding to specified waypoints using a Nonlinear Least-Squares solving algorithm, such as Levenberg-Marquardt minimization [13], of the end-effector pose error. The resulting angles were fed into the algorithm listed below which resulted in a minimum-jerk trajectory in the joint-space.

As illustrated in Fig 3.3, the end effector reaches the required pose with a joint -space trajectory, but the path taken deviates significantly from the desired path. However, in most cases, the path traced by the end-effector is just as important as reaching the final pose so an operational-space trajectory was chosen. The operational-space trajectory was first generated and then a joint-space trajectory was calculated using inverse kinematics such that the end-effector will trace the path outlined by the operational-space trajectory.

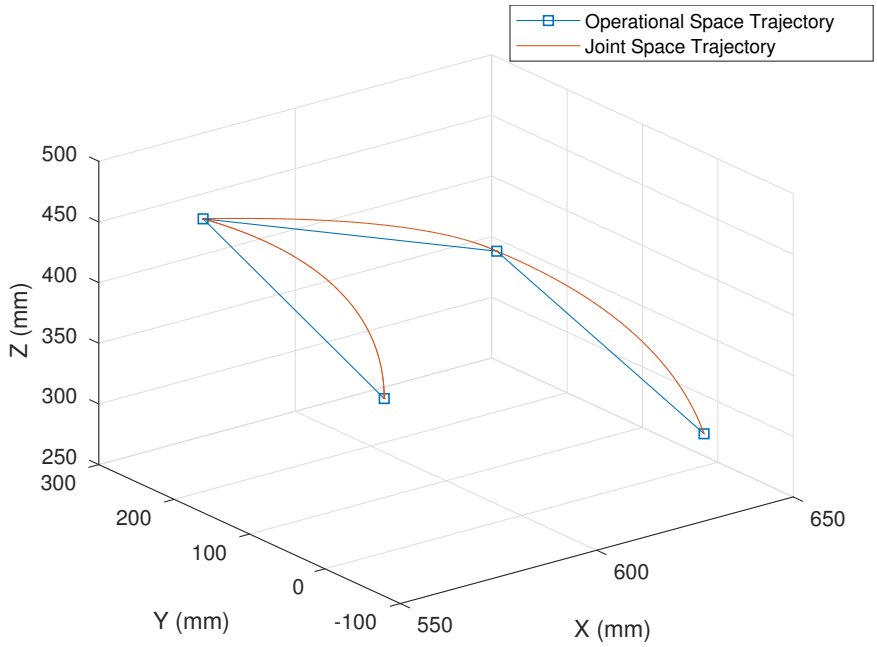

Figure 3.3: Difference between Operational and Joint-space trajectories

3.2.1 Trajectory Generation

Trajectory generation was done similar to the method described by Godbole et al [9]. A trajectory was generated using a fifth order polynomial using s as a general coordinate.

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{3.1}$$

$$\dot{s}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \tag{3.2}$$

$$\ddot{s}(t) = a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \tag{3.3}$$

The desired $\dot{s}_f$ and $\ddot{s}_f$ are zero so the coefficients and final time are as follow

$$a_0 = s(0) \tag{3.4}$$

$$a_1 = 0 \tag{3.5}$$

$$a_2 = 0 \tag{3.6}$$

$$a_3 = 10\sqrt{\frac{\Delta s}{t_f^3}} \tag{3.7}$$

$$a_4 = -15\sqrt{\frac{\Delta s}{t_f^4}} \tag{3.8}$$

$$a_5 = 6\sqrt{\frac{\Delta s}{t_f^5}} \tag{3.9}$$

$$t_f \geq \sqrt{\frac{10}{\sqrt{3}}\frac{\Delta s}{a_{max}}} \tag{3.10}$$

where $\Delta s = s_f - s_0$. Also, t_f denotes the time necessary for completion of the trajectory and t denotes the time elapsed the during motion of the particular segment of the trajectory. It is not reflective of the total simulation time, nor the global "wall" or system time. The coefficients generated in Eq. 3.5 to Eq. 3.9 ensure that jerk is minimized over the time t_f. Since the pose vector elements have two units, i.e. mm/s^2 and rad/s^2, the algorithm was supplied acceleration bounds for both position and orientation. The algorithm then calculcated a final time for both of those values and the lower of the two values was chosen to generate the trajectory. All the tests have been performed with an acceleration bound of 25 mm/s^2 for the position and 2 rad/s^2 for the orientation.

3.3 Inverse Kinematics

As mentioned earlier, inverse kinematics involves solving for the joint angles based on the position and orientation of the end-effector. As the generated trajectory

is defined in the operational-space, inverse kinematics allows us to calculate the joint angles necessary to follow that trajectory. This problem is complex and computationally intensive. There are two ways of solving the inverse kinematics problem, namely first order and second order inverse kinematics. First order kinematics allows the calculation of the joint velocity and position, whereas second order inverse kinematics allows for the calculation of joint accelerations as well. Both of these methods have been implemented in this thesis for various cases.

3.3.1 First Order Inverse Kinematics

Let $\mathbf{x} = f(\mathbf{q})$ be the nonlinear forward kinematics function. The end effector velocity can be mapped to the joint velocities as follows

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{3.11}$$

$$\mathbf{J}(\mathbf{q}) = \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}} \tag{3.12}$$

$$\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{-1}\dot{\mathbf{x}} \tag{3.13}$$

Numerical integration becomes necessary to solve for the joint angles. If allowed to run for a sufficient time, numerical integration suffers from drift and introduces errors into the system. Therefore, inverse kinematics solutions can be calculated using the operational-space error [5].

$$\mathbf{e} = \mathbf{x_d} - \mathbf{x} \tag{3.14}$$

$$\dot{\mathbf{e}} = \dot{\mathbf{x}}_\mathbf{d} - \dot{\mathbf{x}}$$

$$= \dot{\mathbf{x}}_\mathbf{d} - \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{3.15}$$

If the error dynamics are taken as an asymptotically stable system $\dot{\mathbf{e}} + K\mathbf{e} = 0$ gives us the following inverse kinematics algorithm.

$$\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{-1} \left(\dot{\mathbf{x}}_\mathbf{d} + \mathbf{K}\mathbf{e} \right) \tag{3.16}$$

where K is a positive definite gain matrix.

3.3.2 Second Order Inverse Kinematics

Second order inverse kinematics allows the calculation of the joint acceleration as well. This may be necessary for control schemes like inverse dynamics. By differentiating Eq. 3.15

$$\begin{aligned} \ddot{\mathbf{e}} &= \ddot{\mathbf{x}}_\mathbf{d} - \ddot{\mathbf{x}} \\ &= \ddot{\mathbf{x}}_\mathbf{d} - \frac{d}{dt}\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \\ &= \ddot{\mathbf{x}}_\mathbf{d} - \dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} - \mathbf{J}(\mathbf{q})\ddot{\mathbf{q}} \end{aligned} \tag{3.17}$$

If the error dynamics are proposed to be a second order asymptotically stable system $\ddot{\mathbf{e}} + \mathbf{K_d}\dot{\mathbf{e}} + \mathbf{K_p}\mathbf{e} = 0$

The inverse kinematics solution can then be calculated as follows.

$$\ddot{\mathbf{q}}_\mathbf{d} = \mathbf{J}^{-1}(\mathbf{q}) \left(\ddot{\mathbf{x}}_\mathbf{d} + \mathbf{K_d}\dot{\mathbf{e}} + \mathbf{K_p}\mathbf{e} - \dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} \right) \tag{3.18}$$

3.3.3 Damped Least Squares Inversion

The manipulator has 5 degrees of freedom and there are 6 variables in the pose vector so the Jacobian is a 6×5 matrix and is not easily invertible, therefore a pseudo-inverse was used. There may be some joint angle values which may result in Jacobian values that would cause a singularity when inverted. Therefore, the Jacobian was

inverted using the Damped Least Squares pseudo-inversion [14] method to make it singularity-robust.

$$\mathbf{J}^{\dagger} = \left(\mathbf{J}^T\mathbf{J} + \lambda^2\mathbf{I}\right)^{-1}\mathbf{J}^T \tag{3.19}$$

where λ is a damping factor such that $0 < \lambda \leq 1$. So Eq. 3.16 then becomes

$$\dot{\mathbf{q}}_{\mathbf{d}} = \mathbf{J}^{\dagger}(\mathbf{q})\left(\dot{\mathbf{x}}_{\mathbf{d}} + \mathbf{K}\mathbf{e}\right) \tag{3.20}$$

Using Eq. 3.18 and 3.19

$$\ddot{\mathbf{q}}_{\mathbf{d}} = \mathbf{J}^{\dagger}(\mathbf{q})\left(\ddot{\mathbf{x}}_{\mathbf{d}} + \mathbf{K}_{\mathbf{d}}\dot{\mathbf{e}} + \mathbf{K}_{\mathbf{p}}\mathbf{e} - \dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}}\right) \tag{3.21}$$

This method however does not respect nor recognize joint limits. Therefore a dynamic damping factor based on the joint limits was used [3].

$$\mathbf{J}^{\dagger} = \left(\mathbf{J}^T\mathbf{J} + \boldsymbol{\lambda}^2\right)^{-1}\mathbf{J}^T \tag{3.22}$$

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & \lambda_n \end{bmatrix} \tag{3.23}$$

$$\lambda_n = c\left(\frac{2q_n - q_{n_{max}} - q_{n_{min}}}{q_{n_{max}} - q_{n_{min}}}\right)^p \tag{3.24}$$

where c and p are positive, even numbers, q_{min} and q_{max} are the lower and upper joint limits respectively. The values of c and p are tuned by the user based on how close to the joint limits the manipulator is allowed to get. The value of p changes how "cautious" the algorithm is in approaching joint limits as shown in Fig 3.4 which was adopted from [3]. A high value of p places more trust in the joint position and allows it to go closer to the joint limit, whereas a low value of p is a more cautious

approach and starts increasing the damping factor λ_n while the joint is still away from the limit.

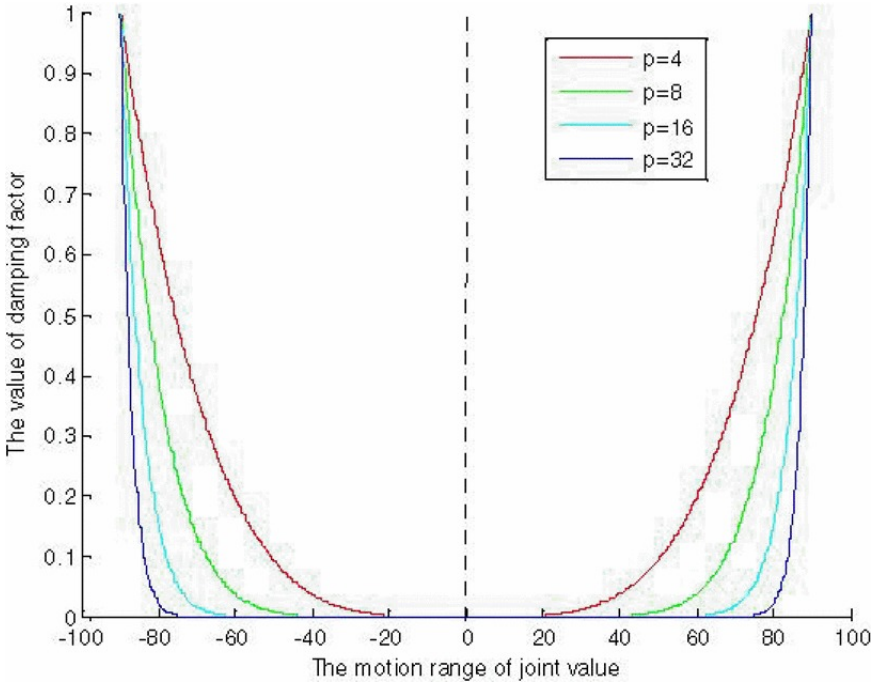

Figure 3.4: Behavior of λ_n as joint limits are approached

3.3.4 Computing the Jacobian and Jacobian Time Derivative

As seen in Eq. 3.16 and Eq. 3.21 both, the Jacobian and its time derivative are necessary. The forward kinematics function is nonlinear so solving for an analytical equation for the Jacobian is impractical. As a result, the Jacobian and its time derivative need to be computed numerically using the forward difference method. Every joint angle is perturbed by a small number h and the Jacobian was calculated according to Eq. 3.26.

$$\mathbf{J}(\mathbf{q}) = \frac{\partial f(\boldsymbol{q})}{\partial \boldsymbol{q}} \tag{3.25}$$

$$\frac{\partial f(q_1, \cdots, q_i, \cdots, q_n)}{\partial q_i} \approx \frac{f(q_1, \cdots, q_i + h, \cdots, q_n) - f(q_1, \cdots, q_i, \cdots, q_n)}{h} \quad (3.26)$$

Calculating the time-derivative is more computationally intensive.

$$\begin{aligned} \dot{\mathbf{J}}(\mathbf{q}) &= \frac{d}{dt}\mathbf{J}(\mathbf{q}) \\ &= \frac{\partial \mathbf{J}(\mathbf{q})}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial t} \\ &= \frac{\partial \mathbf{J}(\mathbf{q})}{\partial \mathbf{q}}\dot{\mathbf{q}} \end{aligned} \quad (3.27)$$

$$\frac{\partial \mathbf{J}(q_1, \cdots, q_i, \cdots, q_n)}{\partial q_i} \approx \frac{\mathbf{J}(q_1, \cdots, q_i + h, \cdots, q_n) - \mathbf{J}(q_1, \cdots, q_i, \cdots, q_n)}{h} \quad (3.28)$$

where $\frac{\partial \mathbf{J}(\mathbf{q})}{\partial \mathbf{q}} \in \mathcal{R}^{6\times l\times l},\ \dot{\mathbf{q}} \in \mathcal{R}^{l\times 1},\ \dot{\mathbf{J}}(\mathbf{q}) \in \mathcal{R}^{6\times l}$ where l is the number of joints.

The Jacobian was first kept constant for the entire period of integration, however, this resulted in oscillations in the trajectory when the joint reached its limits. Updating the Jacobian during integration significantly damped the oscillations.

3.4 MATLAB Simulation

The system was simulated in MATLAB to verify the equations of motion as well as check system performance for motor sizing for the experimental set up. The manipulator was fixed to a surface and there was no force acting on the end effector. All initial conditions were zero. An end-effector trajectory in the operational-space was specified and the tracking performance was studied.

The reference signal was in the form of desired position, velocity and acceleration of the end-effector. An Inverse Dynamics controller was used for this simulation.

Table 3.2: Joint Velocity and Torque Limits

Joint	Velocity limits (rad/s)	Torque limits (Nm)
1	3.08	1.2
2	1.15	2.82
3	1.15	2.82
4	10.47	0.25
5	10.47	0.25

The equations of motion were given by

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q})g \tag{3.29}$$

The control input is proposed as

$$\boldsymbol{\tau_i} = \mathbf{M}(\mathbf{q})\boldsymbol{\tau_i'} + \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q})g \tag{3.30}$$

$$\therefore \boldsymbol{\tau_i'} = \ddot{\mathbf{q}}_\mathbf{d} \tag{3.31}$$

Therefore, the desired input torque was calculated using Eq. 3.21 and 3.30. This simulation was performed with the assumption that both joint velocity and joint position feedback was available. The velocity and torque limits for the motors are given in 3.2

3.4.1 2D Trajectory to a Single Point

The first simulation was done for motion to a single point $\mathbf{x_f}$ from $\mathbf{x_0}$ where the position is described in millimeters and the orientation is in radians.

$$\mathbf{x_0} = \begin{bmatrix} 646.50 \\ 0 \\ 276.50 \\ 0 \\ 0 \\ 1.571 \end{bmatrix} \quad \mathbf{x_f} = \begin{bmatrix} 616.70 \\ 0 \\ 432.96 \\ 0 \\ 0 \\ 1.571 \end{bmatrix} \tag{3.32}$$

The results of the simulation in cartesian-space are seen in Fig. 3.5. There is negligible deviation observed in the path.

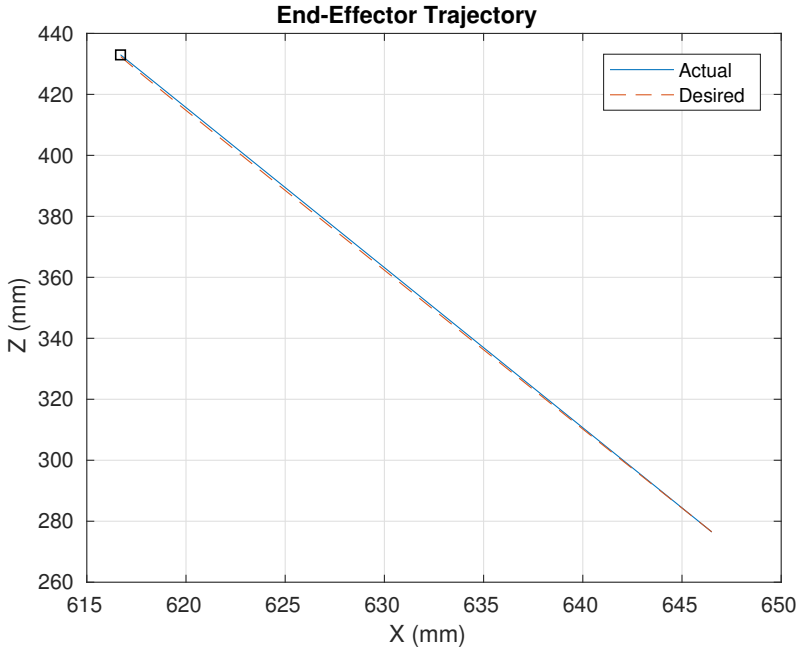

Figure 3.5: Simulated End-Effector Trajectory, Desired and Simulated, 2D Single Point MATLAB

Fig 3.6 shows the joint-space trajectories for the manipulator joints. The end-effector trajectory is defined in the X-Z plane but some movement,however negligible, is seen in joints 1 and 4 which control movement in the X-Y plane, or about the Z-axis, due to the inverse kinematics algorithm. However, it can be observed that the motion due q_1 is reciprocated by q_4. The magnitudes of the joint angles are diffferent, but the trend is similar. This is because joint 4 is further down the chain and has to move further to compensate for the movement due joint 1.

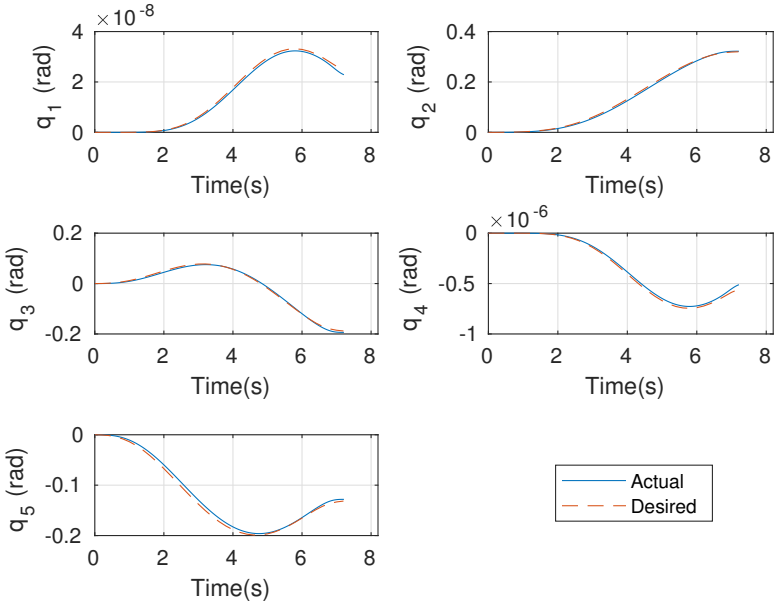

Figure 3.6: Simulated Joint Angle Trajectory, Desired and Simulated, 2D Single Point MATLAB

The end effector follows a minimum-jerk trajectory, so the velocity profile is expected to be parabolic and finish at zero because the final condition is specified to be zero. This is seen in Fig 3.7 where v_1 through v_6 are the velocities of the pose variables. Significant movement is only seen in the x and z positions of the end

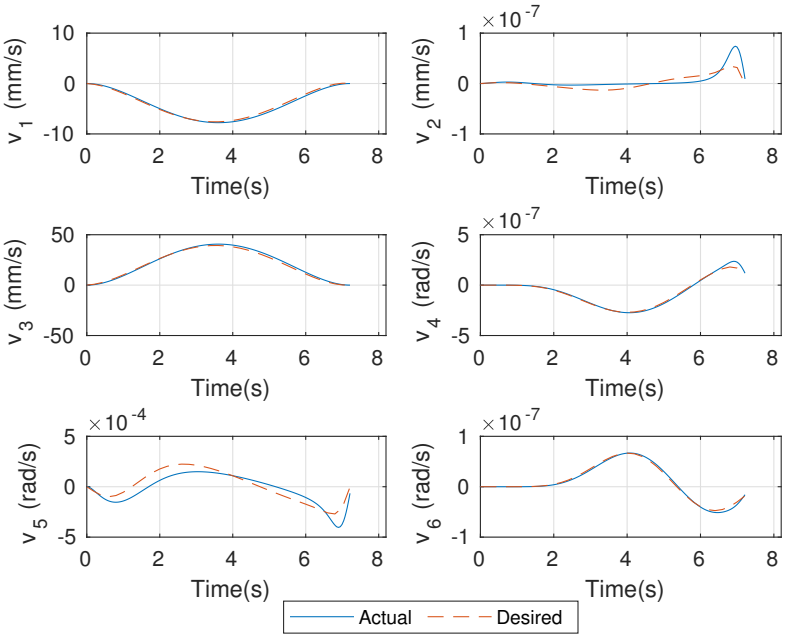

Figure 3.7: End-Effector Cartesian Space Velocities,
2D Single Point MATLAB

effector, while the initial and final orientations are the same. Therefore, the velocity profiles for the x and z coordinates are the most prominent; and a clear parabolic velocity profile can be observed with the end-effector coming to rest at the end of the simulation. Similar trends are observed in the joint velocities as observed in Fig 3.8. Movement is observed in joint 5 to maintain the orientation of the end effector.

The joint torques required to complete the trajectory are shown in Fig. 3.9. All of these torques are within the torque limits for their respective actuators.

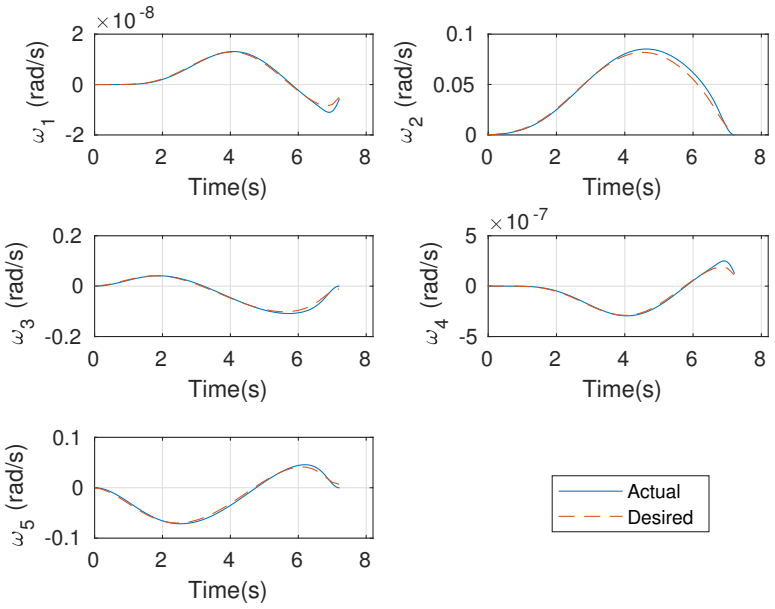

Figure 3.8: Simulated Joint Velocity History, Desired and Simulated, 2D Single Point MATLAB

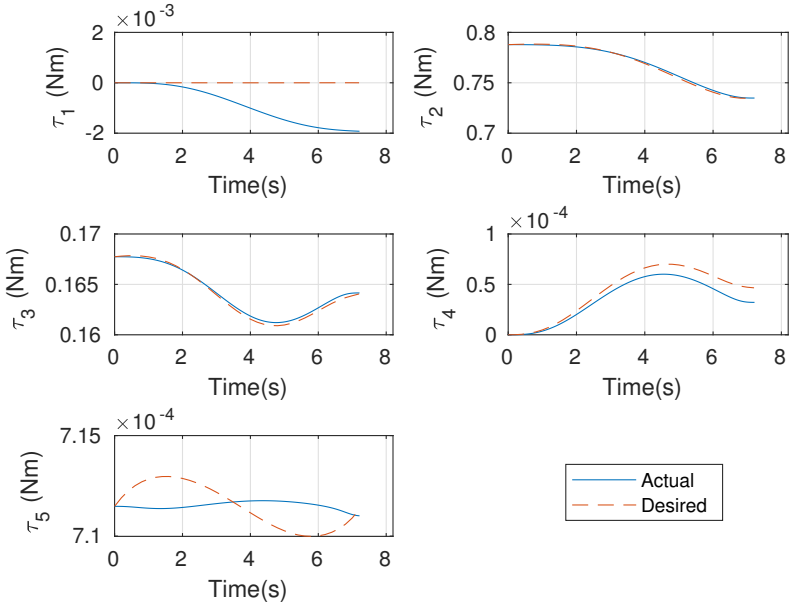

Figure 3.9: Simulated Joint Torque History, Desired and Simulated, 2D Single Point MATLAB

3.4.2 3D Trajectory to a Single Point

The simulation was done for motion to a single point $\mathbf{x_f}$ to $\mathbf{x_0}$ such that

$$\mathbf{x_0} = \begin{bmatrix} 646.50 \\ 0 \\ 276.50 \\ 0 \\ 0 \\ 1.571 \end{bmatrix} \quad \mathbf{x_f} = \begin{bmatrix} 534.07 \\ 308.35 \\ 432.96 \\ 0.5236 \\ 0 \\ 1.571 \end{bmatrix} \tag{3.33}$$

The results of the simulation are seen in the end-effector trajectory in cartesian space as shown in Fig. 3.10 and negligible deviation is seen in the path.

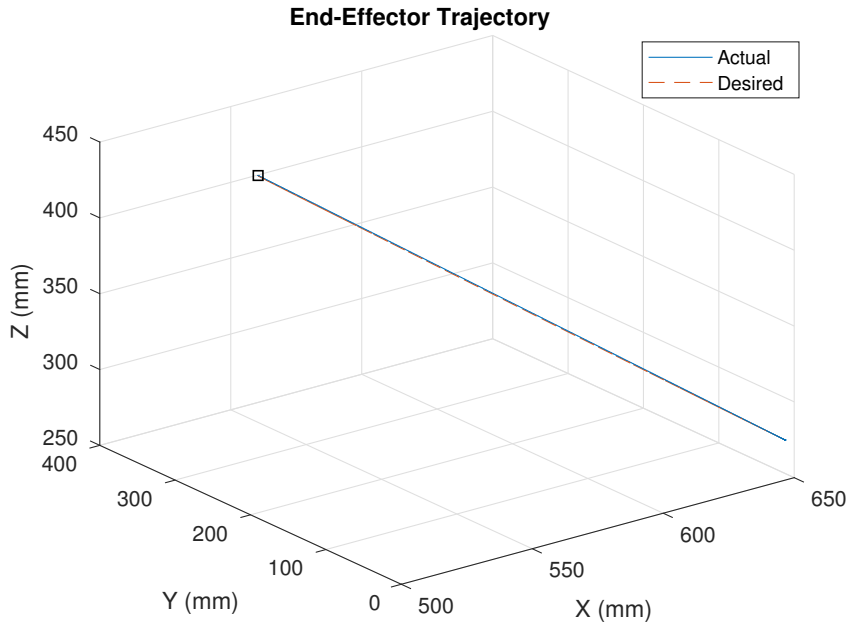

Figure 3.10: Simulated End-Effector Trajectory, Desired and Simulated, 3D Single Point MATLAB

The behavior of the joint angles is seen in Fig 3.11. The joint angles follow the commanded trajectory with little error. The joint velocity profiles are similar to the pose velocity profile in that they have the parabolic shape necessary to be minimum jerk. They both return to zero, which is the specified boundary condition so it can be concluded that the algorithm is working as intended. Significant movement is intended in all joints for this trajectory, therefore the small fluctations seen during the 2D motion in the X-Z plane are not apparent anymore.

The final and initial value for Ψ are the same but there is no constraint on that orientation during the path. It can be seen in Fig. 3.12 that there is movement in the orientation vector, but the final position is still achieved. This is because the end-effector is following a cartesian-space trajectory and the manipulator joint angles are adjusted to ensure that the straight line path is followed.

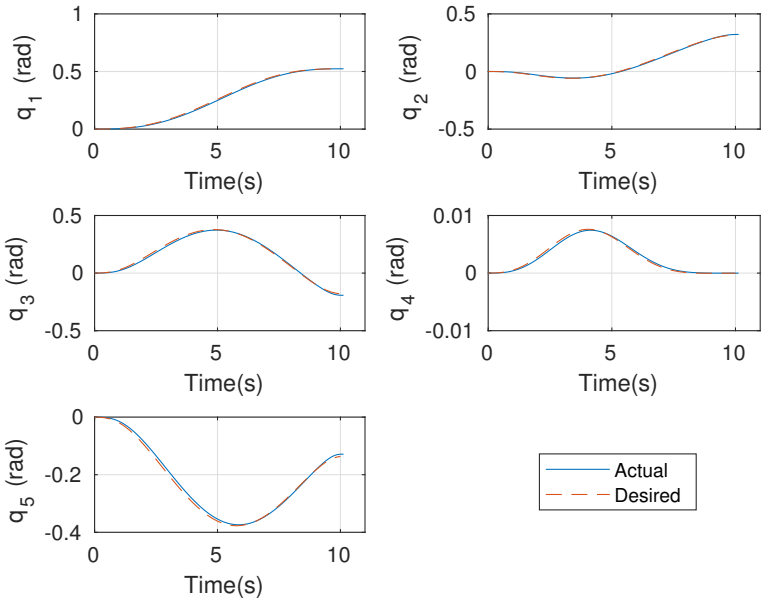

Figure 3.11: Simulated Joint Angle Trajectory, Desired and Simulated, 3D Single Point MATLAB

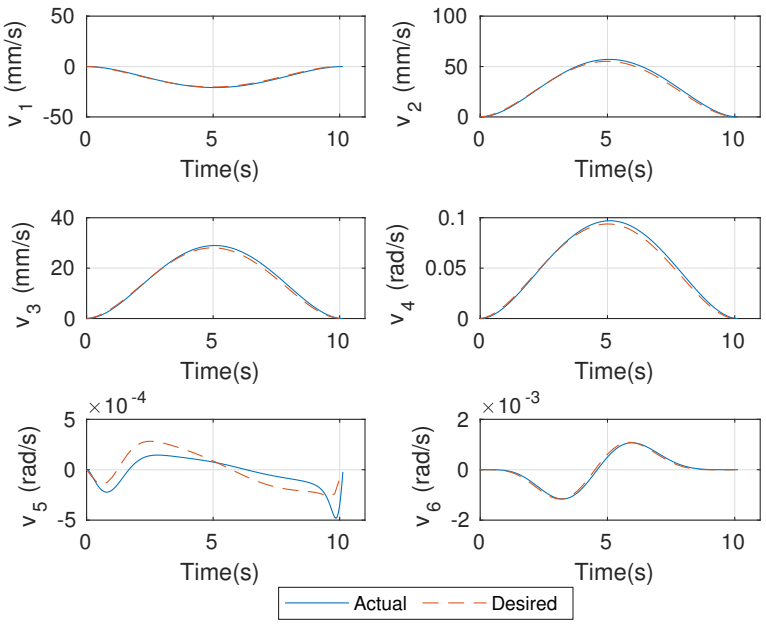

Figure 3.12: End-Effector Cartesian Space Velocities, 3D Single Point MATLAB

The joint torques necessary to complete this motion are within hardware limits as seen in 3.14

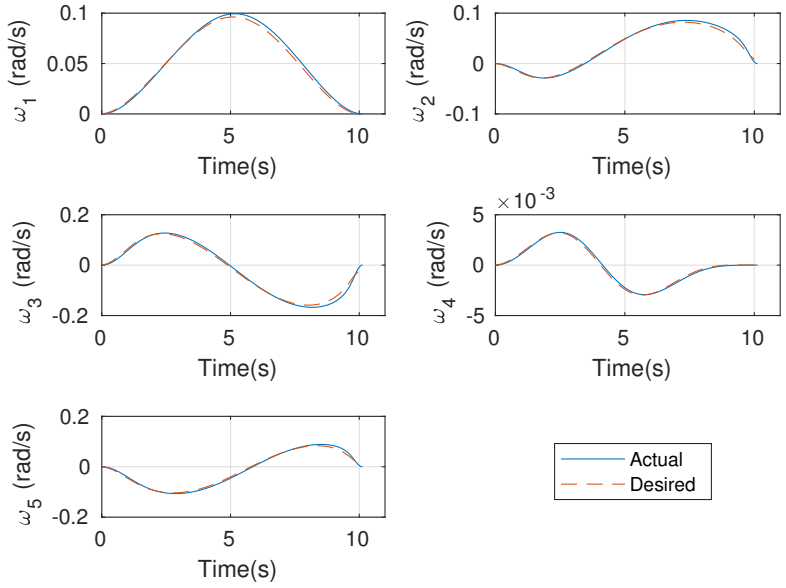

Figure 3.13: Simulated Joint Velocity History, Desired and Simulated, 3D Single Point MATLAB

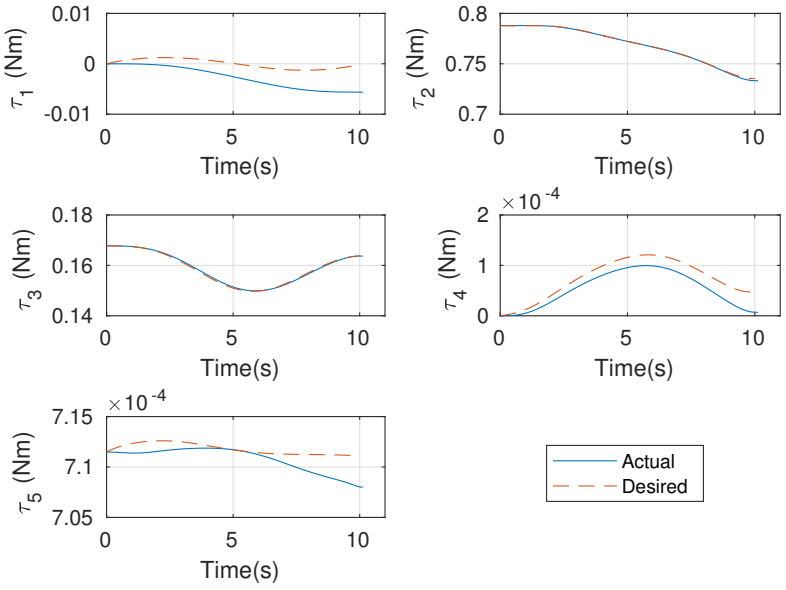

Figure 3.14: Simulated Joint Torque History, Desired and Simulated, 3D Single Point MATLAB

3.4.3 2D Multi-Point Trajectory

The simulation was then performed for a multi-point trajectory with the waypoints specified in Table 3.3. The trajectory is a loop and starts at the same $\mathbf{x_0}$ as defined for the previous simulation, and passes through the waypoints in Table 3.3 which returns it to the starting position as shown in Fig. 3.15. The desired trajectory is tracked with minimal deviation and the waypoints are traversed accurately.

Table 3.3: Waypoints for 2D Multi-point Trajectory

x	y	z	Φ	Θ	Ψ
634.4582	0	366.6756	0	-0.0873	1.5708
616.6964	0	432.9637	0	-0.0873	1.5708
621.2470	0	374.2493	0	-0.2618	1.5708
646.5000	0	276.5000	0	0	1.5708

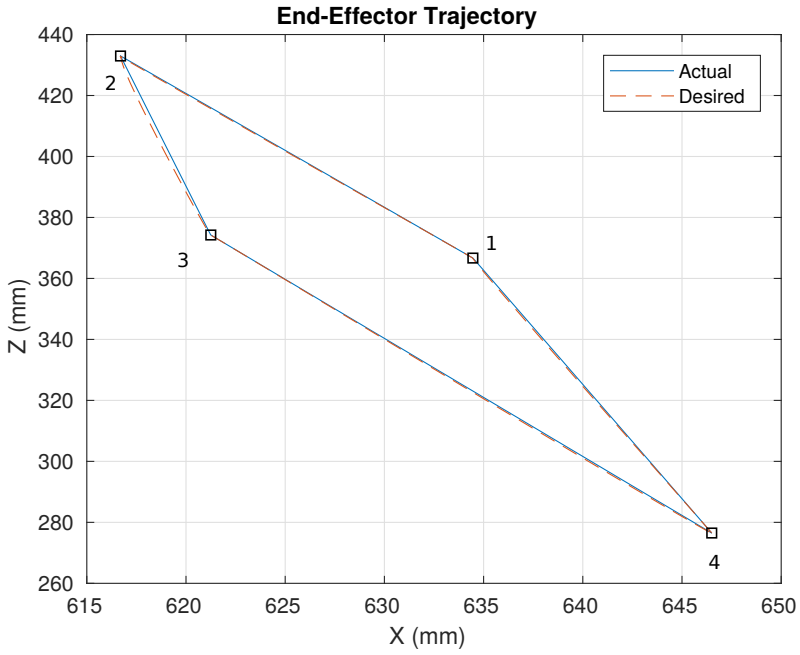

Figure 3.15: Simulated End-Effector Trajectory, Desired and Simulated, 2D Multi-point MATLAB

Similar to Fig 3.6, Fig. 3.16 shows that the motion of joint 1 is reciprocated by joint 4. Figures 3.17 and 3.18 show that the total trajectory is comprised of multiple parabolic trajectories. This is indicative that the motion of the end effector has been minimum-jerk.

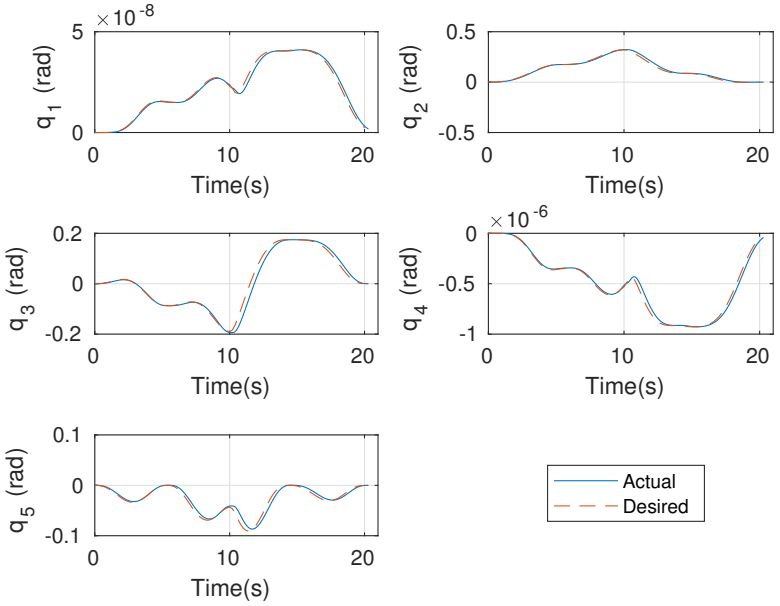

Figure 3.16: Simulated Joint Angle Trajectory, Desired and Simulated, 2D Multi-point MATLAB

The torque trajectories are smooth for the joints responsible for rotation about the global Y-axis, i.e joints 2,3,5, because these joints are holding up the weight of the manipulator and they need a constant torque input. However, the remaining two joints, 2 and 4, only need a torque input during the motion of the manipulator because the load acts along the axis of rotation so the actuators do not have to put in any effort to counteract it.

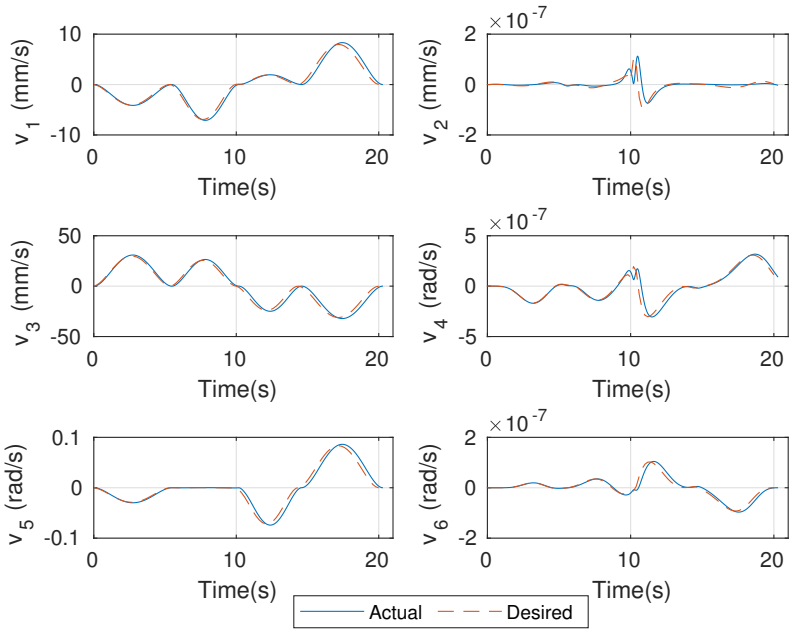

Figure 3.17: End-Effector Cartesian Space Velocities, 2D Multipt MATLAB

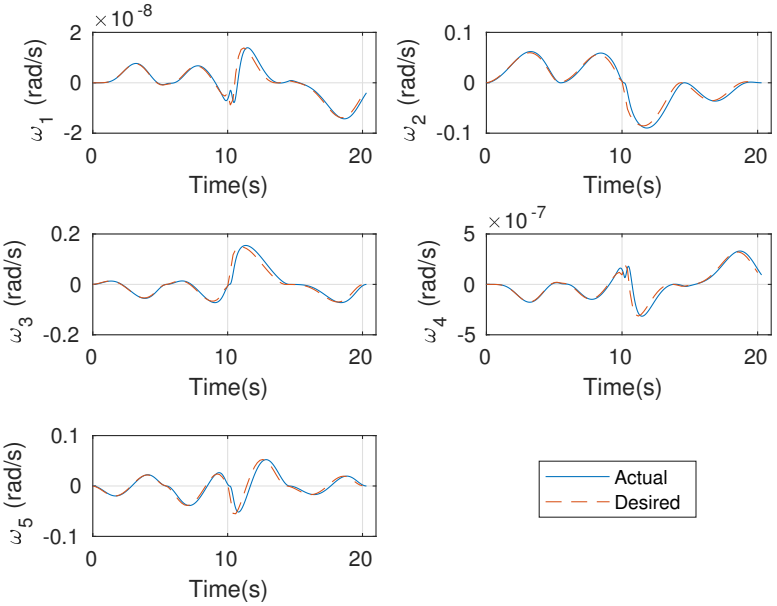

Figure 3.18: Simulated Joint Velocity History, Desired and Simulated, 2D Multi-point MATLAB

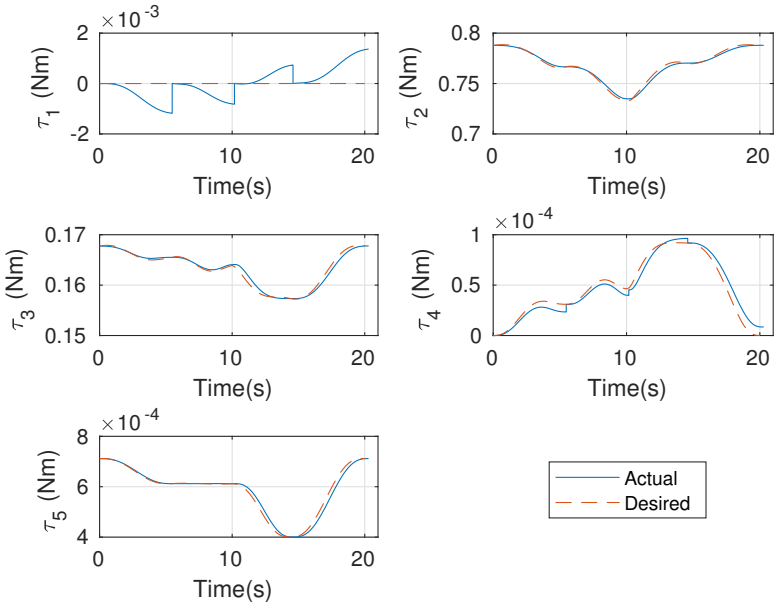

Figure 3.19: Simulated Joint Torque History, Desired and Simulated, 2D Multi-point MATLAB

3.4.4 3D Multi-Point Trajectory

This simulation was performed with the waypoints shown in Table 3.4. The starting pose, $\mathbf{x_0}$ is the same as for all previous simulations. The loop starts at $\mathbf{x_0}$, passes through the points shown in Table 3.4 and returns to the starting position as shown in Fig 3.20. The desired trajectory was completed with minimal deviation. and all the waypoints were traversed accurately.

Table 3.4: Waypoints for 3D Multi-point Trajectory

x	y	z	Φ	Θ	Ψ
634.46	0	366.68	0	-0.087266	1.5708
494.29	179.91	559.96	0.34907	-0.69813	1.5708
414.38	493.83	322.61	0.87266	0.087266	1.5708
646.5	0	276.5	0	0	1.5708

Figures 3.22 and 3.23 show us that the full trajectory is formed from multiple parabolic trajectories.

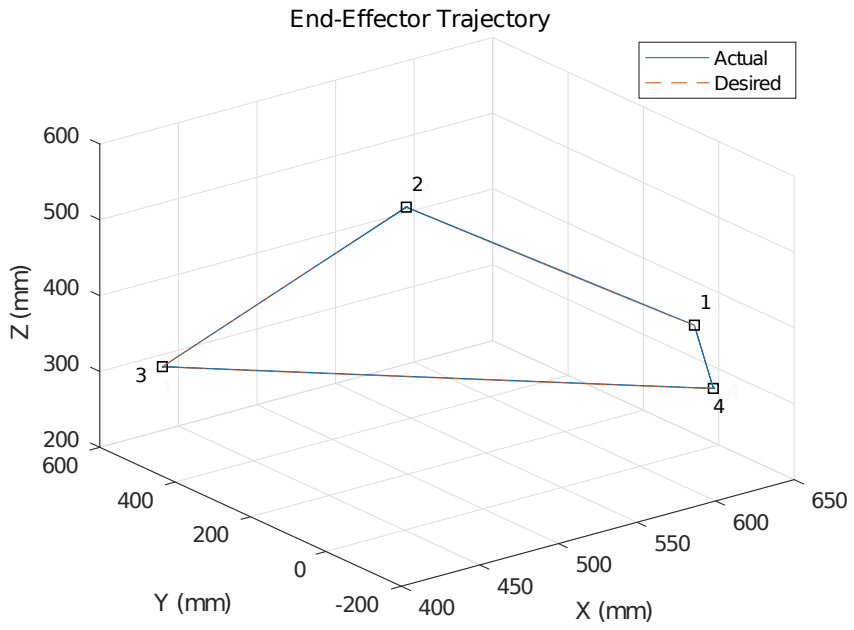

Figure 3.20: Simulated End-Effector Trajectory, Desired and Simulated, 3D Multi-point MATLAB

The torque trajectories in Fig 3.24 are smooth and the torques for joints 1 are observed to drop to zero once the trajectory segment is completed. This is because the weight of the manipulator is acting along the axis of the joint, so the actuator doesn't have to put in any effort to maintain the position.

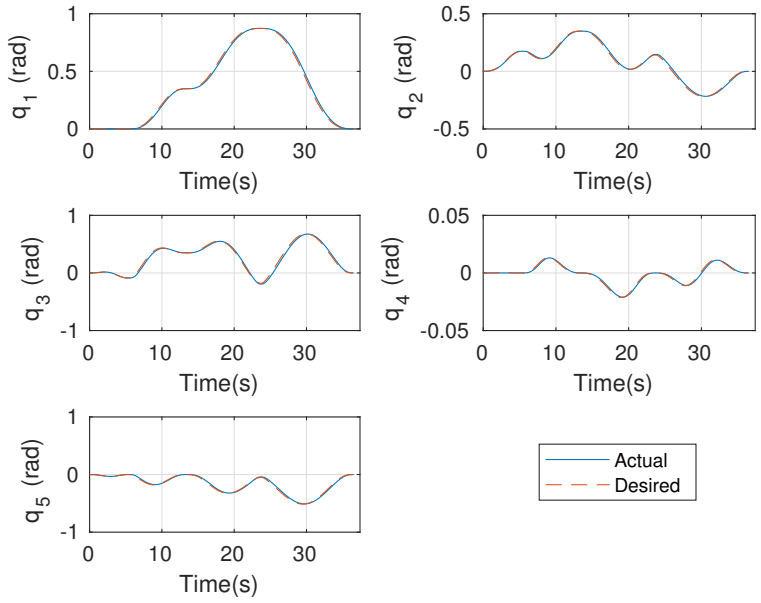

Figure 3.21: Simulated Joint Angle Trajectory, Desired and Simulated, 3D Multi-point MATLAB

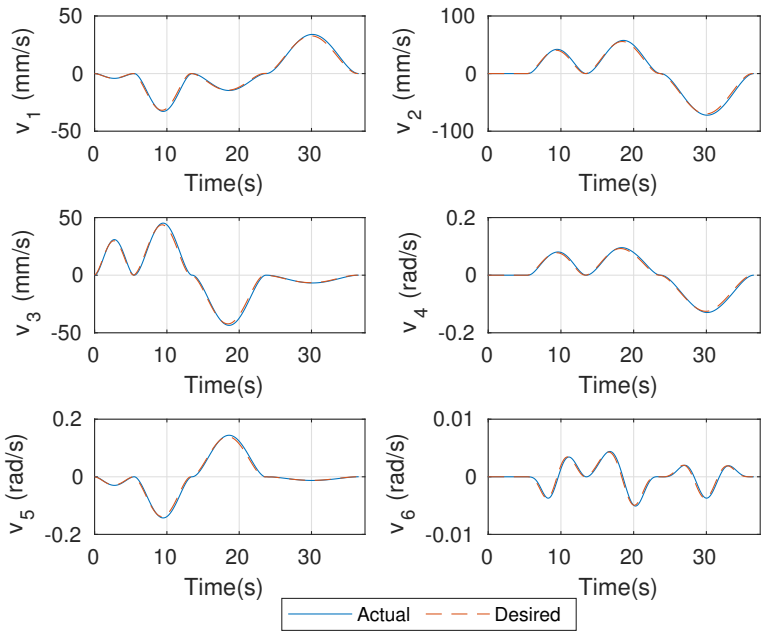

Figure 3.22: End-Effector Cartesian Space Velocities, 3D Multi-point MATLAB

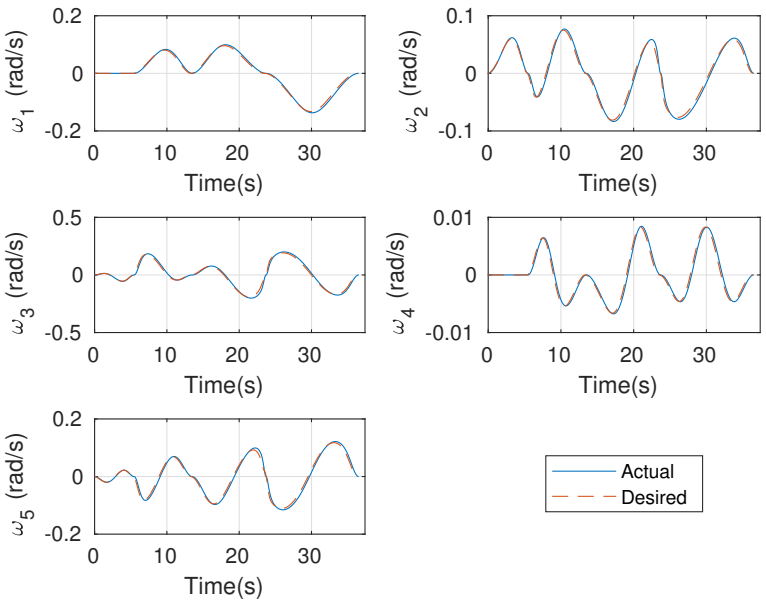

Figure 3.23: Simulated Joint Velocity History, Desired and Simulated, 3D Multi-point MATLAB

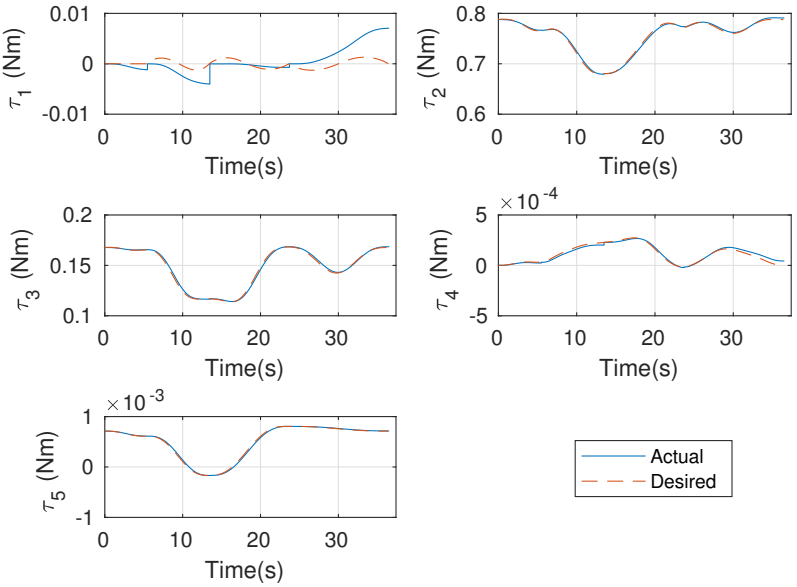

Figure 3.24: Simulated Joint Torque History, Desired and Simulated, 3D Multi-point MATLAB

3.5 ROS - Gazebo Simulation

3.5.1 Introduction to ROS and Gazebo

The Robot Operating System (ROS) is an open-source framework for writing software for robots [15]. It is a collection of libraries and tools that handle the low level communication required for robot operation thereby providing a high level of modularity to the system. It brings a level of standardization to the system which leads to general robustness making colaboration and switching code easy.

Gazebo is the ROS-compatible physics simulator. The user has the ability to import a CAD model, attach actuators, and simulate the system very closely to its behavior in real life. This allows the user to implement and test a variety of controllers, designs and hardware options without disturbing the system hardware, only implementing new software once the simulation results are satisfactory. There are several plugins available for interfacing ROS and Gazebo, allowing the user more time to focus on the robot software. Further details are provided in Appendix A.

3.5.2 Simulation Setup

A CAD model was imported into the ROS-Gazebo environment as a Unified Robot Description File (URDF) using the SolidWorks URDF Exporter. All the mass properties, kinematic data such as joint angle limits, link lengths, joint velocity limits were encoded into this file along with refrence frame information. ROS interfaces with Gazebo models using a package called "ros_control" that provides turn-key actuator and control interfaces for the joints defined in the URDF file [16]. Using this, each joint was attached a motor with a position controller running at 50 Hz and the main inverse kinematics node was run at 10 Hz. The ros_control package provided a topic for a PID controller for each joint. The trajectory was generated at with a time step corresponding to the frequency of the inverse kinematics node, stored and published

to the corresponding joint controllers.

Due to the hardware requirements, torque-control could not be used on some of the motors. As a result position control was used for all motors. Therefore, using the Inverse Dynamics method used for MATLAB simulations was not possible. The control scheme used for the ROS simulation and then extended to the model was to generate the trajectory in operational-space, use first-order inverse kinematics using a Fourth Order Runge-Kutta integrator (RK4) to map it to the joint-space and then use PID controllers for position control. This method was much faster than the MATLAB simulation and could work with the controller frequencies stated previously.

3.5.3 2D Trajectory to a Single Point

This ROS simulation was performed with the same initial and final points as the MATLAB simulation shown in Eq. 3.32. The joint-space trajectory was pre-generated and the desired joint angles were passed to the controller when the error reached below a threshold of 0.02 radians or 1.5 degrees. This threshold was decided based on the performance of the second order filter used to filter the potentiometer values for angle feedback in the manipulator hardware discussed in Chapter 4. The resulting end-effector trajectory is seen in Fig. 3.25. The errors in the final position is -1.47, -0.01 and 4.38 mm in the X,Y and Z directions respectively.

The trajectory tracking of each individual joint angle is seen in Fig 3.26. Accurate tracking is observed in joints with significant movement and the tracking errors in the remaining two joints are in the order of 10^{-4} radians so they are negligible. The correlation between end-effector position and joint angle errors can be observed in Fig 3.26 and Fig 3.27. Joint 1 rotates about the Z-axis in the inertial reference frame so it is responsible for the movement of the end-effector in the global Y-axis, therefore there is movement in the end effector in the Y direction corresponding to

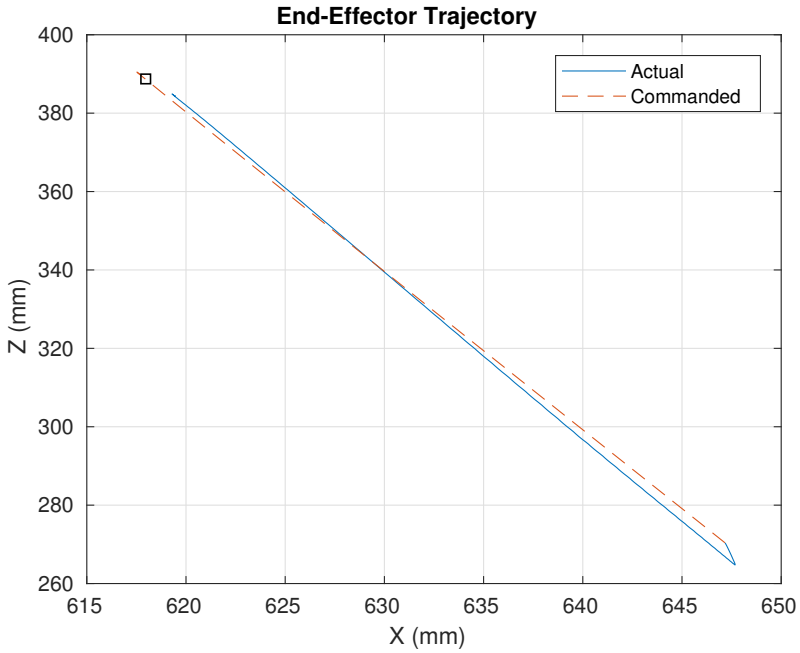

Figure 3.25: Simulated End-Effector Trajectory, Desired and Simulated, 2D Single Point ROS

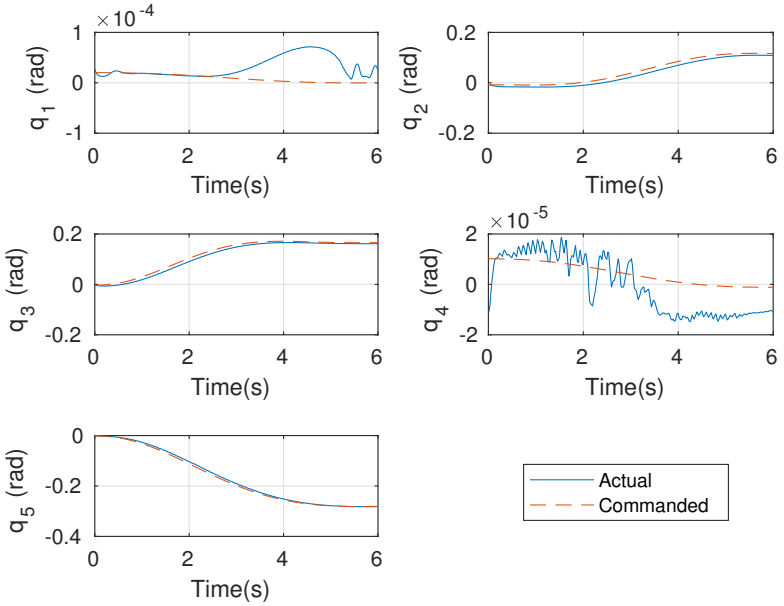

Figure 3.26: Manipulator Joint Angles, Desired and Simulated, 2D Single Point ROS

the error in Joint 1. This error and the error in the orientation in x_5 are responsible for the error in the end-effector position.

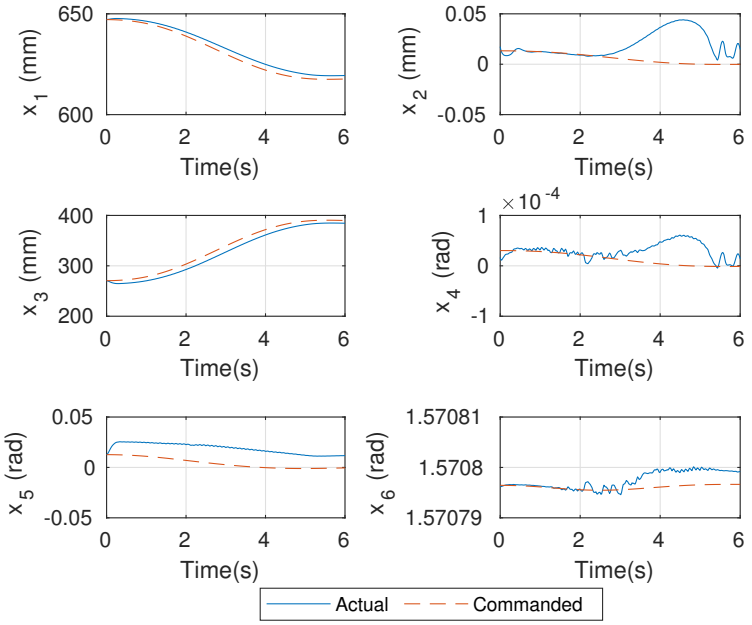

Figure 3.27: Manipulator Pose Variables, Desired and Simulated, 2D Single Point ROS

3.5.4 3D Trajectory to a Single Point

The initial and final points for this simulation are given in Eq. 3.33. The position errors were -1.7000 mm, -1.0000 mm, and 5.5000 mm in the X,Y and Z directions respectively. Therefore, it can be said that the trajectory tracking performance was acceptable as seen in Fig. 3.28. The performance of each individual joint can be seen in Fig. 3.11. The angles follow a different trajectory than that for the simulation in MATLAB. This can be attributed to the inverse kinematics solution as well as the dynamic damping factor used for the Damped Least-Squares inverse.

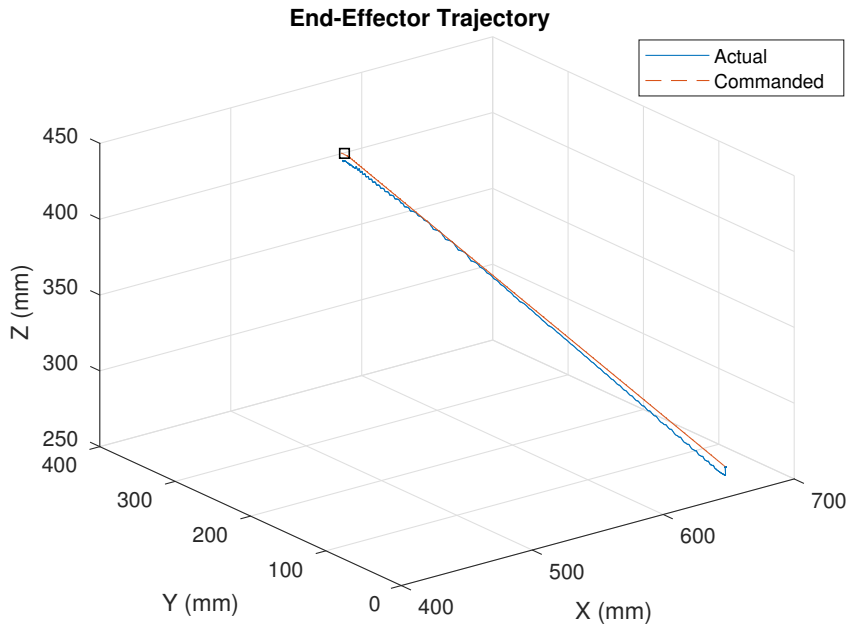

Figure 3.28: Simulated End-Effector Trajectory, Desired and Simulated, 3D Single Point ROS

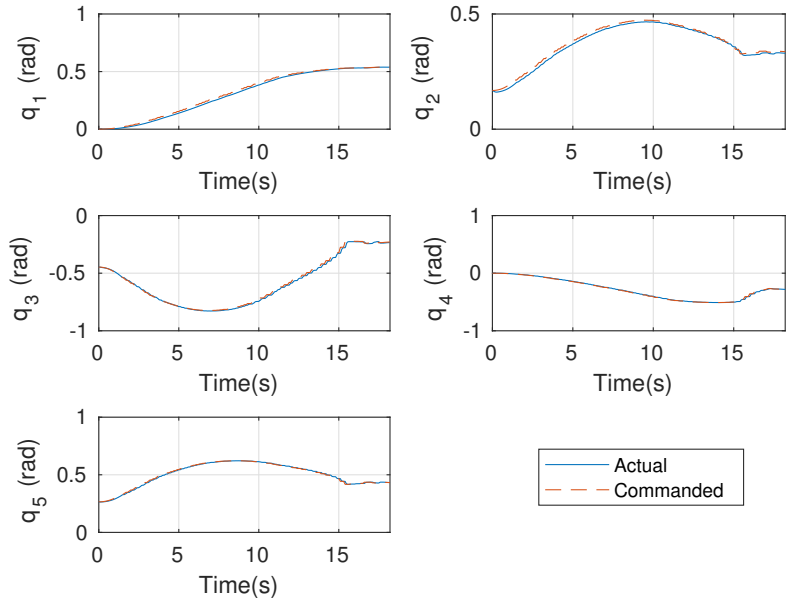

Figure 3.29: Manipulator Joint Angles, Desired and Simulated, 3D Single Point ROS

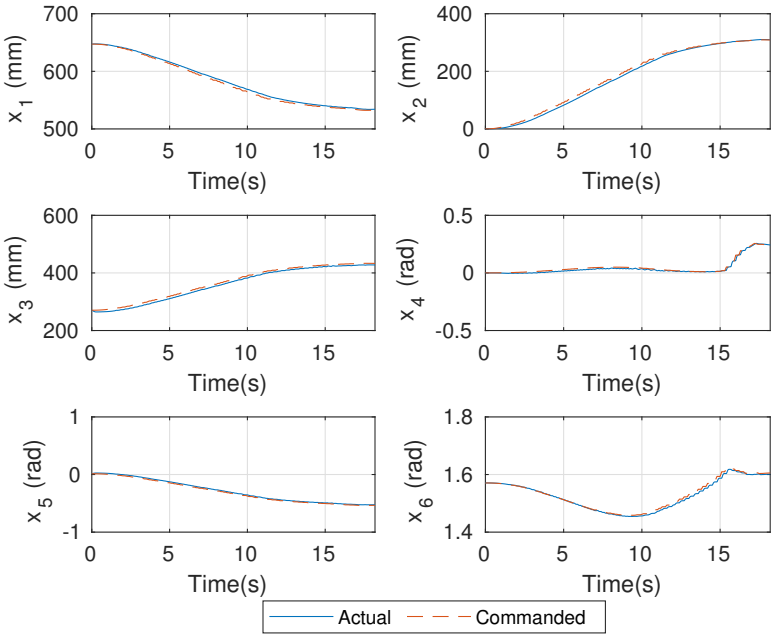

Figure 3.30: Manipulator Pose Variables, Desired and Simulated, 3D Single Point ROS

3.5.5 2D Multi-Point Trajectory

3.5.5.1 Small Loop

The previous simulation was extended to a trajectory with multiple points listed in Table 3.3. The trajectory tracking performance is shown in Fig 3.31. There is a minor trajectory tracking error in the second segment of the loop, however it is a minor deviation and the end effector reaches the waypoint. The errors in each indvidual joints can be observed in Fig 3.32 and their impact on the end-effector pose can be observed in Fig 3.33. There is minor movement in Joint 1 which is compensated by movement in Joint 4 as shown in Fig 3.32 however, as seen previously the trend of the motion is similar, but the magnitude of the movement is larger due to the location of the robot in the joint chain. The trajectory for each segment was pre-generated so the end-effector reached each waypoint, a cartesian space trajectory to the next point

was calculated, which was then mapped to the joint-space with inverse kinematics and then executed.

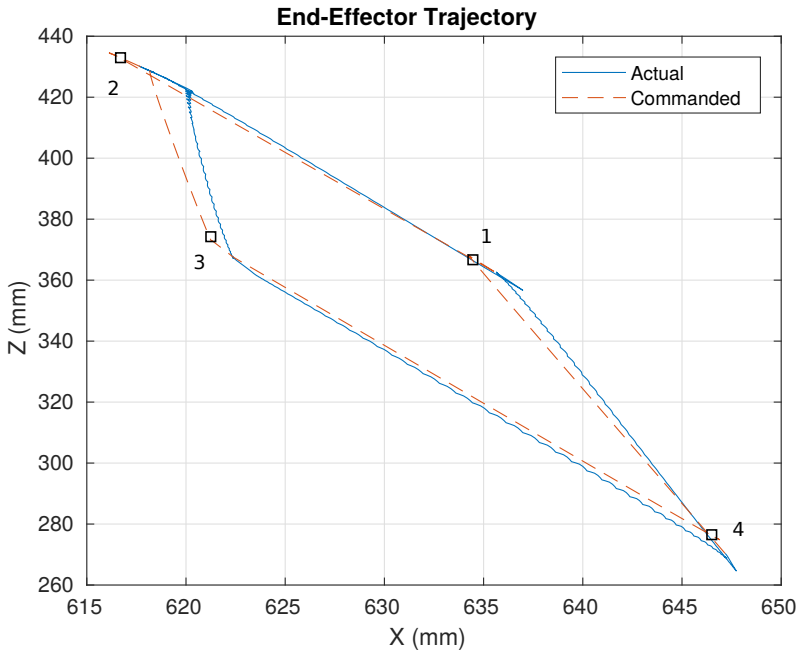

Figure 3.31: Simulated End-Effector Trajectory, Desired and Simulated, 2D Multi-Point ROS

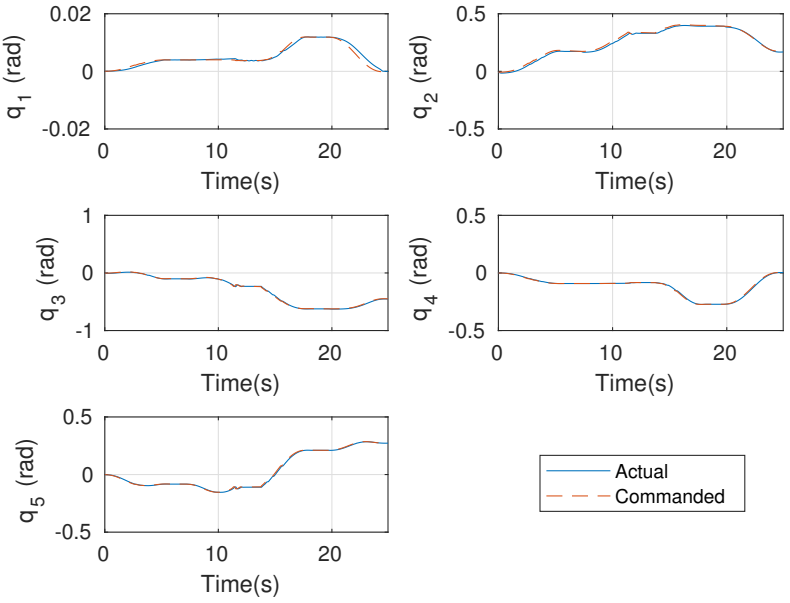

Figure 3.32: Manipulator Joint Angles, Desired and Simulated, 2D Multi-Point ROS

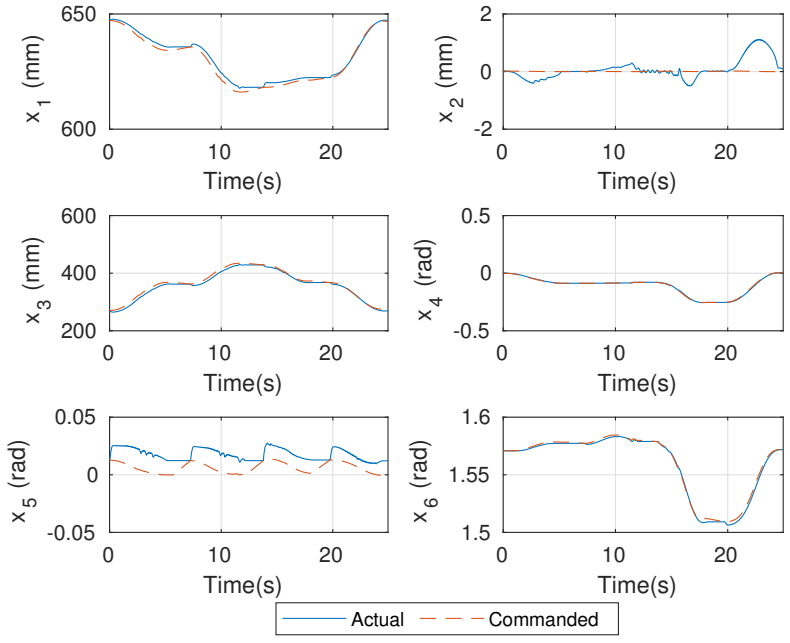

Figure 3.33: Manipulator Pose Variables, Desired and Simulated, 2D Multi-Point ROS

3.5.5.2 Large Loop

A trajectory around a loop with longer segments was also executed to test whether the errors increased over time. The initial position, x_0, was the same as for all previous simulations and the remaining waypoints are listed in Table 3.5. The trajectory tracking was very accurate despite the longer trajectories.

Table 3.5: Waypoints for Extended Multi-point Trajectory

x	y	z	Φ	Θ	Ψ
634.46	0	366.68	0	-0.087266	1.5708
426.59	0	712.51	0	-0.61087	1.5708
530.34	0	577.31	0	-0.61087	1.5708
646.5	0	276.5	0	0	1.5708

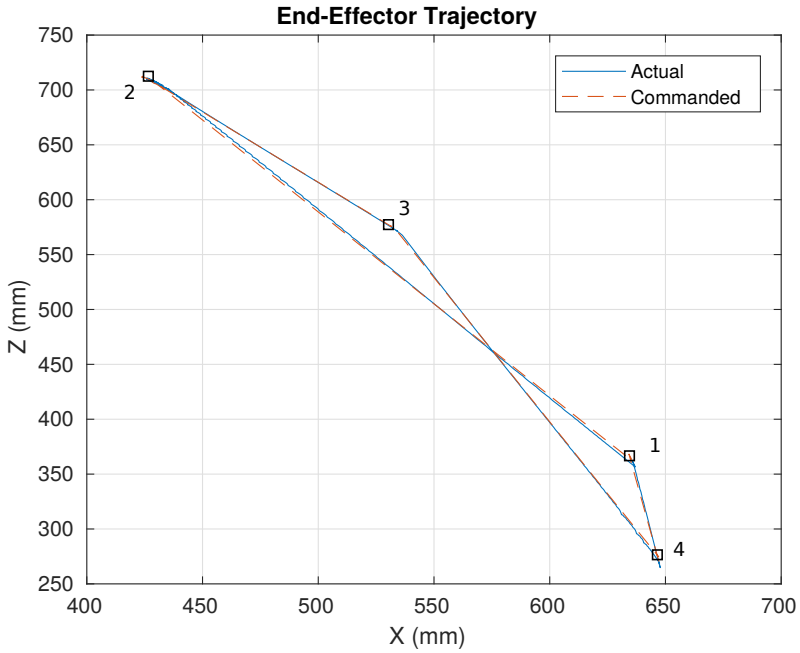

Figure 3.34: Simulated End-Effector Trajectory, Desired and Simulated, 2D Multi-Point ROS

This is a slight high-frequency error in the x_2 position, which corresponds to the Y-position of the end-effector. The error can be attributed to the errors in q_5.There is an error in x_5 due to q_5 as well. The overall accuracy of the trajectory tracking was acceptable over even longer distances, so it can be concluded that the controller and system performs consistently.

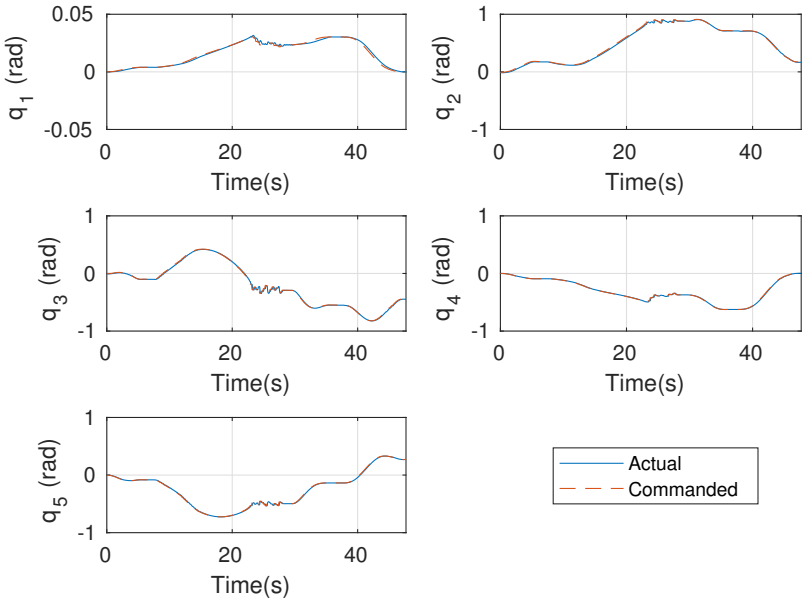

Figure 3.35: Manipulator Joint Angles, Desired and Simulated, 2D Multi-Point ROS

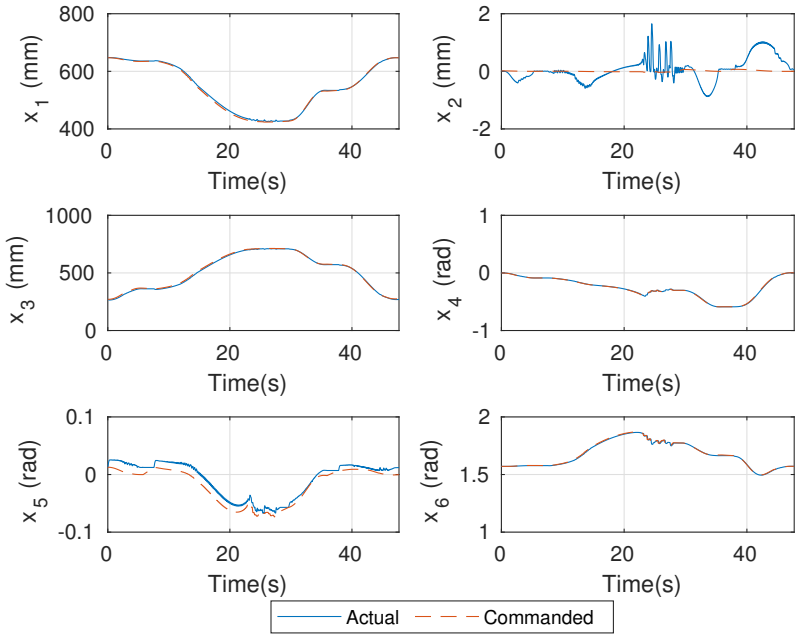

Figure 3.36: Manipulator Pose Variables, Desired and Simulated, 2D Multi-Point ROS

3.5.6 3D Multi-Point Trajectory

The initial position and waypoints for this simulation were the same as shown in Table 3.4. It is observed in Fig 3.37 that the manipulator follows the commanded trajectory with minimal error throughout the loop. A comparison between Fig 3.37 and Fig 3.38 shows that the commanded joint angle trajectory is achieved for a significant portion of the trajectory and the errors are minimal. Upon comparing Fig 3.20 and Fig 3.37 it is evident that the there are minimal deviations between the achieved end-effector trajectory. However, comparing the joint angle histories, it is observed that the joint angle configurations to achieve the same trajectories differ. This is because there isn't a unique solution to the problem and multiple configurations may yield the same end-effector pose.

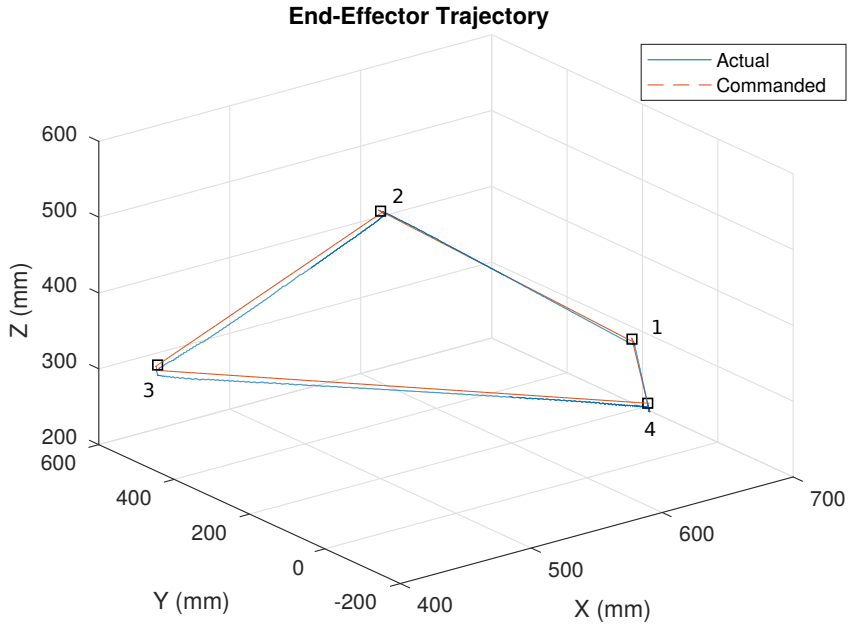

Figure 3.37: Simulated End-Effector Trajectory, Desired and Simulated, 3D Multi-Point ROS

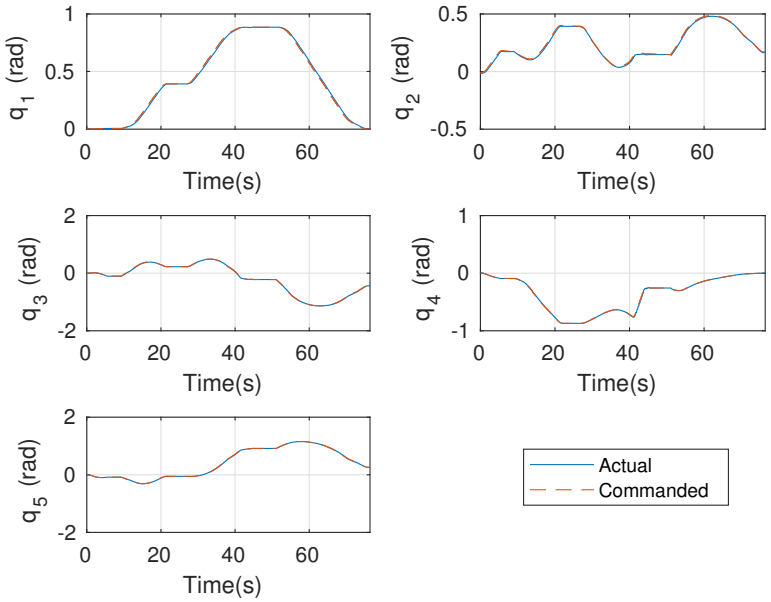

Figure 3.38: Manipulator Joint Angles, Desired and Simulated, 3D Multi-Point ROS

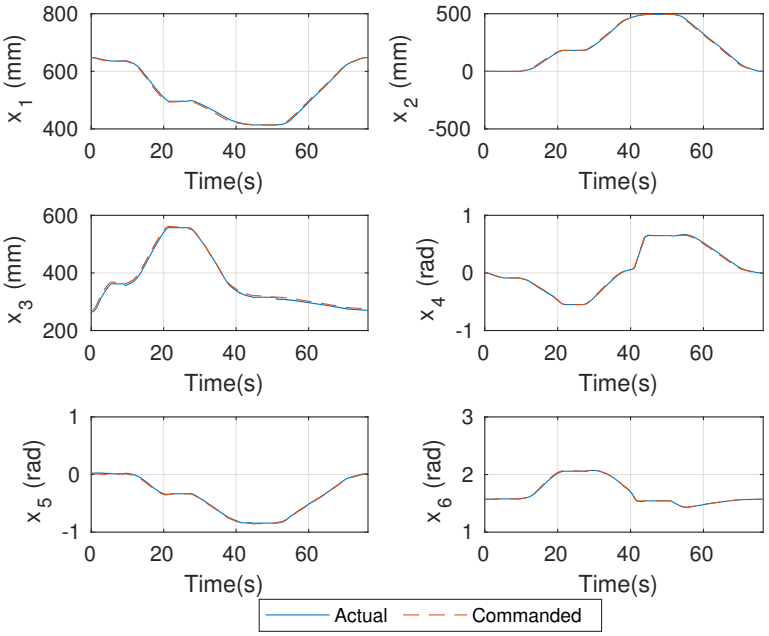

Figure 3.39: Manipulator Pose Variables, Desired and Simulated, 3D Multi-Point ROS

3.5.7 Comparison to MATLAB

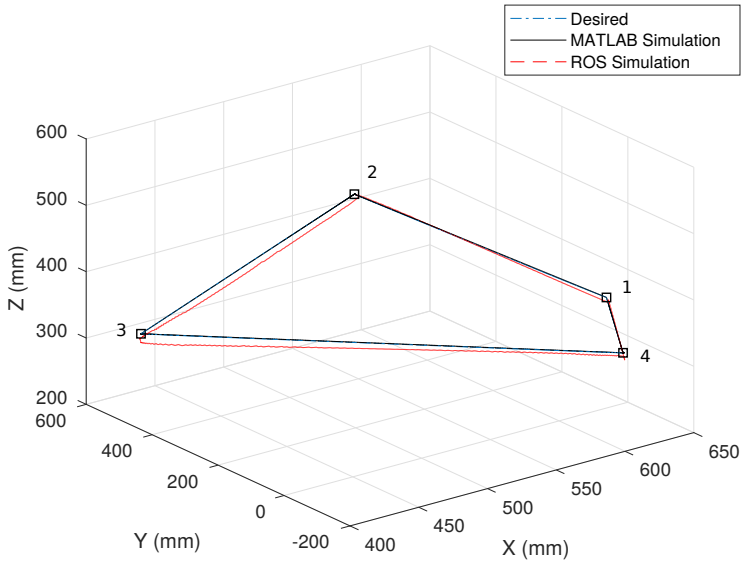

Figure 3.40: Simulated End-Effector Trajectory, ROS and MATLAB

Comparison of the end effector trajectories is the most appropriate metric to evaluate the validity and compatibility of the ROS ans MATLAB simulations as different joint angle trajectories still result in the desired end-effector trajectory. Figure 3.40 illustrates the comparison between the ROS simulation and MATLAB simulation the difference in the trajectory is minimal. The largest error seen is 7 mm at waypoint 3. Thus we can conclude that the ROS simulation and the MATLAB simulation are comparable with negligible errors. There is minor deviation between the two trajectories throughout the loop.

CHAPTER 4

EXPERIMENTAL SETUP

4.1 Construction

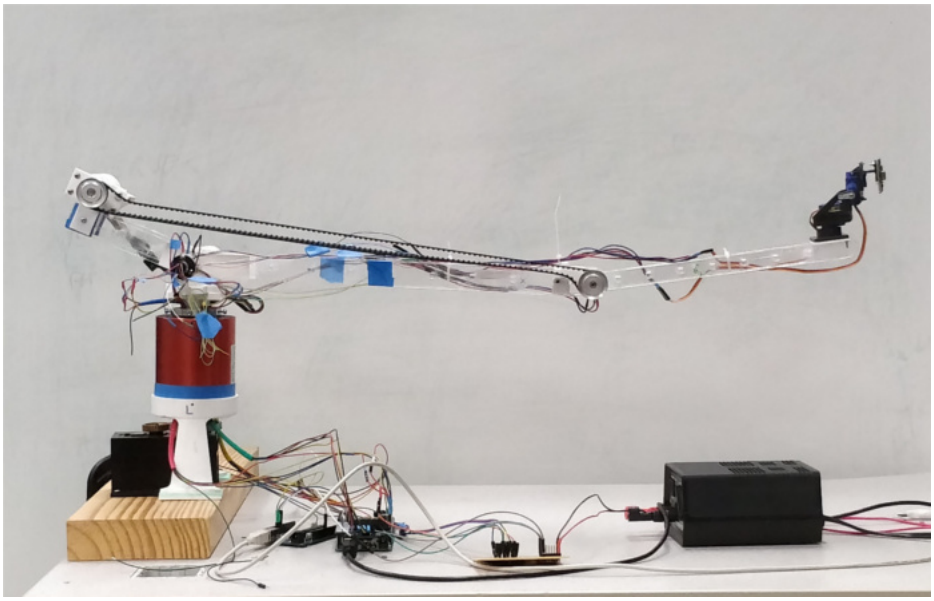

Figure 4.1: The 5 DOF Manipulator

The body of the manipulator is made of a combination of acrylic and ABS plastic. Link 1, all motor housings and sensor housings were fabricated out of ABS plastic using additive manufacturing processes. Links 2 and 3 were fabricated from laser cut acrylic with cross members for torsional rigidity. Links 4 and 5 make up the pan and tilt system to control the final two degrees of freedom. Details about the fabrication are included in reference [17]

Being a networked cyber-physical system the manipulator makes use of a myriad of hardware to achieve continous and reliable performance. The hardware setup of the manipulator is explained in the following sections.

4.2 Sensors and Motors

The manipulator has 5 degrees of freedom without any constraints so it has five actuators. The pan and tilt system was an off the shelf product. The motor for joint 1 is a servo gearbox chosen because it provides good structural support to the assembly as well as meets the space constraints. Joints 2 and 3 have high torque requirements so they are actuated using 399:1 gear ratio DC motors driven by an H-bridge motor driver based on the L298 chip. The entire system runs at 6VDC. The DC motors caused a ripple in power line with introduced jitter to the servos. This jitter was eliminated by placing bypass $680\mu F$ capacitors across the servo power terminals.

4.2.1 Sensing

The chosen DC motors came with an encoder, however, due to the high gear ratio of 399:1 the encoder output was nearly 28000 ticks per revolution of the output shaft. The Arduino is not fast enough to sense readings at that frequency as well as handle rest of the processing. Encoders also do not provide absolute position, so $10k\Omega$ potentiometers were used as position feedback for the DC motors. The potentiometers have a range of motion of 300 degrees so they are suitable for this application.

A second order low-pass filter was implemented to run at 50z with a cut-off frequency of 10Hz. The transfer function of the filter is given by Eq 4.1 where ω_c is

the cutoff frequency and ζ is the damping. The filter can be expressed in the state space format as shown in Eq. 4.2.

$$\frac{\omega_c^2}{s^2 + 2\zeta\omega + \omega_c^2} \tag{4.1}$$

$$\begin{bmatrix} \hat{\theta} \\ \dot{\hat{\theta}} \end{bmatrix} = \mathbf{A} \begin{bmatrix} \hat{\theta} \\ \dot{\hat{\theta}} \end{bmatrix} + \mathbf{B}\tilde{\theta} \tag{4.2}$$

$$A = \begin{bmatrix} 0 & 1 \\ -\omega_c^2 & -2\zeta\omega_c \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \omega_c^2 \end{bmatrix} \tag{4.3}$$

The data acquisition is discrete so the equations shown in Eq. 4.2 were discretized as follows [18]

$$\begin{bmatrix} \hat{\theta}_{k+1} \\ \dot{\hat{\theta}}_{k+1} \end{bmatrix} = \mathbf{\Phi} \begin{bmatrix} \hat{\theta}_k \\ \dot{\hat{\theta}}_k \end{bmatrix} + \mathbf{\Gamma}\tilde{\theta}_k \tag{4.4}$$

$$\mathbf{\Phi} = e^{\mathbf{A}\Delta t}, \quad \mathbf{\Gamma} = \mathbf{A}^{-1}\left(\mathbf{\Phi} - \mathbf{I}\right)\mathbf{B} \tag{4.5}$$

4.2.1.1 Filter Perfomance

A sample signal was with noise was filtered to test filter performance. The filter was designed with $\omega_c = 10$ Hz and $\zeta = 0.5$. The signal was y such that

$$y = \sin(10t) + v, \quad v \sim \mathcal{N}\left(0, 0.1^2\right) \tag{4.6}$$

The term v represents the noise which is zero-mean Gaussian white noise with a standard deviation of 0.1.

In a second order low-pass filter, the filtered signal lags behind the source signal. The lag is affected by the values of ω_c and ζ. The values of the cutoff frequency affect

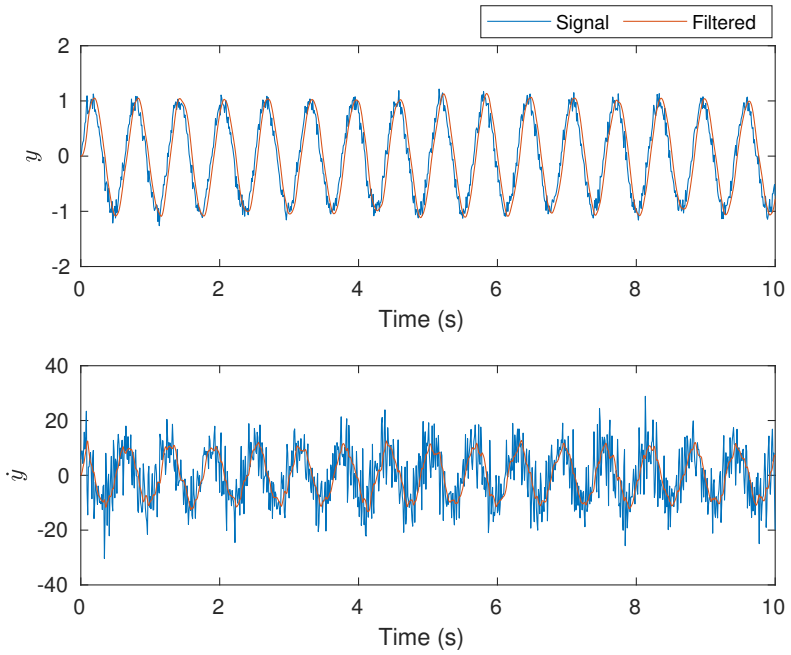

Figure 4.2: Filter Performance for sample signal

the lag the most, so it must be chosen carefully. The cutoff frequency and damping were chosen to minimize lag and overshoot.

4.2.2 Arduino

The motors are controlled using Pulse Width Modulation (PWM) which controls the the time for which the signal stays on and off. The PWM signal ranges from $1000\mu s$ to $2500\mu s$ and the percentage of the range for which the signal stays on is known as the duty cycle. Motor speed is controlled by modulating this duty cycle. The Arduino boards a very capable in this regard, so an Arduino Mega 2560 board was connected to the Raspberry Pi. The Arduino can be used as a ROS node using a package called "rosserial" [19]. Rosserial provides firmware to bring out-of-the-box ROS support to the Arduino ecosystem so all the sensor information is readily

available through ROS. The motors and the position sensors are connected to the Arduino which in turn passes the data on to the ROS network.

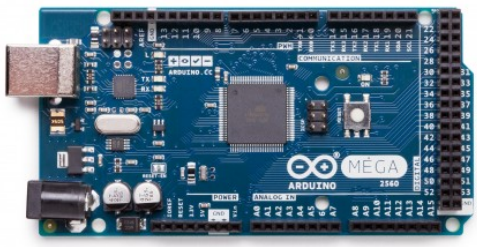

Figure 4.3: Arduino-Mega micro-controller used on the ASL-Gremlin-Rover
Source: https://store.arduino.cc/usa/arduino-mega-2560-rev3

4.2.3 Raspberry Pi

The manipulator is remotely controlled and a Raspberry Pi 3 installed with ROS Kinetic Kame is used to enable network connectivity required for ROS usage. The Raspberry Pi is a single board computer capable of running a full Linux installation. For the particular use-case of the manipulator, Ubuntu 16.04 MATE has been installed as it has full ROS support. The board has inbuilt WiFi capability which makes remote operation with ROS possible. The addition of the Raspberry Pi offers the manipulator the flexibility of being part of a network of robots or act as a standalone system when necessary as the Rasperry Pi is capable of running its own ROS instance or connect to an external instance.

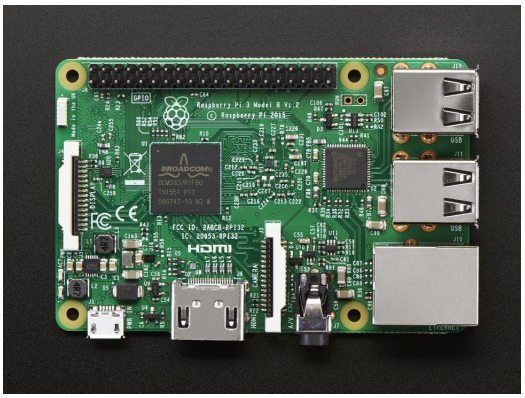

Figure 4.4: Raspberry Pi 3
Source: https://www.adafruit.com/product/3055

4.3 Data Transfer

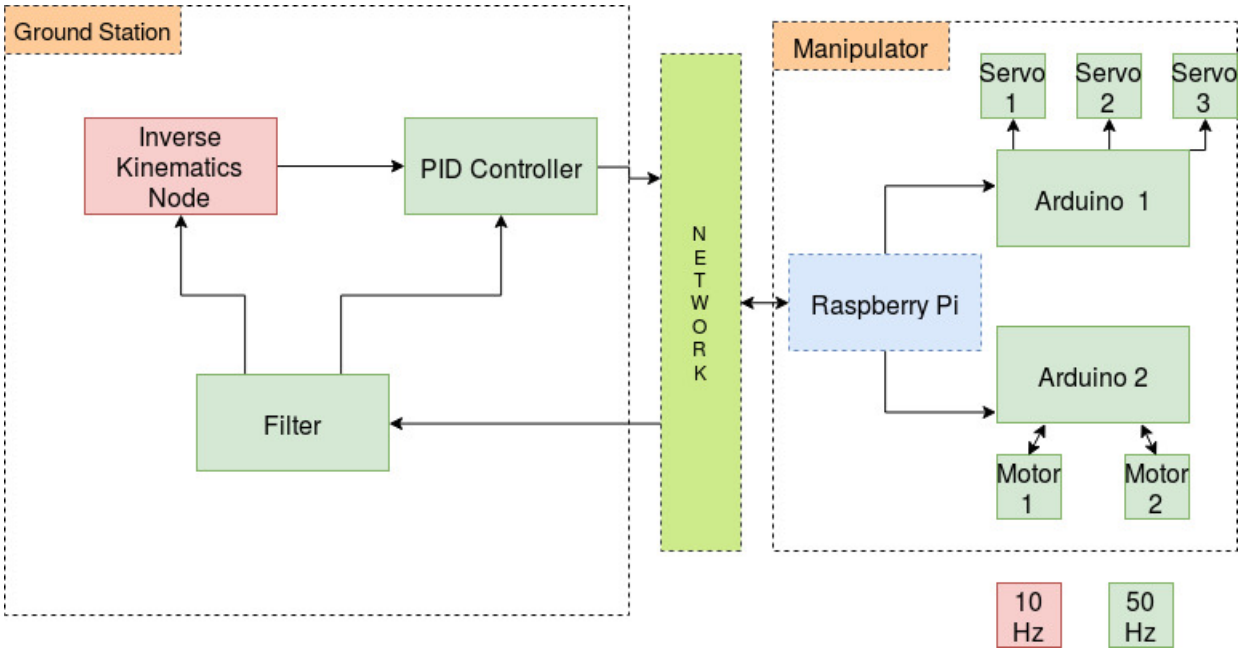

Figure 4.5: Data Flow Diagram

The data flow of the system used in the experiement is shown in Fig. 4.5. The second Arduino was added to relieve the computation and data processing load on the lone rosserial link used previously. This served to improve the reliability and

performance of the system. The splitting and stitching of the signals is performed using ROS and Python.

4.3.1 Communication Setup between Arduino and Raspberry Pi

A serial connection over USB is used to transfer data between the Arduino and the Rasperry Pi. This allows the sensor data from the Arduino to be accessed by the Raspberry Pi and relayed to the ROS network. There are two ways of achieving this i.e. directly through Python or through existing ROS packages for communication. The rosserial package provides libraries which extend the Arduino as a ROS node. This package provides turnkey tools for data transfer and communication, over serial ports, which are common to the ROS nodes running on the Raspberry Pi and the ground station, it also provides the Arduino the ability to handle multiple ROS topics at once. This method provides standardization which helps cuts down code deployment time and simplifies troubleshooting. The messages are broadcast to the ROS master node so they are accessible by all component nodes in the network.

4.3.2 Communication Setup between Arduino and Motors

There are two types of motors used on the manipulator. DC motors with position feedback using potentiometers and servo motors. These motors are both controlled via PWM but have different control interfaces; the servos are controlled directly via the servo control library provided with the Arduino software. The DC motors are controlled by sending a PWM signal to an H-Bridge motor controller shown in Fig 4.6. The motor controller has two inputs, one for the PWM input and another for motor direction. The direction can be controlled by setting the direction pin to High or Low, i.e. powering it on or off. The PWM signal is a measured in milliseconds and the duty cycle is varied. The total length of a PWM pulse is $1500\mu s$,

and the duty cycle is the portion of that pulse for which the signal is High. The duty cycle is represented as a percentage. This modulation of the duty cycle varies the voltage provided to the motors, thereby varying the motor speed.

The PWM timer on the Arduino is a 8-bit so the PWM commands range from

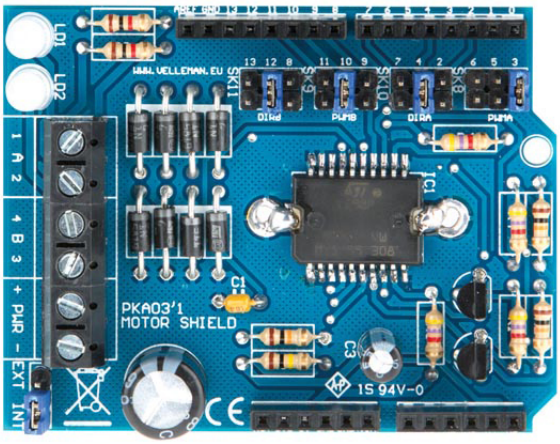

Figure 4.6: DC Motor Driver
Source: https://www.velleman.eu/products/view/?id=412174

0-255. This varies the output voltage from 0V-6V. The state of the direction pin determines which motor terminal receives the positive voltage. The motor driver gives a choice for all of the pins. The direction pins were 2 and 9 and the PWM pins were 3 and 12. Setting the direction pins High results in positive rotation of the motors and setting them low results in negative rotation in the manipulator reference frames. The total current draw of the motors is more than the Arduino can provide, so external power is provided.

The servo motors have 3 connections, power, ground and signal. The power and ground are connected to the external power supply and the signal wire is connected

directly to the Arduino. Bypass capacitors are attached across the power and ground of the motors to smooth out any noise in the DC power line.

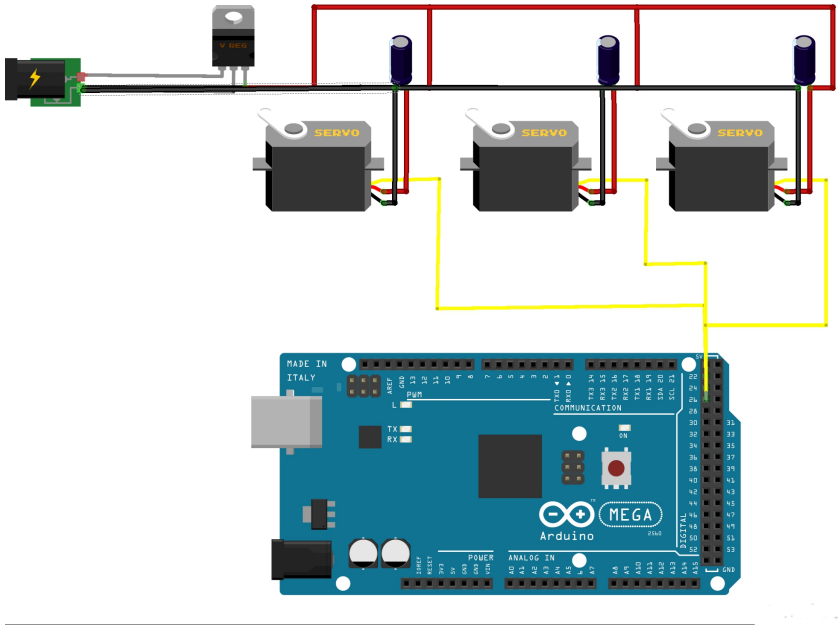

Figure 4.7: Servo layout

4.3.3 Communication Setup between the Arduino and Onboard Sensors

The two sensors being used for feedback are $10k\Omega$ potentiometers. Their range of motion is 300 degrees as tested in Fig 4.9. The Arduino has a 10-bit Analog-to-Digital Converter (ADC) so the reading the analog input from the potentiometers returns a value between 0-1023. This value was mapped from 0-300 to represent the range of motion of the potentiometer and a suitable offset was added to accomodate joint limits. Both of these sensors are powered from the Arduino's onboard 5V supply as their power-consumption is negligible and this setup allows the system to acquire sensor readings even when the external power supply is off, aiding diagnostics.

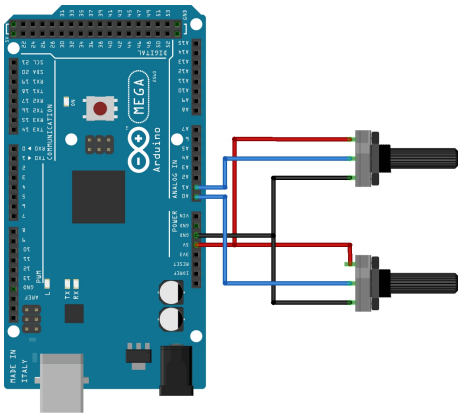

Figure 4.8: DC Motor Position Feedback

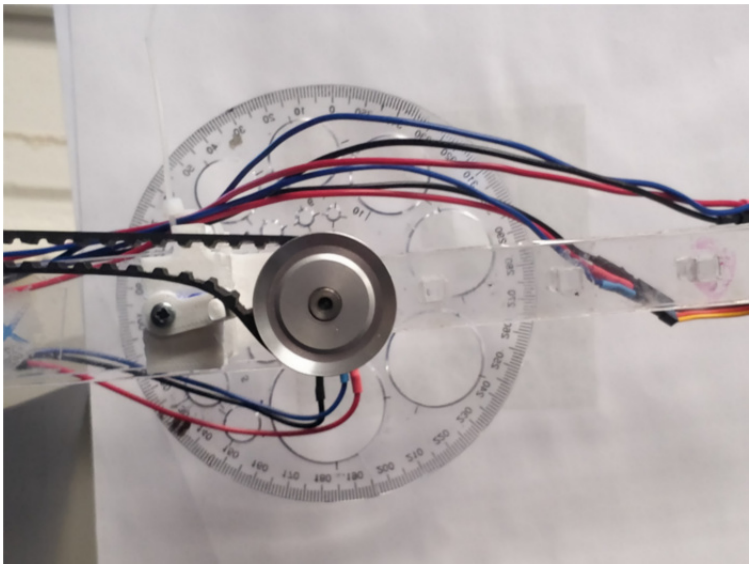

Figure 4.9: Potentiometer Calibration Setup

4.3.4 Communication Setup between the Raspberry Pi and Ground Station

The manipulator is a cyber-physical system (CPS) which means it has a decentralized control station that handles all of the processing and several nodes that handle the execution of the code. One way to operate the manipulator would have been to write all the code on the Raspberry Pi and make it system specific. However, this would mean there would be two different pieces of code used for testing and

deployment. This sort of fragmentation presents several difficulties and so a unifying framework such as ROS was used. Code can be written in C++/Python and used for testing and upon successful testing, the same code can be deployed on the robot hardware. Being a CPS, manipulator signals are sent over a network connection. The advantage is that the high processing power of the ground station can be harnessed along with the mobility and power efficiency of low-power components such as the Arduino. In addition, several such nodes can be run in parallel. The ROS networking setup requires minimal changes to the system after which there the test environment can be replaced with the robot hardware by simply changing the publisher blocks. The PID was run on the ground station using a Python PID module [20] to conserve processing on the Arduino due to its limited processing power and prevent delays [21]. The PID controller output was a value in the range of -255 to 255 which signifies the direction and PWM the motor should follow. This value was then transmitted to the Arduino.

CHAPTER 5

EXPERIMENTAL RESULTS

The same trajectories used for simulation were used to obtain the experimental results shown below. As expected, there were differences in the simulation and experimental results due to unmodeled phenomena such as friction, slack in the belt drive system, and friction forces.

5.1 2D Trajectory to a Single Point

The initial and final point for this trajectory are shown in Eq. 3.32 and the tracking performance is shown in Fig.5.1. Joints 2 and 3 are observed to follow their respective commanded trajectories with minor deviations. However, since joint 2 and 3 are in the very beginning of the chain and have the longest associated link lengths, these small deviations have a large impact on the end effector position. The maximum deviation seen in the path is around 1 cm as seen in Fig. 5.3. The performance of joints 2 and 3 can be improved with more PID tuning as there is some overshoot evident in the end-effector motion.

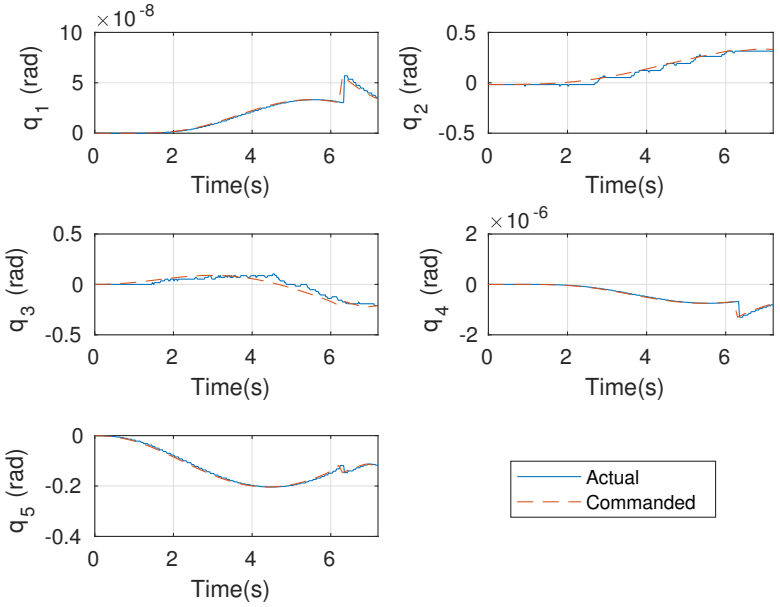

Figure 5.1: Manipulator Joint Angles, Desired and Actual

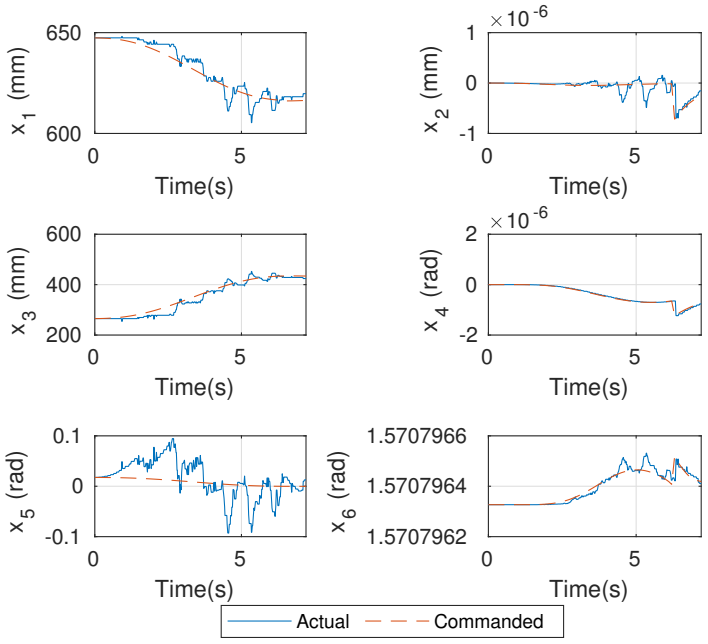

Figure 5.2: Manipulator Pose Variables, Desired and Actual

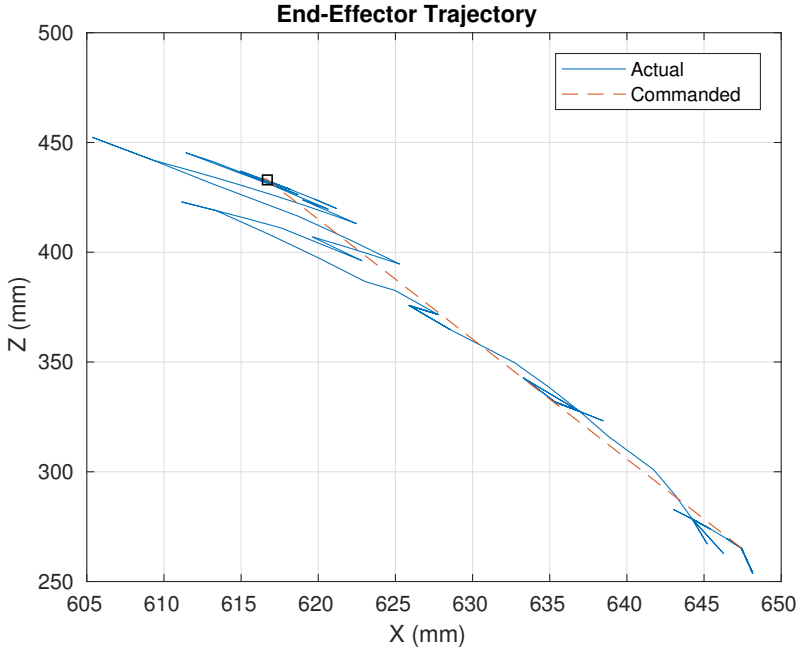

Figure 5.3: End-Effector Trajectory, Desired and Actual

5.2 3D Trajectory to a Single Point

The initial and final points for this test are listed in Eq. 3.33 As with the previous test, the commanded trajectory is followed with minor deviations as seen in Fig 5.4. There was an overshoot in joint 2 around the 5 second mark which is responsible for the largest error in the end effector trajectory which can be observed in Fig 5.5 and Fig 5.6.

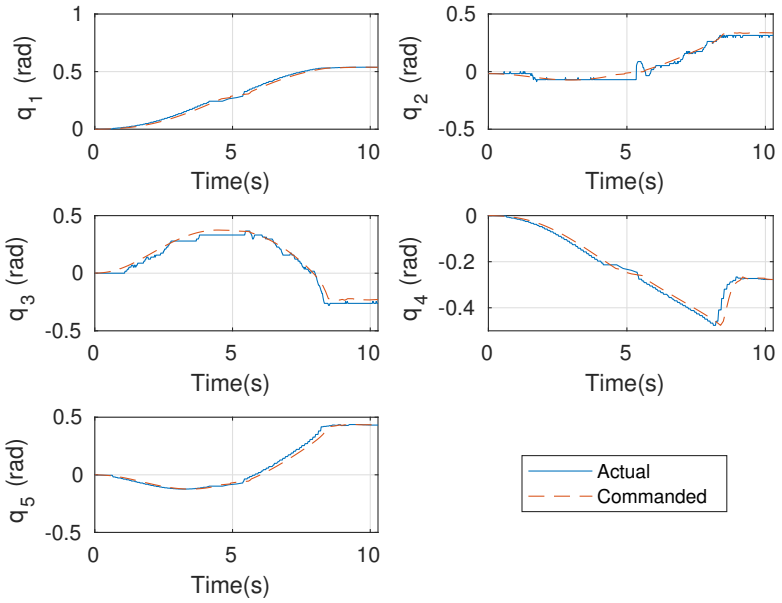

Figure 5.4: Manipulator Joint Angles, Desired and Actual

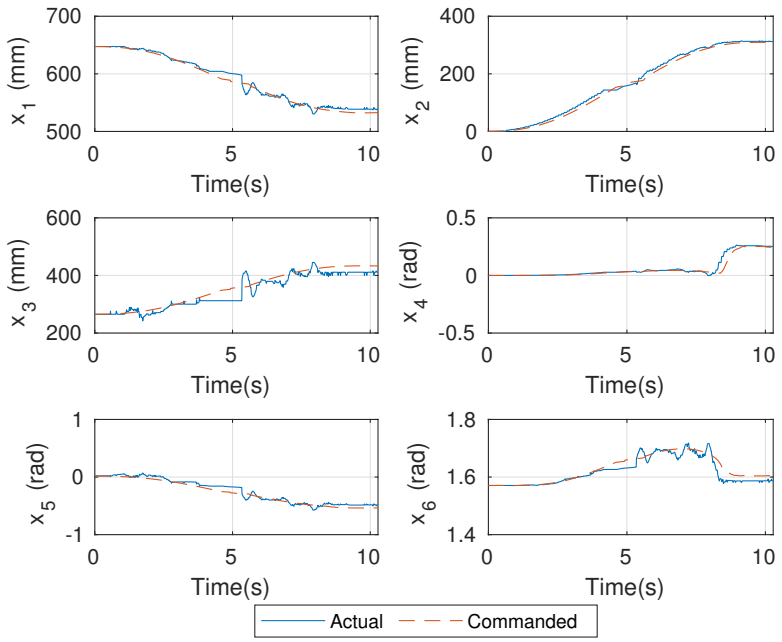

Figure 5.5: Manipulator Pose Variables, Desired and Actual

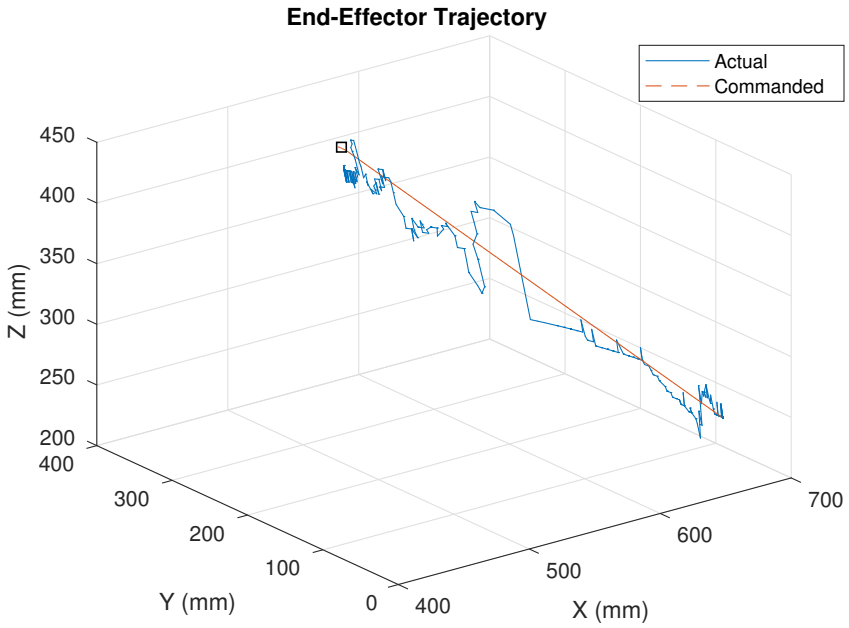

Figure 5.6: End-Effector Trajectory, Desired and Actual

5.3 2D Multi-Point Trajectory

The waypoints for this test are given in Table 3.3. Similar to previous tests, joint two shows the most deviation from the commanded trajectory in 5.7. This results in deviations in the end-effector position, however these deviations are limited to 2cm as seen from 5.9. The deviations are also reflected in the individual pose elements in Fig 5.8. Despite the deviations in the trajectory, the end effector reaches the final waypoints.

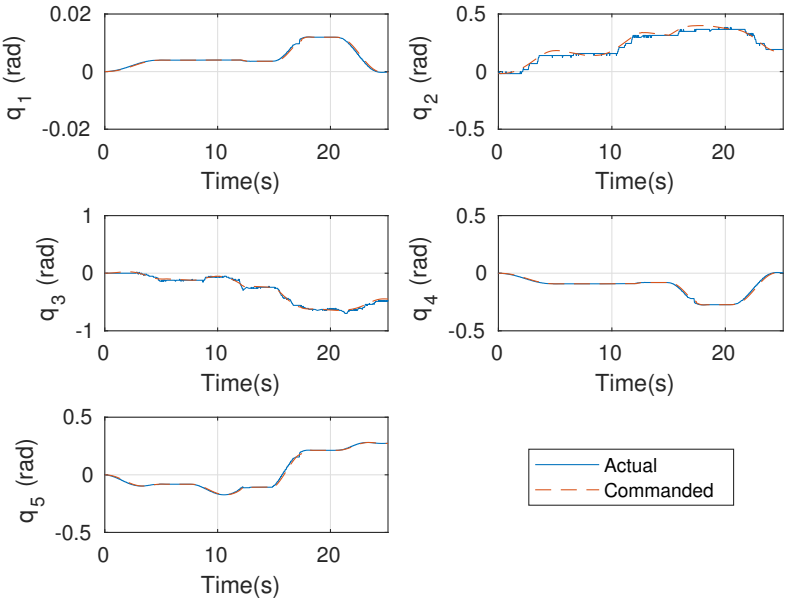

Figure 5.7: Manipulator Joint Angles, Desired and Actual

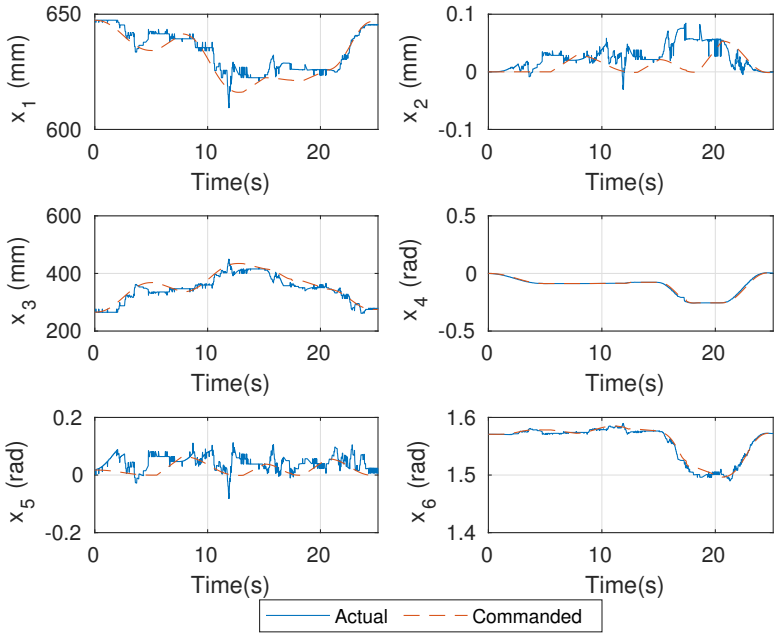

Figure 5.8: Manipulator Pose Variables, Desired and Actual

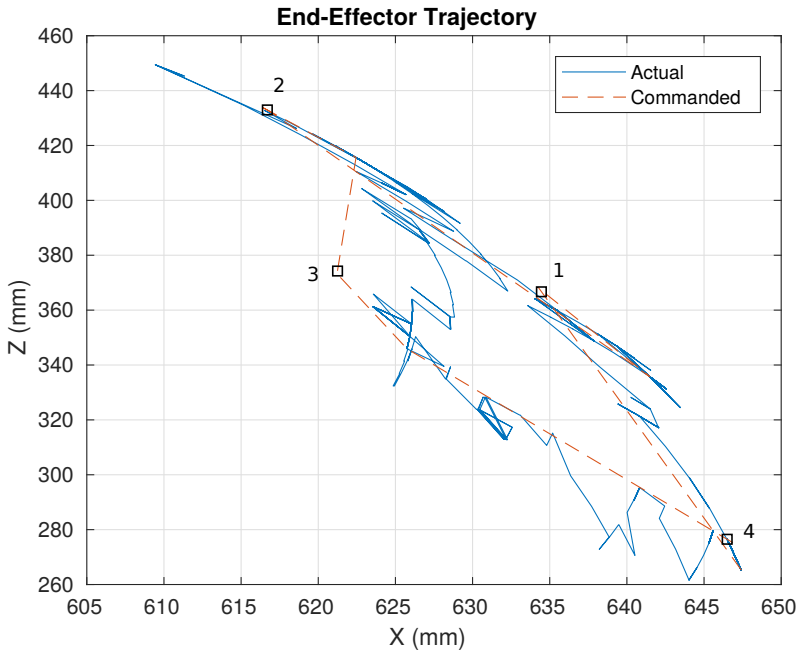

Figure 5.9: End-Effector Trajectory, Desired and Actual

5.4 3D Multi-Point Trajectory

The waypoints for this test are given in Table 3.4. Following the trend from previous tests, all joints follow the command trajectory. There was some oscillation in the second joint but it settled quickly and the manipulator tracked the trajectory with little error ater that as shown in Fig 5.12. The errors stem mainly from the second joint as seen in Fig 5.10 which reflect in the value of the Z-coordinate in the end-effector pose vector as seen in 5.11.

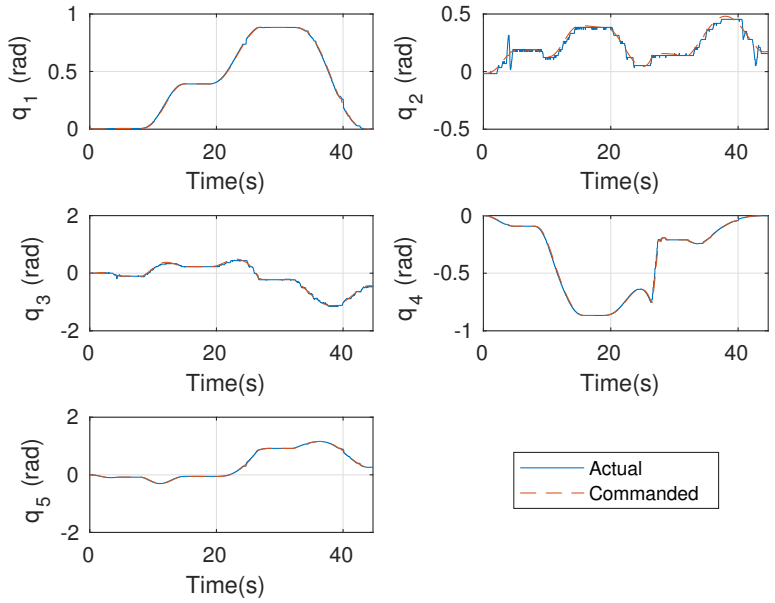

Figure 5.10: Manipulator Joint Angles, Desired and Actual

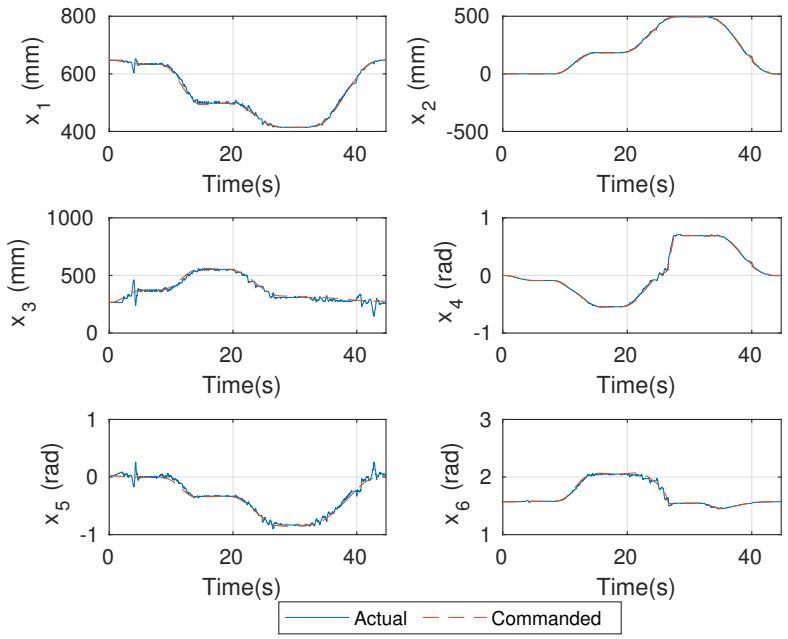

Figure 5.11: Manipulator Pose Variables, Desired and Actual

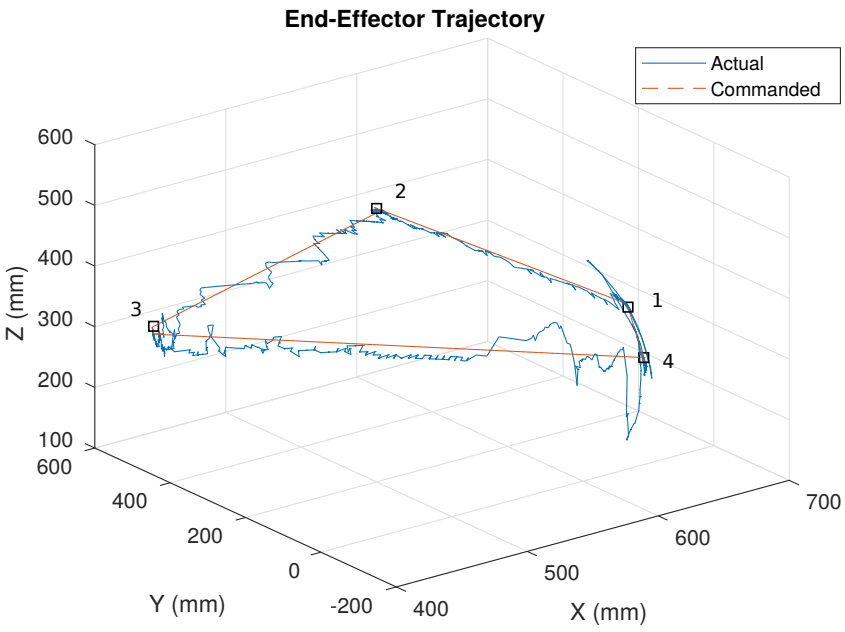

Figure 5.12: End-Effector Trajectory, Desired and Actual

CHAPTER 6

SUMMARY, CONCLUSION AND FUTURE WORK

6.1 Summary and Conclusion

This thesis presents the simulation and control of a 5 DOF manipulator designed to be mounted on a mobile robot. It is a networked cyber-physical system that offloads the computational heavy-lifting to a more powerful ground station allowing the system itself to remain lightweight. It features a high degree of modularity in terms of hardware as well as software.

The trajectory algorithm for the manipulator was analyzed and the motors were found to be adequate for a range of end-effector acceleration values despite the motors being designed for high-torque and low-velocity. These motors were placed towards the beginning of the kinematic chain, so low angular velocity of the motors resulted in much greater translational velocity for the end-effector. It is possible to replace these motors with motors with even higher gear ratios to increase load capacity as the necessary velocities are well within operational bounds.

The inverse kinematics solver was created to take into account desired end-effector velocity and used a damped least squares inversion with dynamic weights to invert the non-square manipulator Jacobian. With the initial and final conditions for velocity and acceleration being zero, the each subsequent trajectory segement was calculated once the end effector reached the goal position. Oscilliations were observed in the output trajectory when the joints approached their limits. These oscillations were damped by increasing Jacobian computation with the trade-off being additional computation time. This algorithm works as long as the goal position is

within reachable space.

The experimental results were satisfactory. The largest error seen in tested trajectories was close to 2 cm. This is within the expected values due to the precision of the sensors and fabrication tolerances. Joint 2 was the most difficult to tune for because that is the joint experiencing most torque as well as being responsible for the largest contribution to the end-effector position. As such, a majority of the errors in the end-effector pose were a result of position errors in joint 2.

6.2 Future Work

The work presented in this thesis creates a good testbed for future research on manipulation, perception, and opens up interesting avenues of research for the rover. Some future work includes:

- Revise trajectory planner so that the inverse kinematics for the next trajectory segment is performed while the robot is performing its assigned trajectory.
- Create a global trajectory planner such that it can modify the trajectory to the final point if a singularity cannot be removed.
- Design and implement a more robust controller for the most sensitive joints i.e joints that contribute most to the end effector position.
- Add force sensors to the end effector and factor input force into the feedback.

Appendices

Appendix A

Software Configuration

The Robot Operating System (ROS) is software framework used for robot control. It is a collection of tools, protocols, libraries and packages that simplify the creation of complex, networked, cyber-physical systems. ROS allows for standardization across several robotic platforms and handles all the low level communication between nodes. This leaves the user more time to focus on the robot software. Further details are available in [22]

- ROS Filesystem
  - Packages: ROS Packages are the most basic unit of the ROS system. It contains the ROS runtime process, libraries, configuration files, etc. which are organized as a single unit.
  - Messages: ROS messages are a type of formatted information sent from one ROS process to another using topics. Custom message types can be defined within a package such as

    `package_name/msg/msg_name.msg`
- ROS Computational-Graph level: This is the framework used for handling all communication between processes.
  - Master: ROS Master provides the registration and lookup for all nodes and topics. Nodes must be able to connect to the master to communicate with each other.
  - Nodes: Processes that form the network and carry out the computations.

  - Parameter Server: The parameter server is a centralized data storage location that stores model data.
  - Topics: Topics are the buses for data exchange. A node is said to "publish" data when it writes to a topic and is said to "subscribe" to a topic when it reads data from the topic. Each node may subscribe and publish to multiple topics.
  - Bag: Bags are message storage containers that store all topic activity while they are recording. This is useful for data logging and analysis after simulations. Bag data can be analyzed in MATLAB using tools provided in the Robotics System Toolbox [23].
- Robot Simulation and Visualization: ROS has two tools for simulation and visualization.
  - Gazebo: Gazebo is the integrated robot simulator which offers multiple physics engines for simulating the robot. The user can create environments with obstacles, objects to be detected, terrain to be navigated to simulate various scenarios for the robot and test control algorithms.
  - Rviz: Rviz is used to play back recorded data from the simulations and visualize the robot's movements. It can also be used to check how the Gazebo environment reads the robot description file and verify all joints and motors are working as intended.

A.1 Connecting the Arduino and the Raspberry Pi

The Arduino IDE and ROS package rosserial needs to be installed on the Raspberry Pi [19]. The connection can be established by running the following command

```
rosrun rosserial_python serial_node.py /dev/ttyACMX
```

where /dev/ttyACMX is replaced by the port to which the Arduino is attached.

A.2 Network setup

ROS nodes can connect to remote masters as long as the IP address and port of the master node is exported as an environment variable in the shell. ROS handles all the other communication details with the user handling minimum setup effort. The two variables can be changed with the following commands

```
export ROS_IP = <IP address of current machine>
export ROS_MASTER_URI = http://<IP of the machine running the
    ROS master node>:<The port on which the master is talking>
```

The default port is 11311 however the port can be specified while launching ROS with

```
roscore -p <port number>
```

for example the commands for the test hardware were as follows (The router was configured such that the IP addresses were static)

- Commands run on the ground station

```
export ROS_IP = 192.168.0.42
export ROS_MASTER_URI = http://192.168.0.42:11311
```

- Commands run on the Raspberry Pi

```
export ROS_IP = 192.168.0.21
export ROS_MASTER_URI = http://192.168.0.42:11311
```

REFERENCES

[1] Wampler, C. W., "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 16, No. 1, 1986, pp. 93–101.

[2] Nakamura, Y. and Hanafusa, H., "Inverse kinematic solutions with singularity robustness for robot manipulator control," *Journal of dynamic systems, measurement, and control*, Vol. 108, No. 3, 1986, pp. 163–171.

[3] Na, M., Yang, B., and Jia, P., "Improved damped least squares solution with joint limits, joint weights and comfortable criteria for controlling human-like figures," *Robotics, Automation and Mechatronics, 2008 IEEE Conference on*, IEEE, 2008, pp. 1090–1095.

[4] Buss, S. R. and Kim, J.-S., "Selectively damped least squares for inverse kinematics," *Journal of Graphics tools*, Vol. 10, No. 3, 2005, pp. 37–49.

[5] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G., *Robotics: Modelling, Planning and Control*, Springer Publishing Company, Incorporated, 2010.

[6] Sharma, S., Kraetzschmar, G. K., Scheurer, C., and Bischoff, R., "Unified closed form inverse kinematics for the KUKA youBot," *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, VDE, 2012, pp. 1–6.

[7] Simon, D. and Isik, C., "A trigonometric trajectory generator for robotic arms," *International Journal of Control*, Vol. 57, No. 3, 1993, pp. 505–517.

[8] Piazzi, A. and Visioli, A., "Global minimum-jerk trajectory planning of robot manipulators," *IEEE transactions on industrial electronics*, Vol. 47, No. 1, 2000, pp. 140–149.

[9] Godbole, A., VNV, M., Quillen, P., and Subbarao, K., "Optimal Trajectory Design and Control of a Planetary Exploration Rover," *AIAA SPACE 2017*, 2017.

[10] Wang, S., Wan, J., Zhang, D., Li, D., and Zhang, C., "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination," *Computer Networks*, Vol. 101, 2016, pp. 158–168.

[11] Craig, J. J., *Introduction to Robotics: Mechanics and Control*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd ed., 1989.

[12] Ollydbg, "DenavitHartenberg parameters," Mar 2018, Available at `https://en.wikipedia.org/wiki/DenavitHartenberg_parameters#/media/File:DHParameter.png`.

[13] Moré, J. J., "The Levenberg-Marquardt algorithm: implementation and theory," *Numerical analysis*, Springer, 1978, pp. 105–116.

[14] Buss, S. R., "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, Vol. 17, No. 1-19, 2004, pp. 16.

[15] "Robot Operating System," Available at `http://www.ros.org`.

[16] "ROS control," Available at `http://gazebosim.org/tutorials/?tut=ros_control`.

[17] Bhoraniya, A. L., "Design and Development of 5 Degree of Freedom Robotic Arm," Aerospace Systems Lab, unpublished.

[18] Crassidis, J. L. and Junkins, J. L., *Optimal Estimation of Dynamic Systems, Second Edition (Chapman & Hall/CRC Applied Mathematics & Nonlinear Science)*, Chapman & Hall/CRC, 2nd ed., 2011.

[19] "ROSSerial protocol for wrapping ROS serialized messages over a serial port or network socket." Available at `http://wiki.ros.org/rosserial`.

[20] Ivmech Mekatronik, "Python PID Controller ivPID," 2016, Available at https://github.com/ivmech/ivPID.

[21] Beauregard, Brett, "Improving The Beginner's PID," Available at http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-reset-windup/.

[22] Joseph, L., *Mastering ROS for robotics programming*, Packt Publishing Ltd, 2015.

[23] MATLAB, "Robotics System ToolBox," Available at https://www.mathworks.com/products/robotics.html.

BIOGRAPHICAL STATEMENT

Roopak Karulkar received his Honors Bachelor of Science in Mechanical Engineering, Magna cum Laude, from the University of Texas at Arlington in 2016. After which, he joined the Graduate School at The University of Texas at Arlington to pursue an M.S. in Mechanical Engineering. His areas of interest include design and control of dynamical systems.