Design, Simulation and Testing of a Controller

And

Software Framework for Automated Construction by a Robotic Manipulator

By

Sushrut Jigneshbhai Gandhi

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2019 by the
Graduate Supervisory Committee:

Spring Berman, Chair
Hamidreza Marvi
Sze Zheng Yong

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

The construction industry is very mundane and tiring for workers without the assistance of machines. This challenge has changed the trend of construction industry tremendously by motivating the development of robots that can replace human workers. This thesis presents a computed torque controller that is designed to produce movements by a small-scale, 5 degree-of-freedom (DOF) robotic arm that are useful for construction operations, specifically bricklaying. A software framework for the robotic arm with motion and path planning features and different control capabilities has also been developed using the Robot Operating System (ROS).

First, a literature review of bricklaying construction activity and existing robots' performance is discussed. After describing an overview of the required robot structure, a mathematical model is presented for the 5-DOF robotic arm. A model-based computed torque controller is designed for the nonlinear dynamic robotic arm, taking into consideration the dynamic and kinematic properties of the arm. For sustainable growth of this technology so that it is affordable to the masses, it is important that the energy consumption by the robot is optimized. In this thesis, the trajectory of the robotic arm is optimized using sequential quadratic programming. The results of the energy optimization procedure are also analyzed for different possible trajectories.

A construction test bed setup is simulated in the ROS platform to validate the designed controllers and optimized robot trajectories on different experimental scenarios. A commercially available 5-DOF robotic arm is modeled in the ROS simulators Gazebo and Rviz. The path and motion planning are performed using the MoveIt-ROS interface, and

are also implemented on a physical small-scale robotic arm. A Matlab-ROS framework for execution of different controllers on the physical robot is described. Finally, the results of the controller simulation and experiments are discussed in detail.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Spring Berman for believing in me, and for giving me this wonderful opportunity to work in her lab. I have been extremely lucky to have an advisor who motivated me to learn and work harder. I will forever be grateful to her guidance during my journey in ASU.

I would like to thank Dr. Marvi for showing me the field of robotics and teaching me the modeling strategies for manipulation. I will miss his lectures and interesting videos of new unique robots.

I would also like to thank Dr. Yong for introducing me to concepts of controls, which helped me throughout my thesis work. I hope the long discussions on control with Dr. Yong will guide me throughout my career.

I would like to thank all my ACS lab members and specially Hamed Farivarnejad for guiding me in design of controllers. I am also thankful to Shreyans and Subhrajyoti for helping me in my research and thesis documentation.

Finally, I would like to express profound gratitude to my family and friends for providing me with support and encouragement throughout the process of researching and writing this thesis. This journey is difficult to imagine without them.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

APPENDIX

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

1.1 Introduction

Over the past decade, a major focus in technological innovations has been to make industrial processes autonomous. In recent years, there has been a tremendous increase in demand and use of robots to replace humans for performing mundane and tiring jobs. As the construction industry has been booming over the last few years, the labor shortage is a major concern for efficient productivity. According to a survey by the National Association of Homebuilders (QUOCTRUNG BUI 2018), sixty percent of contractors are finding it difficult to recruit skilled laborers for their projects. As trained labors are aging out of the industry, robots equipped with effective autonomous technologies will increase productivity and efficiency. Currently, extensive research is being conducted to make the bricklaying process independent of humans. Robots like Hadrian X (Pivac 2016) and SAM100 (PODKAMINER 2016) have been demonstrated to be five times faster than human workers. According to (PODKAMINER 2016), SAM100 can lay 300-400 bricks per hour compared to a human bricklayer's 60-75 bricks. There are currently different types of robots producing houses and industrial structures. Hadrian X uses pre-fabricated walls to construct a house in 3 days (Quirke 2018), and SAM100 uses bricks and mortar for building walls. Bricklaying is difficult to automate, due to the uncertain environment of the construction sites. Describing the scenario of automation in bricklaying, one of the participants of the Las Vegas Bricklaying competition said that it is difficult for a robot to judge how much mortar is enough for particular brick or how to build a round wall with square bricks (QUOCTRUNG BUI 2018). The current robots are also not flexible and

mobile enough to build walls higher than 2 meters. These issues are preventing robots from being widely used in the construction industry.

1.2 Outline of the thesis

This thesis aims to achieve better movement of robotic arms that are used in bricklaying operations by designing a model-based controller for such robots and considering the nonlinearity of the robot dynamics. It is useful to have a software framework that can be used to simulate the robot operating with different controllers before implementing them on a real robot.

This thesis describes the controller design, simulation and software implementation for a 5 degree-of-freedom (DOF) PhantomX Reactor robot arm (Rick 2017). From a Solidworks design, the link and joint properties were considered for the controller design. The required design concepts are discussed in Chapter 2. The computed torque controller design and simulation are described in Chapter 3. In industry, when the robot is used for mass production, it is important to minimize the energy consumption of the robot along its trajectory. This optimization procedure is described in Chapter 4. Chapter 5 explains the software requirements and software setup for path planning and controller implementation. The simulation results, experimental setup and results are illustrated in Chapter 6. The conclusion and future possible modifications are explained in Chapter 7.

1.3 Contributions of the thesis

In this thesis, a computed torque controller has been designed for a nonlinear 5 degree-of-freedom robotic arm to perform movements that are useful for bricklaying operations. A new method of trajectory optimization is implemented to make the robot energy efficient. A software framework is developed to integrate modeling, path planning, controller implementation and simulation of the robotic arm. This setup can be implemented on any robotic arm by just changing its Unified Robot Description Format (URDF) files.

CHAPTER 2

MATHEMATICAL MODELING OF ROBOTIC ARM

2.1 Rotational Matrices

The rotation matrix R is very important in modeling a robotic arm. It transforms the current coordinate frame to a reference frame and vice-versa. The column vectors of the rotation matrix are orthogonal to each other, as they are unit vectors of an orthonormal frame. The rotation matrix is in the Special Orthonormal group SO(n) for m × m real matrices, because of these special properties:

- Rotation matrices have a determinant of 1.
- Rotation matrices are orthogonal matrices: $R^T = R^{-1}$

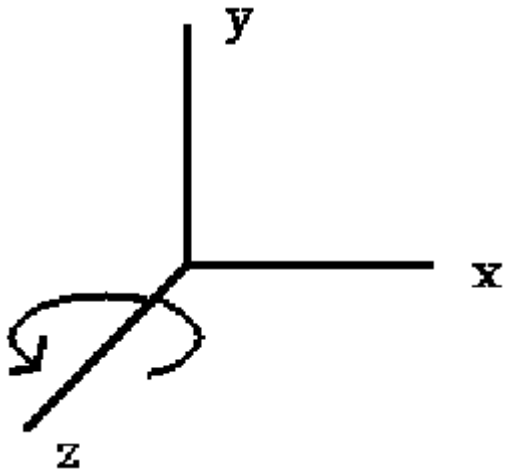

Figure 2.1 Rotation of coordinate frame about its z-axis

The rotation matrix for rotating a coordinate frame about its z-axis by angle φ is:

$$Rz(\varphi) = \begin{bmatrix} cos\varphi & -sin\varphi & 0 \\ sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

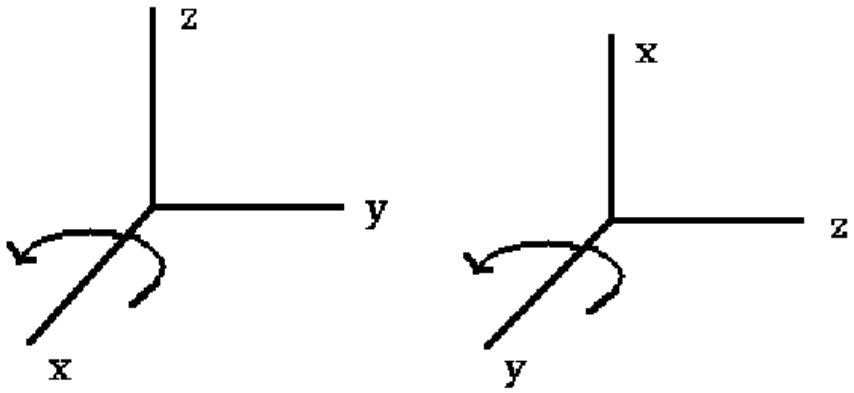

Figure 2.2 Rotation of frame about x-axis and y-axis, respectively

The rotation matrix for rotating a frame about its x-axis by angle ψ and y-axis by angle ϕ, respectively, is:

$$Rx(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\psi & -sin\psi \\ 0 & sin\psi & cos\psi \end{bmatrix}$$

$$Ry(\phi) = \begin{bmatrix} cos\phi & 0 & sin\phi \\ 0 & 1 & 0 \\ -sin\phi & 0 & cos\phi \end{bmatrix}$$

These matrices are useful to describe rotation about any arbitrary axis.

2.2 Homogeneous Transformation

The configuration of a rigid body is expressed by the translation of the body from a reference frame and the body's orientation relative to this frame.

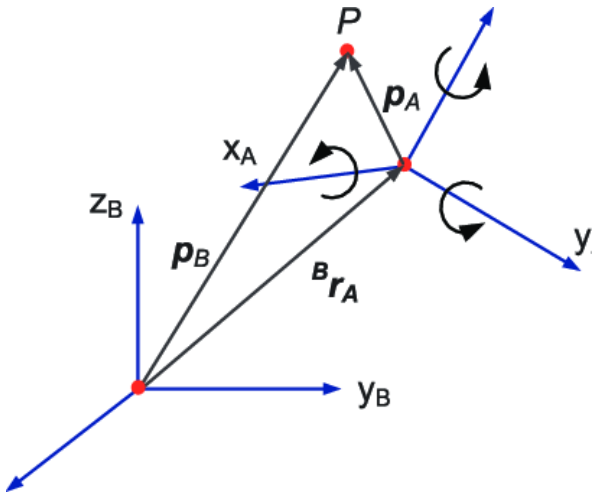

Figure 2.3 Representation of a point P in two different frames

Consider an arbitrary point P in three-dimensional space (Figure 2.3). Here, p_B contains the vector coordinates of point P with respect to the origin of reference frame B. The vector from the origin of frame B to the origin of frame A is represented by p_{AB}. Let ${}^B r_A$ be the rotation matrix of frame A with respect to frame B. Then p_B will be,

$$p_B = p_{AB} + {}^B r_A p_A$$

The rigid transformation (p_{AB}, ${}^B r_A$) is in the special Euclidean group SE(3) and can be represented by a 4×4 matrix which is called the *homogeneous transformation matrix*.

2.3 Forward Kinematics

A robotic arm consists of a series of links which are connected by joints. The joints are usually of two types: revolute and prismatic. The whole setup is called a kinematic chain, where one end of the robotic arm is attached to a fixed base and the other end is attached to an end-effector.

The structure of a robotic arm can be described by the number of degrees-of-freedom (DOFs) associated with the arm. The configuration space of the arm is comprised of all possible values of the joint variables, which are angles for revolute joints and distances for prismatic joints. The configuration space of an arm with *r* revolute joints and *p* prismatic joints has (*r* + *p*) DOFs. The forward kinematics can be used to compute the pose of the end-effector with respect to base of the robot. This computation is done using particular functions of joint variables, described in the next section.

2.4 Denavit-Hartenberg (D-H) Convention

The most commonly used method to calculate forward kinematics for an open-chain robotic arm is the Denavit-Hartenberg (D-H) method. The D-H method derives the position and orientation of a link with respect to an adjacent link. The DH parameters of the manipulator denote the geometric relations between its joints and links. These parameters are used for the dynamic modelling of the robot.

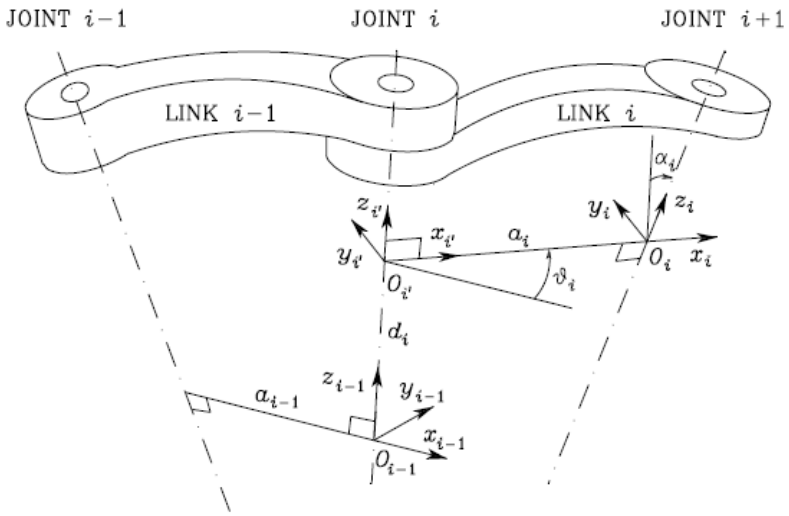

Figure 2.4 Denavit-Hartenberg (D-H) parameters (Siciliano et al. 2009, p.62)

There are specific rules for the D-H convention (Siciliano et al. 2009, p.62), outlined below (see Figure 2.4 for notation):

- Select axis Z_i along the axis of joint i+1.
- Find the common normal to the Z_{i-1} and Z_i axes and intersect it with axis Z_i. The intersection point is considered to be the origin point O_i .
- Select axis X_i with the same direction as the common normal to the axes Z_{i-1} and Z_i. This axis points from joint i to joint i+1.
- Select the axis Y_i to make the coordinate frame a right-handed frame.

The convention has some exceptions and assumptions.

- For the frame at joint 1, the Z_0 axis direction is specified by the joint orientation. The X_0 axis and the origin of the frame are defined arbitrarily.
- If two continuous frames are parallel, then the common normal between them is not defined uniquely.
- When two Z axes intersect each other, then the X axis direction can be chosen arbitrarily.
- If joint i is prismatic, then the Z_{i-1} axis direction is chosen arbitrarily.

After establishing these frame conventions, the DH parameters can be used to find the positions and orientations of the robotic arm frames. The DH parameters are defined as:

a_i = Distance between the frame origins O_i and $O_{i'}$

d_i = Offset of $O_{i'}$ from O_{i-1}

α_i = Angle between axes Z_{i-1} and Z_i about axis X_i

θ_i = Angle between axes X_{i-1} and X_i about axis Z_{i-1}

(An angle is considered positive if the rotation is made counter-clockwise.)

If a joint is revolute, then its joint variable is θ_i, and if a joint is prismatic, then its joint variable is d_i.

The relationship between two adjacent frames is determined by a 4×4 homogeneous transformation matrix. For example, translate the frame with origin O_{i-1} by distance d_i along axis Z_{i-1} and rotate is about Z_{i-1} by an angle θ_i. The associated transformation matrix is:

$$A_{i'} = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(Here, $c\theta = \cos\theta$ and $s\theta = \sin\theta$.)

Now translate the frame by distance a_i along axis $X_{i'}$ and rotate it about $X_{i'}$ by an angle α_i. The associated transformation matrix is:

$$A_i = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The resulting homogeneous transformation matrix after post-multiplication is:

$$A = A_{i'}A_i = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Following this method, it is possible to find the position and orientation of the end-effector with respect to the base of the robotic arm. For a 5-DOF robotic arm, this configuration can be derived by post-multiplication of 5 rotation matrices. Here, the forward kinematics of a 5-DOF robotic arm is derived using the DH parameters.

2.5 PhantomX Reactor Robotic Arm

In this section, the DH parameters of a PhantomX Reactor robotic arm (Figure 2.5) are computed, and the kinematics of the robot are modeled. The robot has 5 DOFs, and all five of its joints are revolute joints.

Table 1. Values of Robot Parameters

$a_0 = 0\ cm$	$d_0 = 0\ cm$
$a_1 = 0\ cm$	$d_1 = 4\ cm$
$a_2 = 14.611\ cm$	$d_2 = 3.97\ cm$
$a_3 = 14.551\ cm$	$d_3 = 0\ cm$
$a_4 = 0\ cm$	$d_4 = 0\ cm$
$a_5 = 0\ cm$	$d_5 = 4.6\ cm$

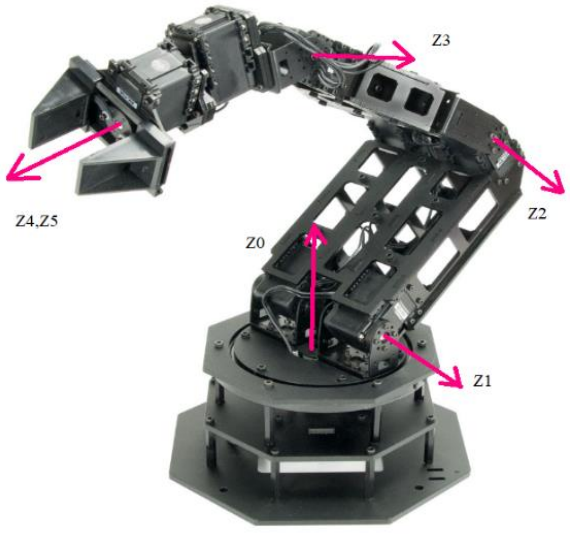

Figure 2.5 PhantomX Reactor robotic arm (Rick 2017) with axes superimposed.

Table 2. Robot DH Parameters

Link	a	d	α	θ
1	0	d_1	90°	q_1
2	a_2	d_2	0	q_2
3	a_3	0	0	q_3
4	0	0	90°	q_4
5	0	d_5	0	q_5

Let A^j_i represent the transformation of frame i to frame j, and define c1 = cos θ_1 , c1c2 = cos θ_1 * cos θ_2, c12 = cos (θ_1+θ_2), and so on.

For the PhantomX Reactor robotic arm,

$$A^0_1 = \begin{bmatrix} c1 & 0 & s1 & 0 \\ s1 & 0 & -c1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A^0_2 = \begin{bmatrix} c1c2 & -c1s2 & s1 & 4*s1+15*c1c2 \\ c2s1 & -s1s2 & -c1 & 15*c2s1-4*c1 \\ s2 & c2 & 0 & 15*s2+3 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$A^0_3 = \begin{bmatrix} c1c23 & -c1s23 & s1 & 4*s1+15*c1c2+15*c1c2c3-15*c1s2s3 \\ c23s1 & -s1s23 & -c1 & 15*c2s1-4*c1-15*s1s2s3+15*c2c3s1 \\ s23 & c23 & 0 & 15*s23+15*s2+3 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$A^0_4 = \begin{bmatrix} c1c234 & s1 & c1s234 & 4*s1+15*c1c2+15*c1c2c3-15*c1s2s3 \\ c234s1 & -c1 & s234*s1 & 15*c2s1-4*c1-15*s1s2s3+15*c2c3s1 \\ s234 & 0 & -c234 & 15*s23+15*s2+3 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$A^0_5 = \begin{bmatrix} s1s5 + c1c5c234 & c5s1 - c234c1s5 & c1s234 & Px \\ c234s1 & -c1 & s234 * s1 & Py \\ s234 & 0 & -c234 & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

where,

- Px = $4 * s1 + 15 * c1c2 + 8 * c4(c1c2s3 + c1c3s2) + 8 * s4(c1c2c3 - c1s2s3) + 15 * c1c2c3 - 15 * c1s2s3$
- Py = $-4 * c1 + 15 * s1c2 + 8 * c4(s1c2s3 + s1c3s2) + 8 * s4(s1s2s3 - s1c2s3) + 15 * c1c2c3 - 15 * c1s2s3$
- Pz = 15*s23 – 8*c234 +15*s2+3

CHAPTER 3

DESIGN AND SIMULATION OF CONTROLLER

3.1 Jacobian

Differential kinematics are used to find the relationships between the robotic arm's joint velocities and its end-effector linear and angular velocities. Defining the end-effector linear velocity as $\dot{p}_e$, the end-effector angular velocity as w_e, and the vector of joint velocities as $\dot{q}$, then

$$\dot{p}_e = J_p(q)\,\dot{q} \tag{3.1}$$

$$w_e = J_o(q)\,\dot{q} \tag{3.2}$$

Here J_p is a (3 × n) matrix that relates the end-effector linear velocity to the joint velocities. J_o is a (3 × n) matrix that relates the end-effector angular velocity to the joint velocities. These equations can also be written in the form,

$$v_e = \begin{bmatrix} \dot{p}_e \\ w_e \end{bmatrix} = J(q)\dot{q} \tag{3.3}$$

Here J is known as the Geometric Jacobian of the 5-DOF manipulator. It is a function of the manipulator's joint variables. The Geometric Jacobian is a very useful property for the robotic arm, as it is used to its find singularities, inverse kinematics and dynamics.

3.2 Singularities

During the motion of the robotic arm, there are some configurations at which the instantaneous Jacobian matrix is not a full rank matrix. Those configurations of the manipulator are called the singularities of the robotic arm.

At singular configurations, the robotic arm loses its one degree of freedom , which shows that it cannot achieve any arbitrary motion. The Jacobian matrix is used to find the inverse kinematics, and if a singularity exists, then there will be infinite possible configurations. Near a singularity, small velocities of the end-effector cause large velocities in the joints of the arm, making the motion jerky and hazardous for motors.

Singularities are usually of two types:

1) Boundary singularities: This singularity arises when the robotic arm is fully stretched or retracted; i.e., when the arm reaches the boundaries of its workspace. It can be avoided by assigning minimum and maximum values for each joint angle.
2) Internal singularities: This singularity arises during particular motions of the end-effector in its workspace. This can be hazardous for joint motors, as it can occur during the planned motion of the manipulator. It can be avoided by considering internal singularities during the trajectory planning.

3.3 Dynamics

In the manipulator design, the major concept is its dynamic modeling. The dynamic modeling reduces the costs of actual experiments, as simulations are possible if the dynamic model is accurate. Dynamic models are often derived using the Lagrange method, which takes into account the kinetic energy and potential energy of the robot.

3.3.1 Lagrange Method

The equation of motion of a manipulator can be derived using the Lagrange method. The Lagrangian of the robotic arm is defined as:

$$\mathrm{L}(q, \dot{q}) = \mathrm{K}(q, \dot{q}) - \mathrm{U}(q)\ , \tag{3.4}$$

where K is kinetic energy and U is total potential energy of the robotic arm. From the given Lagrangian, the Lagrange equations are written as

$$\frac{\partial}{\partial \mathrm{t}} \frac{\partial \mathrm{L}}{\partial \dot{q}_i} - \frac{\partial \mathrm{L}}{\partial q_i} = \mathrm{f}_i\ , \tag{3.5}$$

i = 1,…,n, where $\mathrm{f}_{[n\times 1]}$ is a vector of forces associated with the joint angles q_i and n is the number of joints.

As Lagrange equations are combination of kinetic energy and potential energy, the formulas of each are shown here.

The kinetic energy of a link *i* is defined by (Bruno Siciliano, et al. 2009),

$$K_{l_i} = \frac{1}{2} m_{l_i} \dot{p}_{l_i}^T \dot{p}_{l_i} + \frac{1}{2} w_i^T R_i I_{l_i}^i R_i^T w_i \tag{3.6}$$

where $\dot{p}_{l_i}$ = the linear velocity of the center of mass of link *i*

w_i = the angular velocity of link *i*

R_i = the rotation matrix of link *i* with respect to the base frame

$I_{l_i}^i$ = the inertia matrix relative to the center of mass of link *i*

m_{l_i} = the mass of link *i*

Here the linear and angular velocities can be written in generalized form, which is a function of Jacobian matrices:

$$\dot{p}_{l_i} = J_{P_1}^{l_i} \dot{q}_1 + \cdots + J_{P_i}^{l_i} \dot{q}_i = J_P^{l_i} \dot{q} \tag{3.7}$$

$$w_i = J_{O_1}^{l_i} \dot{q}_1 + \cdots + J_{O_i}^{l_i} \dot{q}_i = J_O^{l_i} \dot{q} \tag{3.8}$$

where Jacobians can be derived as,

$$J_P^{l_i} = \begin{bmatrix} J_{P_1}^{l_i} \ldots & J_{P_i}^{l_i} & 0 \end{bmatrix} \tag{3.9}$$

$$J_O^{l_i} = \begin{bmatrix} J_{O_1}^{l_i} \dots & J_{O_i}^{l_i} & 0 \end{bmatrix} \tag{3.10}$$

The Jacobian matrices in these definitions are computed as (Marvi 2018),

$$J_{P_j}^{l_i} = \begin{cases} Z_{j-1} & prismatic\ joint \\ Z_{j-1} \times (p_{l_i} - p_{j-1}) & revolute\ joint \end{cases}$$

$$J_{O_j}^{l_i} = \begin{cases} O & prismatic\ joint \\ Z_{j-1} & revolute\ joint \end{cases}$$

Then the kinetic energy formula can be written in terms of the Jacobian matrices:

$$K_{l_i} = \frac{1}{2} m_{l_i} \dot{q}^T {J_P^{l_i}}^T J_P^{l_i} \dot{q} + \frac{1}{2} \dot{q}^T {J_O^{l_i}}^T R_i I_{l_i}^i R_i^T J_O^{l_i} \dot{q} \tag{3.11}$$

The motor is also considered in dynamics modelling when the motors are very heavy compared to links. The kinetic energy of a motor *i* can be written as,

$$K_{m_i} = \frac{1}{2} m_{m_i} \dot{p}_{m_i}^T \dot{p}_{m_i} + \frac{1}{2} w_{m_i}^T I_{m_i} w_{m_i} \tag{3.12}$$

where $\dot{p}_{m_i}$ = the linear velocity of the center of mass of motor *i*

w_{m_i} = the angular velocity of motor *i*

I_{m_i} = the inertia matrix relative to the center of mass of motor *i*

m_{m_i} = the mass of motor *i*

Here the linear and angular velocities can be written in generalized form, which is a function of Jacobian matrices:

$$\dot{p}_{m_i} = J_P^{m_i} \dot{q}$$

$$w_{m_i} = J_O^{m_i} \dot{q}$$

The Jacobians are calculated as,

$$J_P^{m_i} = \begin{bmatrix} J_{P_1}^{m_i} \dots & J_{P_{i-1}}^{m_i} & 0 \end{bmatrix} \tag{3.13}$$

$$J_O^{m_i} = \begin{bmatrix} J_{O_1}^{m_i} \dots & J_{O_i}^{m_i} & 0 \end{bmatrix} \tag{3.14}$$

The Jacobian matrices in these definitions can be calculated as,

$$J_{P_j}^{m_i} = \begin{cases} Z_{j-1} & prismatic\ joint \\ Z_{j-1} \times (p_{m_i} - p_{j-1}) & revolute\ joint \end{cases}$$

$$J_{O_j}^{m_i} = \begin{cases} J_{O_j}^{l_i} & prismatic\ joint \\ k_{r_i} Z_{m_i} & revolute\ joint \end{cases}$$

Then the kinetic energy formula for motors can written in terms of the Jacobian matrices:

$$K_{m_i} = \frac{1}{2} m_{m_i} \dot{q}^T {J_P^{m_i}}^T J_P^{m_i} \dot{q} + \frac{1}{2} \dot{q}^T {J_O^{m_i}}^T R_i I_{m_i}^{m_i} R_{m_i}^T J_O^{m_i} \dot{q} \quad (3.15)$$

The total kinetic energy is calculated by adding the kinetic energy of the links and the motors, which is

$$\mathrm{K} = \frac{1}{2} \dot{q}^T M(q) \dot{q} \quad (3.16)$$

where,

$$M(q) = \sum_{i=1}^{n} (m_{l_i} {J_P^{l_i}}^T J_P^{l_i} + {J_O^{l_i}}^T R_i I_{l_i}^i R_i^T J_O^{l_i} + m_{m_i} {J_P^{m_i}}^T J_P^{m_i} + {J_O^{m_i}}^T R_i I_{m_i}^{m_i} R_{m_i}^T J_O^{m_i})$$

Here, M is an $n \times n$ matrix called the Inertia matrix and is symmetric and positive definite. Here n is number of degrees of freedom and k_r is gear ratio.

This method is also used for finding the total potential energy. The total potential energy is function of the joint positions, which is derived as,

$$\mathrm{U} = -\sum_{i=1}^{n} (m_{l_i} g_0^T p_{l_i} + m_{m_i} g_0^T p_{m_i}) \quad (3.17)$$

where $g_{0\,[3\times1]}$ is gravitational acceleration with respect to the base frame and total potential energy is defined as gravity matrix.

After calculating the elements of the Lagrangian, it is written in the form of an inertia matrix and gravity matrix g(q). Finally, the equations of motion can be derived from the resulting Lagrange equations as:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{3.18}$$

where C is an $n \times n$ matrix called the Coriolis matrix. Every element c_{ij} at row i and column j of the matrix C is defined according to the equation,

$$\sum_{j=1}^{n} c_{ij}\dot{q}_j = \sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}\,\dot{q}_k\dot{q}_j \tag{3.19}$$

where h_{ijk} is,

$$h_{ijk} = \frac{\partial M_{ij}}{\partial q_k} - \frac{1}{2}\frac{\partial M_{jk}}{\partial q_i}$$

In this way, equations of motion are derived for a 5-DOF robotic arm.

3.4 Control of Manipulators

3.4.1 Computed Torque Control

Robotic arms have nonlinear behavior, since the dynamic equations of the arms contain nonlinear functions of the joint positions and velocities. When a controller is included, the resulting closed-loop equation of the system is therefore nonlinear. So, usually motion controllers for robotic arms are nonlinear controllers. However, the computed torque controller (CTC) is an exception to these types of controllers. It makes the closed-loop control equations linear because of its selection of input and output signals. The computed torque controller is a model-based controller, unlike PD and PID controllers. In computed torque control, torque is considered as an input and it is a function of nonlinear terms that contain the mass matrix, Coriolis matrix and gravity matrix.

The computed torque control law is as follows,

$$\tau = M(q)\left[\ddot{q}_d + K_d\dot{\tilde{q}} + K_p\tilde{q}\right] + C(q,\dot{q})\dot{q} + g(q) \tag{3.20}$$

Here $\tilde{q}$ is the joint angular position error, which is the difference between the desired joint angles q_d and the actual joint angles q , and K_d and K_p are symmetric positive definite matrices. The computed torque controller is a model-based controller, since it contains the desired joint angular acceleration $\ddot{q}_d$ and errors in joint angular velocity and angular position, as well as the mass, Coriolis and gravity matrix.

Figure 3.1 shows the block diagram of a computed torque controller for a robotic arm.

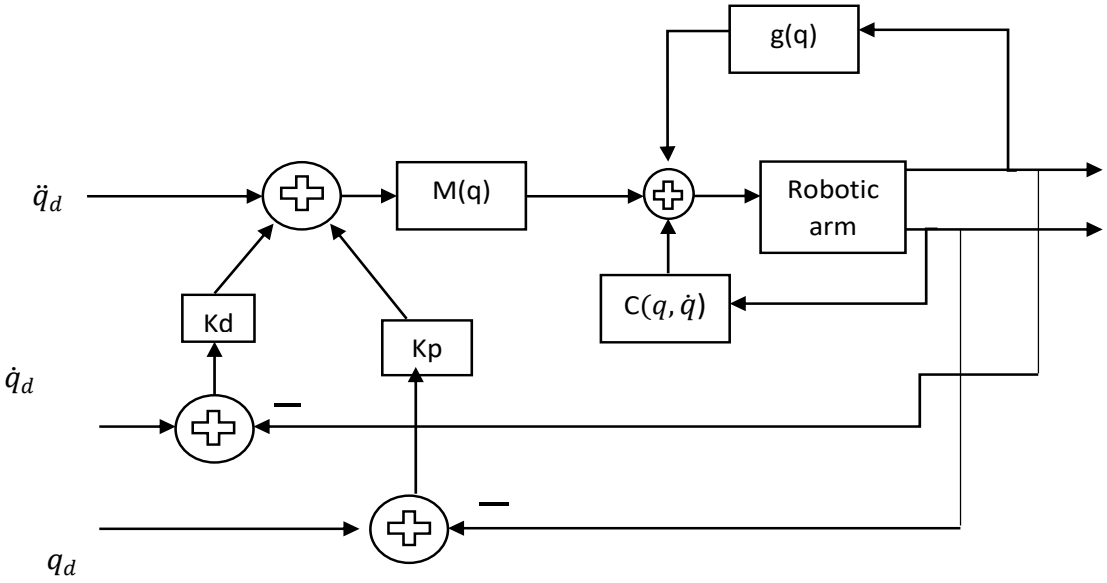

Figure 3.1 Block diagram of computed torque controller

The dynamic equation of a robotic arm is given by equation (3.18). In this equation, τ can be replaced by the control inputs defined in equation (3.20), which gives:

$$M(q)\ddot{q} = M(q)\left[\ddot{q}_d + K_d\dot{\tilde{q}} + K_p\tilde{q}\right] \tag{3.21}$$

The mass matrix has to be a positive definite matrix, and therefore equation (3.21) implies that:

$$\ddot{\tilde{q}} + K_d\dot{\tilde{q}} + K_p\tilde{q} = 0 \tag{3.22}$$

This equation is linear and autonomous. Lyapunov candidate functions exist for this equation, which means that the closed-loop system is asymptotically stable, i.e.

$$\lim_{t\to\infty} \dot{\tilde{q}}(t) = 0 \text{ and } \lim_{t\to\infty} \tilde{q}(t) = 0$$

CHAPTER 4

TRAJECTORY PLANNING WITH OPTIMIZED ENERGY CONSUMPTION

This chapter presents a mathematical formulation for energy optimization of a 5 degree-of-freedom robotic manipulator by using a 7^{th}-order time-dependent trajectory equation. The equation of motion of the robot is modelled from its dynamics, as described in Chapter 3. The objective is for the manipulator to reach a target position from an initial position, which are both user-defined, in a specified amount of time. Physical constraints like joint angle limits and joint velocity limits of the manipulator are implemented. The energy optimization method outputs a set of joint angles over a sequence of time steps, which yields an energy-optimized trajectory for the manipulator to follow over the defined time interval.

4.1 Mathematical Modelling of Objective Function

The power P_i consumed by a servo motor i is the product of the torque τ_i produced by the motor and the angular velocity $\dot{q}_i$ of the motor,

$$P_i = \tau_i \dot{q}_i \tag{4.1}$$

The equation of motion of a 5-DOF robot arm is given by equation (3.18),

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

where $\tau = [\tau_1\ \tau_2\ \tau_3\ \tau_4\ \tau_5]'$, M is the mass matrix, C is the Coriolis matrix, g is the gravity matrix, and $q,\ \dot{q},\ \ddot{q}$ are the vectors of the 5 joint angular positions, velocities, and accelerations, respectively.

Since the torques are derived from the mass matrix, Coriolis matrix, and gravity matrix, which are functions of the joint positions and velocities, the torques are also functions of $q, \dot{q}$. As power is a function of torque and joint velocity, it is also function of $q, \dot{q}$.

4.2 Trajectory & Constraints

The path of the robotic manipulator is the locus of joint angles in the joint space that the manipulator must follow for the execution of an assigned path. A path that is parameterized by time is called a trajectory. We use a time-varying polynomial equation to define the trajectory of the end-effector. To ensure smooth movement of the end-effector, a 7^{th}-order polynomial is considered.

- Path equation for all 5 joints:

$$q_1 = x_1t^7 + x_2t^6 + c_1t^5 + d_1t^4 + e_1t^3 + f_1t^2 + g_1t + h_1$$

$$q_2 = x_3t^7 + x_4t^6 + c_2t^5 + d_2t^4 + e_2t^3 + f_2t^2 + g_2t + h_2$$

$$q_3 = x_5t^7 + x_6t^6 + c_3t^5 + d_3t^4 + e_3t^3 + f_3t^2 + g_3t + h_3$$

$$q_4 = x_7t^7 + x_8t^6 + c_4t^5 + d_4t^4 + e_4t^3 + f_4t^2 + g_4t + h_4$$

$$q_5 = x_9t^7 + x_{10}t^6 + c_5t^5 + d_5t^4 + e_5t^3 + f_5t^2 + g_5t + h_5$$

- Velocity equation for joint 1:

$$\dot{q}_1 = 7x_1t^6 + 6x_2t^5 + 5c_1t^4 + 4d_1t^3 + 3e_1t^2 + 2f_1t + g_1$$

- Acceleration equation for joint 1:

$$\ddot{q}_1 = 42x_1t^5 + 30x_2t^4 + 20c_1t^3 + 12d_1t^2 + 6e_1t + 2f_1$$

Similar velocity and acceleration equations can be written for the remaining 4 joints. We substitute in the initial and final conditions of position, velocity and acceleration for the joints, as shown below.

Table 3. Initial and final conditions

At time t = 0	At time t = t_f
$q = q_{0,i}$ where $q_{0,i}$ is initial position of joint i	$q = q_{f,i}$ where $q_{f,i}$ is final position of joint i
$\dot{q} = 0$ for all joints	$\dot{q} = 0$ for all joints
$\ddot{q} = 0$ for all joints	$\ddot{q} = 0$ for all joints

Applying the initial conditions to the equation for q_1,

$$f_1=0, \quad g_1=0 \quad \text{and} \quad h_1=q_{0,1.}$$

Hence, the equation of q_1 is reduced to

$$q_1 = x_1t^7 + x_2t^6 + c_1t^5 + d_1t^4 + e_1t^3 + q_{0,1}$$

Equations for q_2, q_3, q_4 and q_5 are developed in a similar manner. By applying the initial and final conditions to these equations, the corresponding values of c_i, d_i, e_i, f_i, g_i and h_i are found.

Nonlinear constraints on q_1 through q_5 are imposed as follows. The units are radians.

$$-3.14 \leq q_1 \leq 3.14$$

$$-1.57 \leq q_2 \leq 1.57$$

$$-1.57 \leq q_3 \leq 1.57$$

$$-1.57 \leq q_4 \leq 1.57$$

$$-3.14 \leq q_5 \leq 3.14$$

Nonlinear constraints on $\dot{q}_1$ through $\dot{q}_5$ are imposed as follows. The units are radians/sec.

$$-2.26 \leq \dot{q}1 \leq 2.26$$

$$-2.26 \leq \dot{q}2 \leq 2.26$$

$$-2.26 \leq \dot{q}3 \leq 2.26$$

$$-2.26 \leq \dot{q}4 \leq 2.26$$

$$-2.26 \leq \dot{q}5 \leq 2.26$$

In this formulation, we use time as a parameter and not as a variable. The variables x_1 through x_{10} are optimized and the above constraints are imposed on the q_i and $\dot{q}i$. The values of x_1 through x_{10} obtained from the optimization procedure are substituted into the path equation and are checked to confirm that the constraints are satisfied.

In the table below, $q_{0,i}$ is the user-defined initial position of the joint angle q_i and $q_{f,i}$ is the joint angle q_i at the desired final position.

Table 4. Initial and Final Conditions (in radians)

	q_1	q_2	q_3	q_4	q_5
$q_{0,i}$	0.7071	1.7071	1.7071	1.7071	1.7071
$q_{f,i}$	1.7070	1.2217	1.2217	1.2217	1.2217

4.2 Optimization Results

Total power consumption ΣP_i is minimized in this optimization problem. The 'fmincon' optimization function in MATLAB is used with the 'SQP' (Ren 2018) algorithm. To initiate the optimization, the initial guesses of the variables used are:

Table 5. Initial guesses of the variables

x_1= 5*1e-5	x_3=9*1e-5	x_5=5*1e-5	x_7=6*1e-5	x_9=6*1e-5
x_2=2*1e-4	x_4=7*1e-4	x_6=1.5*1e-4	x_8=1*1e-4	x_{10}=1*1e-4

The optimization was performed for 7 different final times t_f. The optimized values for x_1 through x_{10} and the power consumption calculated from these variables are shown for each final time in Table 6. It is evident from the table that the minimum value of power consumption is achieved for t_f = 2.5 s. Figure 4.22 plots the values of power versus the final time t_f.

Table 6. Optimization Results

t_f (s)	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	**Power (Watts)**
1	1E-05	-0.01	0.104	0.02	1E-05	10	1E-05	10	1E-05	10	**1.5543**
1.5	0.064	-0.04	0.005	0.004	0.047	7.935	0.036	5.82	1E-05	4.93	**1.9**
2	1E-05	0.03	1E-04	1E-05	7E-03	1.78	6E-03	1.42	1E-05	1.19	**0.7765**
2.5	1E-05	-0.12	2E-04	1E-05	3E-04	0.189	3E-04	0.18	1E-05	0.20	**0.3198**
3	1E-05	0.05	8E-05	0.034	0.005	0.220	0.004	0.17	1E-05	0.16	**0.8599**
4	2E-05	0.02	3E-05	0.034	3E-05	0.030	3E-05	0.03	1E-05	0.02	**2.6165**
5	1E-05	0.09	1E-05	1E-05	1E-05	0.012	1E-05	0.01	1E-05	0.01	**1.361**

Visually, the trajectory of the end-effector over a time span of 1 s looks straight. However, in that case the power consumption is higher compared to the power for a time span of 2.5 s. This could be due to the fact that the joint velocities for the case where t_f = 1 s are relatively high(Fig.4.12).

- *For t_f = 5 seconds*

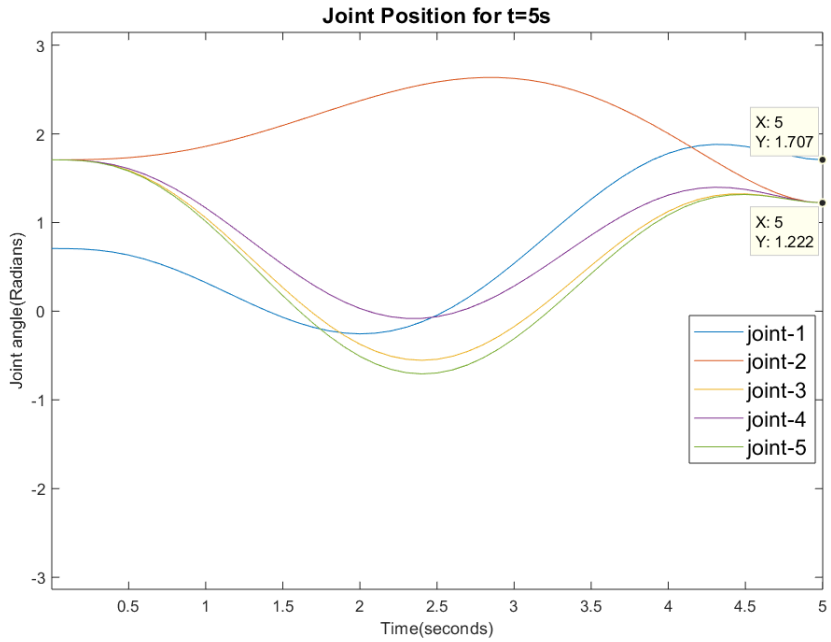

Figure 4.1 Joint Position for t = 5 seconds

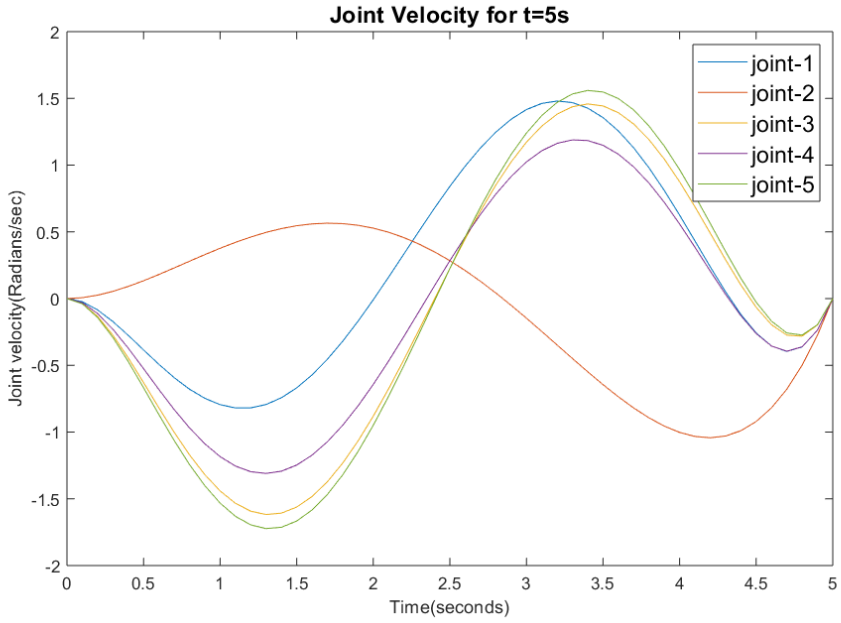

Figure 4.2 Joint velocity for t = 5 seconds

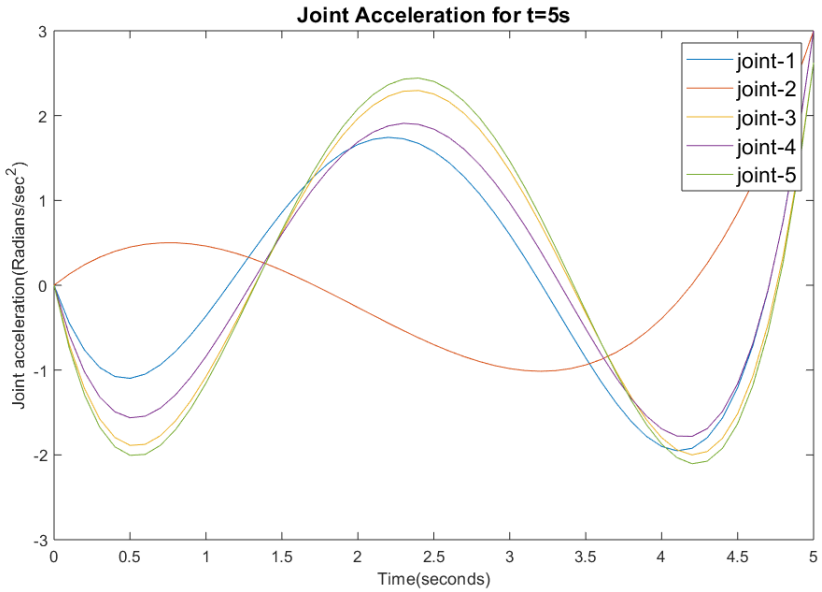

Figure 4.3 Joint Acceleration for t = 5 seconds

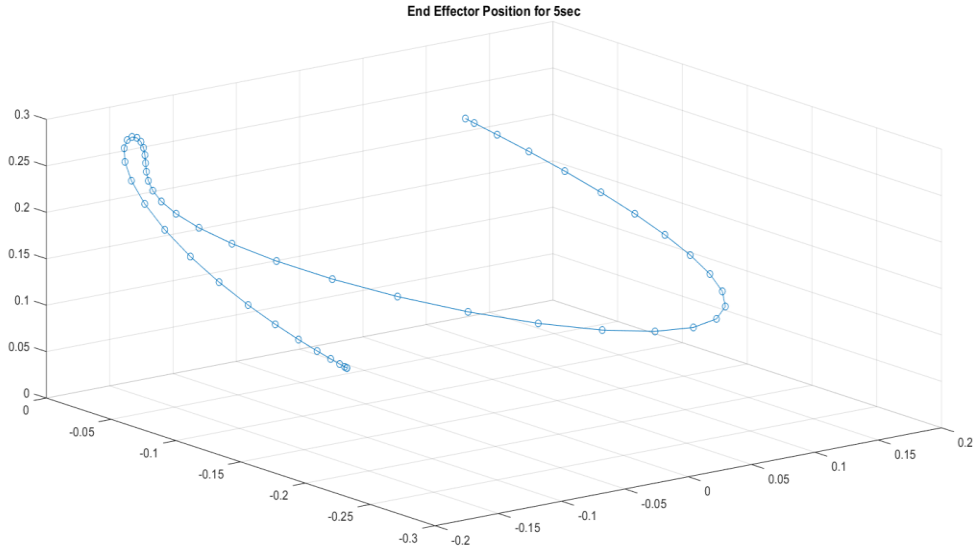

Figure 4.4 End-effector trajectory for t = 5 seconds

- *For t = 3 seconds*

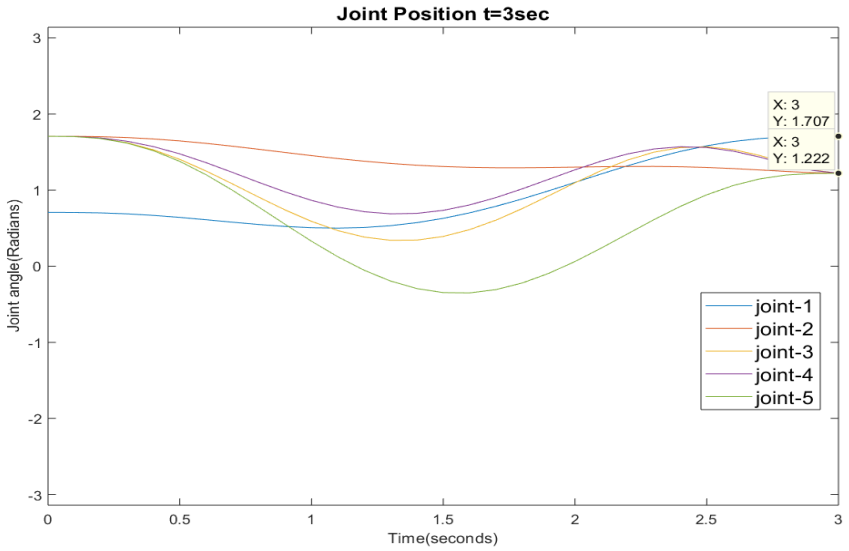

Figure 4.5 Joint position for t = 3 seconds

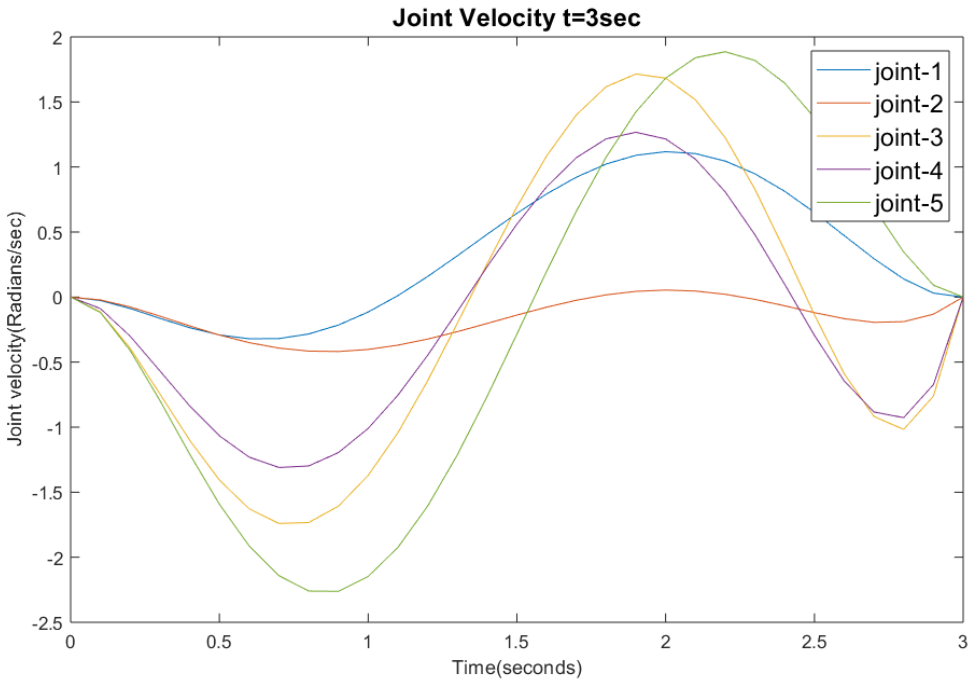

Figure 4.6 Joint velocity for t = 3 seconds

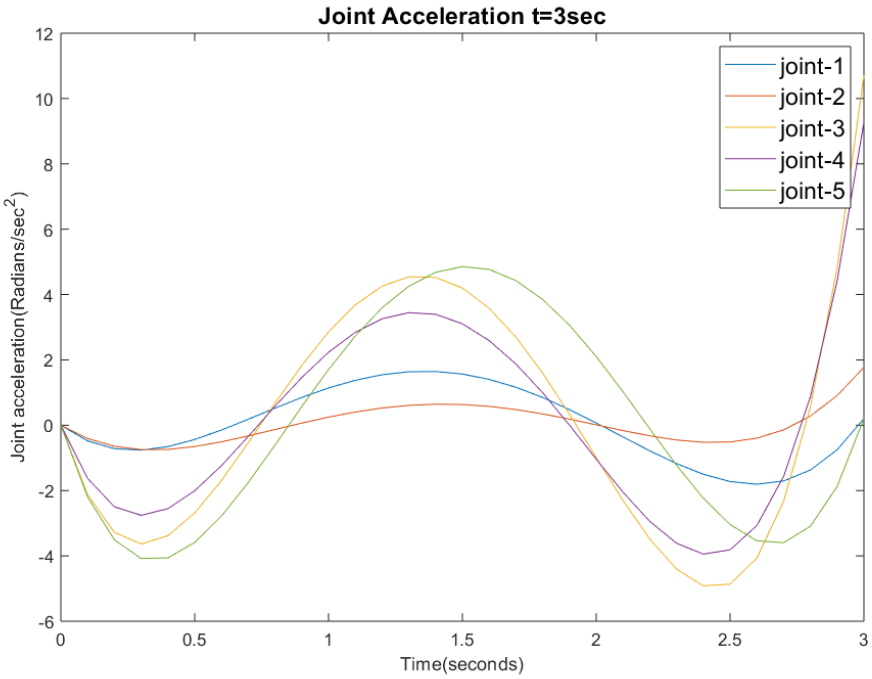

Figure 4.7 Joint acceleration for t = 3 seconds

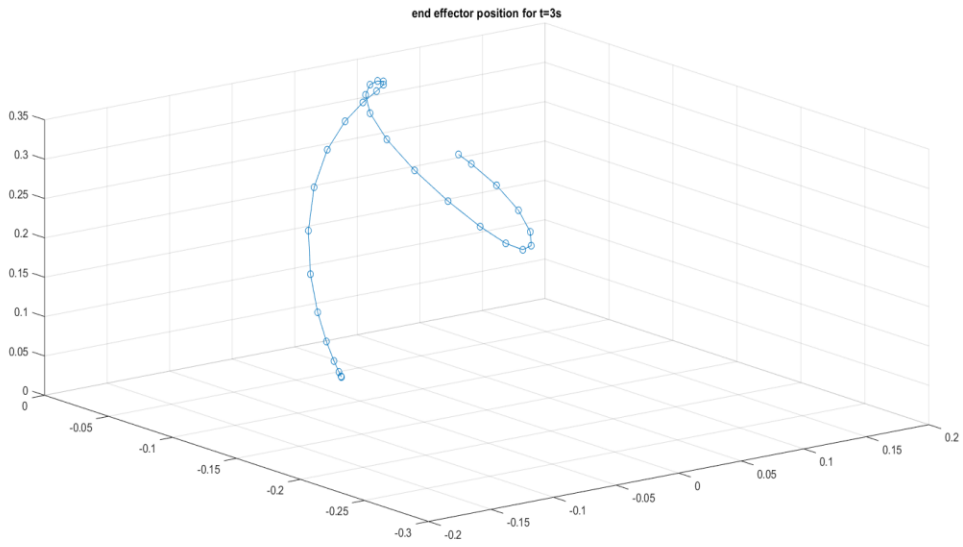

Figure 4.8 End-effector trajectory for t = 3 seconds

- *For t = 2 seconds*

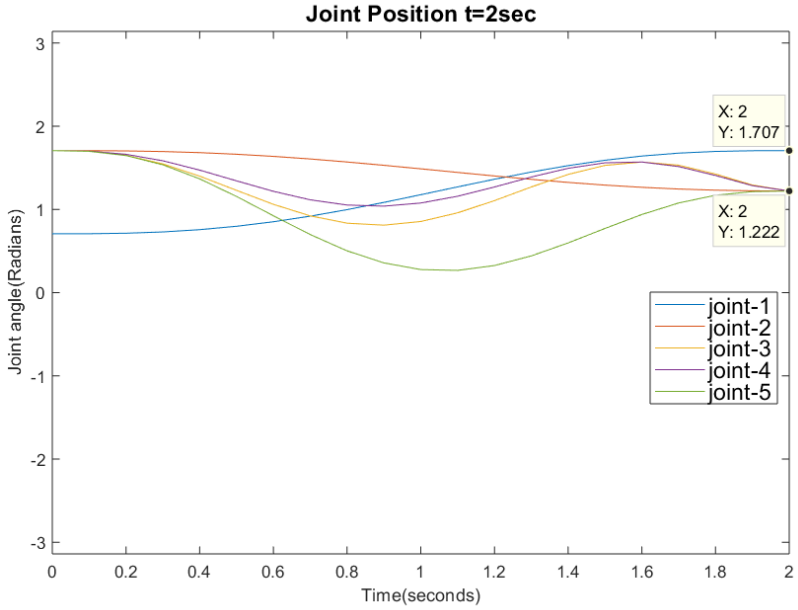

Figure 4.9 Joint position for t = 2 seconds

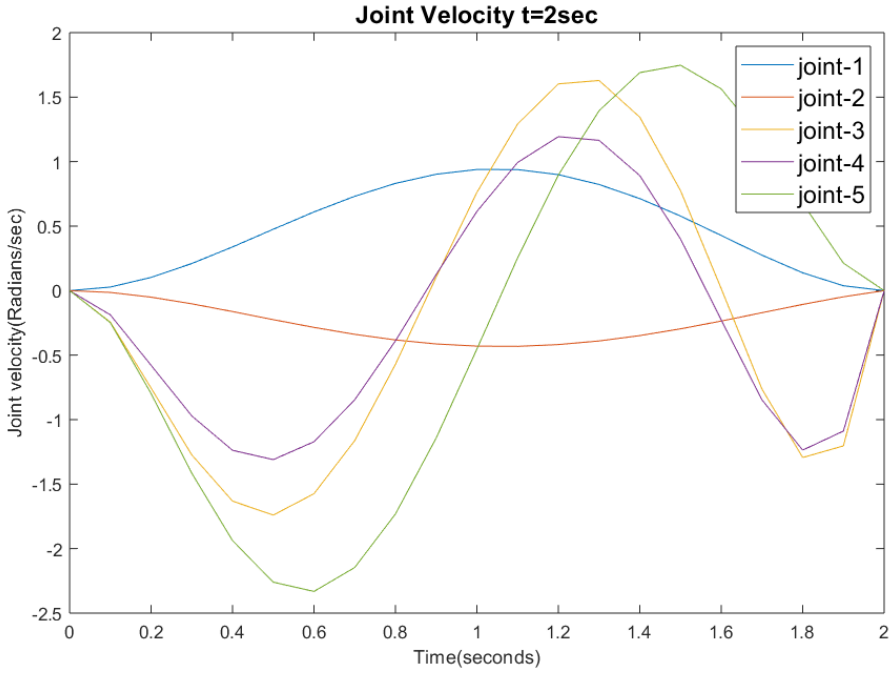

Figure 4.10 Joint velocity for t = 2 seconds

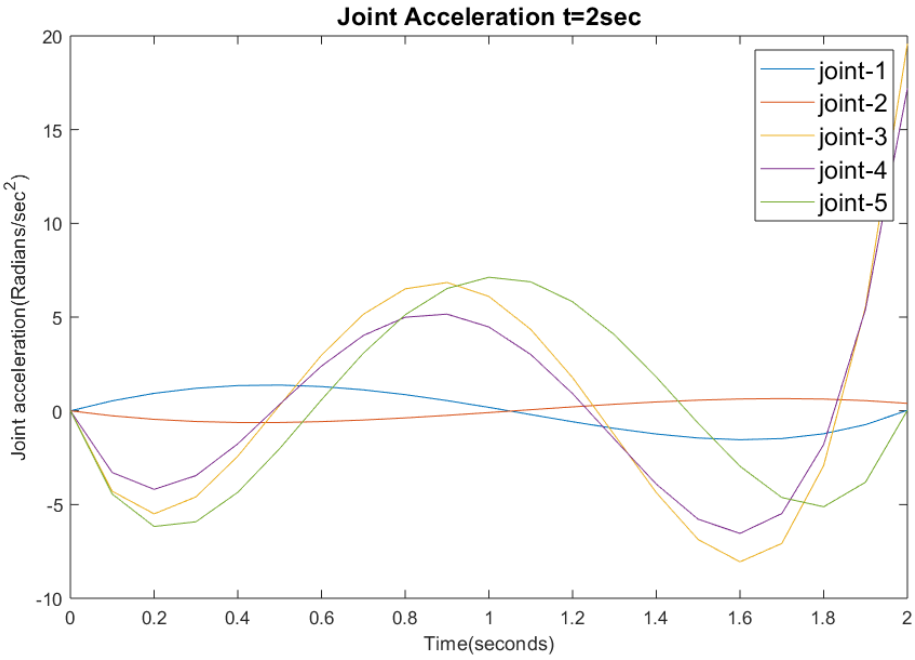

Figure 4.11 Joint acceleration for t = 2 seconds

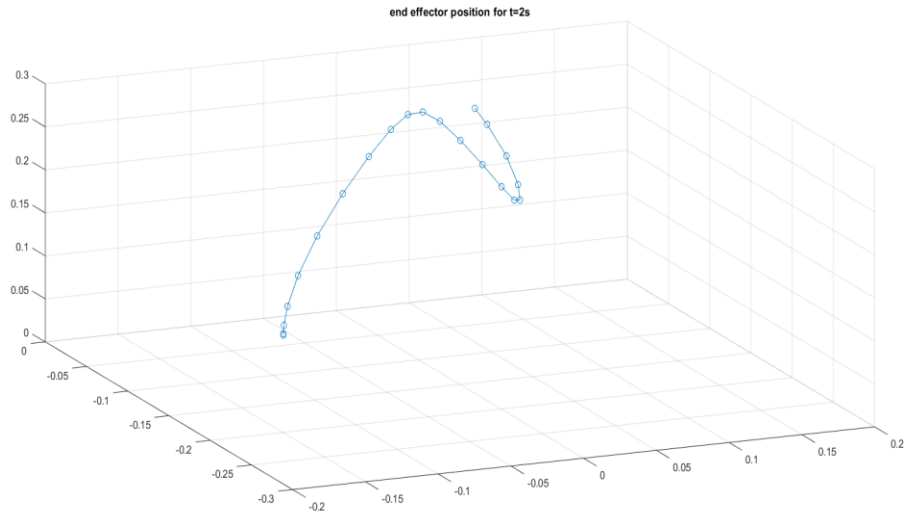

Figure 4.12 End-effector trajectory for t = 2 seconds

➢ *For t = 1 second*

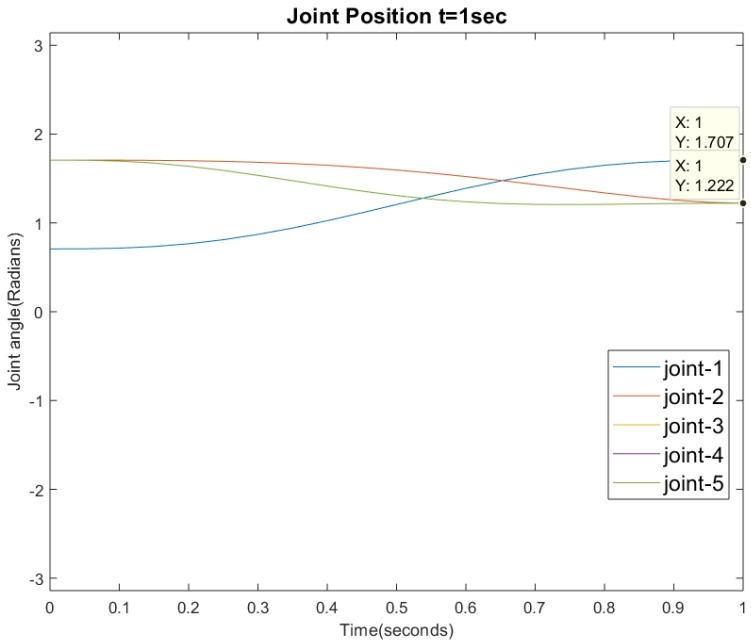

Figure 4.13 Joint position for t = 1 second

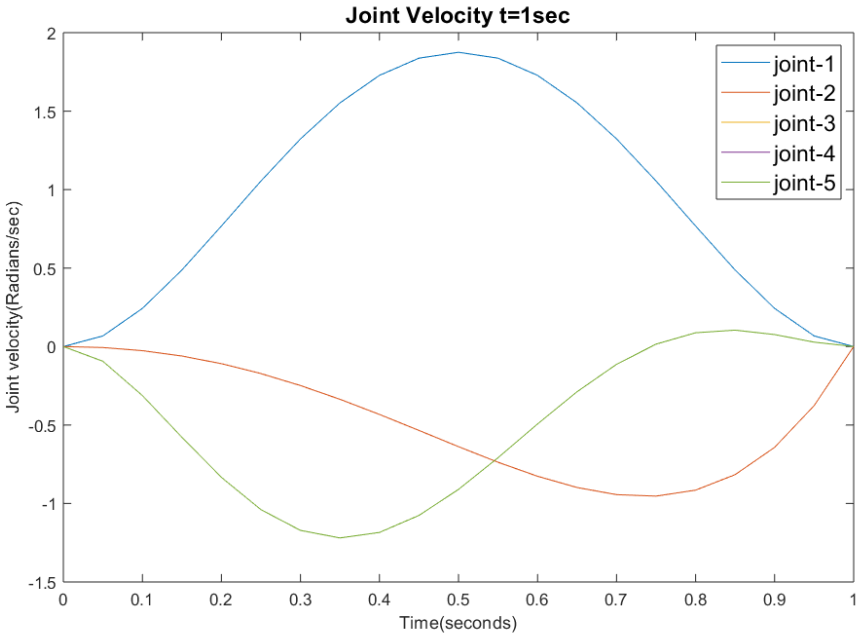

Figure 4.11 Joint velocity for t = 1 second

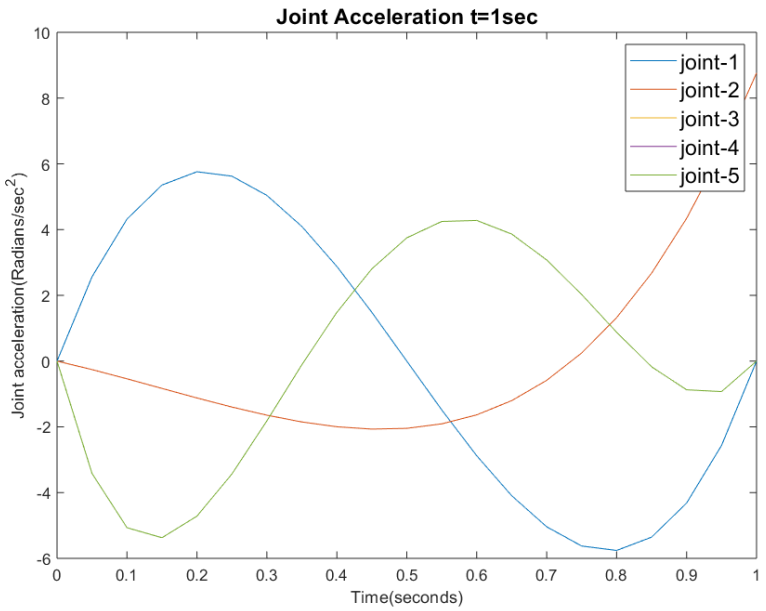

Figure 4.15 Joint acceleration for t = 1 second

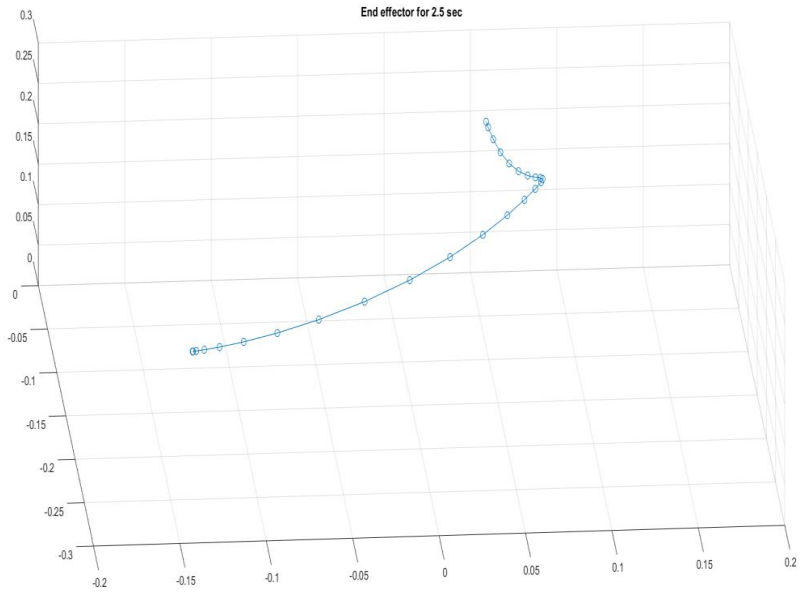

Figure 4.16 End-effector trajectory for t = 1 second

➢ *For t = 2.5 seconds*

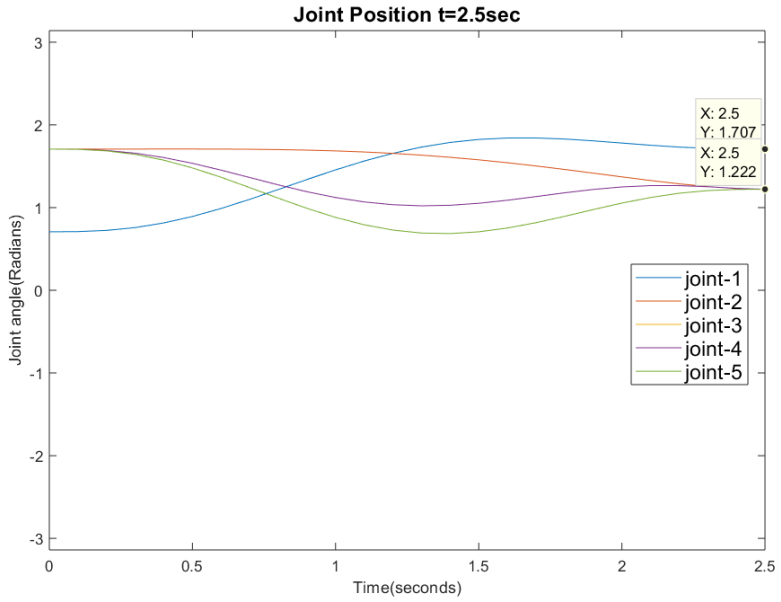

Figure 4.17 Joint position for t = 2.5 seconds

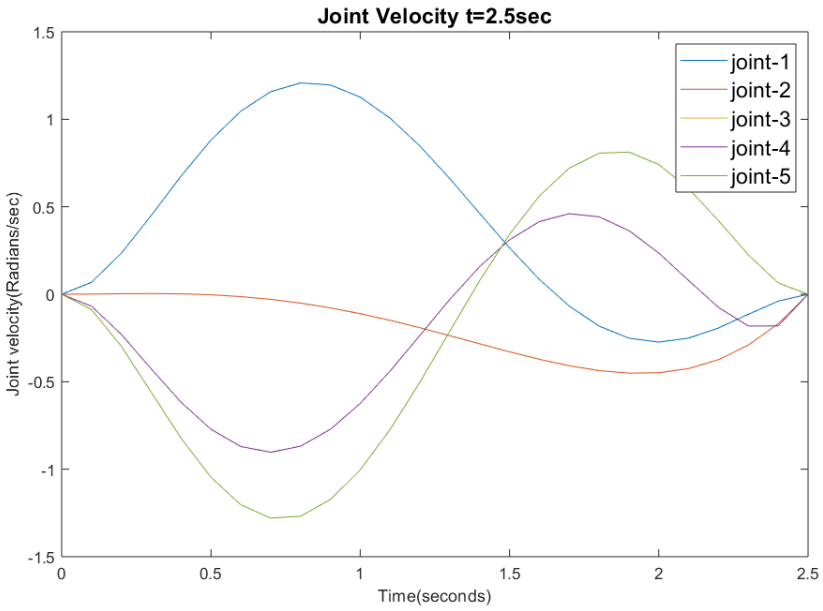

Figure 4.18 joint velocity for t = 2.5 seconds

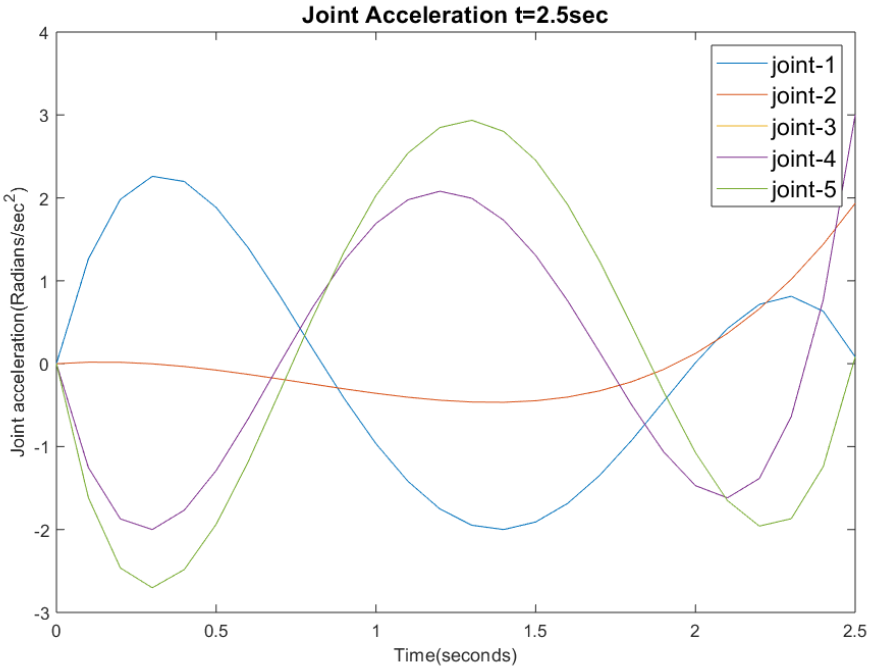

Figure 4.19 Joint acceleration for t = 2.5 seconds

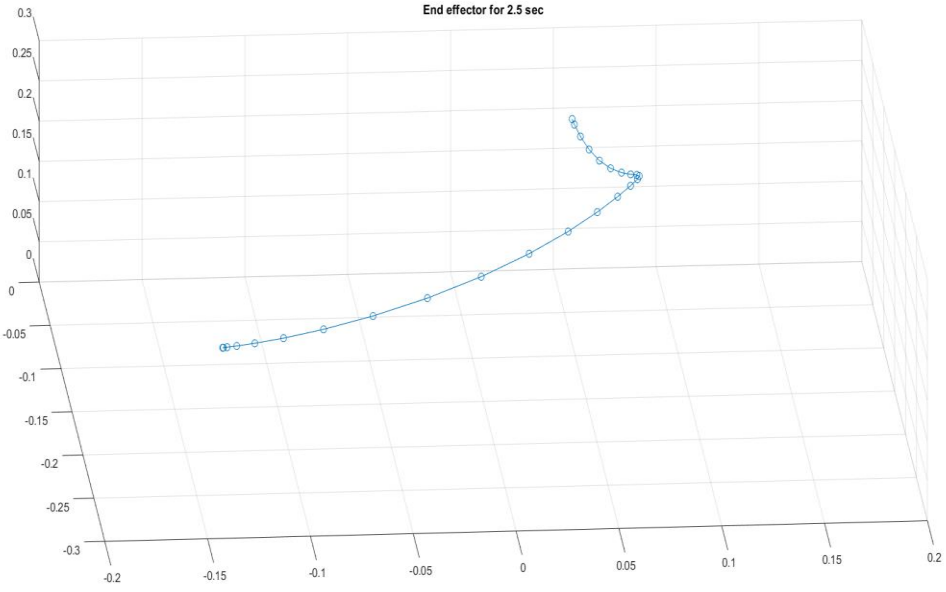

Figure 4.20 End-effector trajectory for t = 2.5 seconds

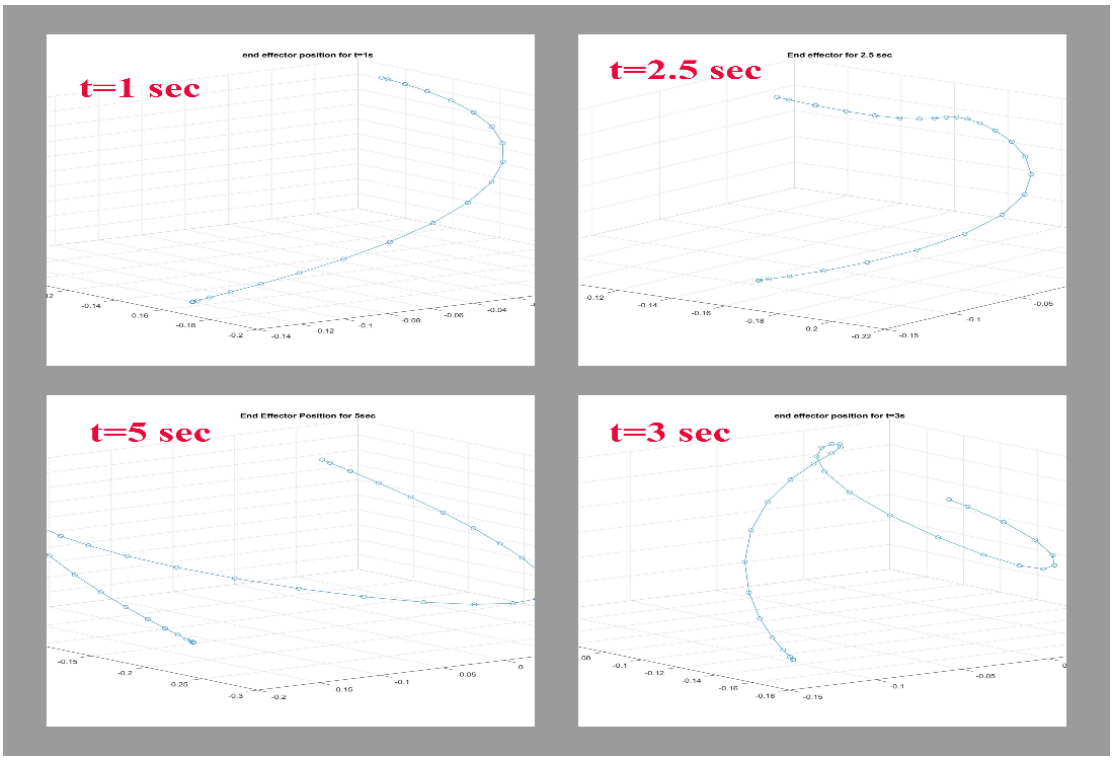

Figure 4.21 Comparison of End-effector Trajectories for the same Initial and Final Positions

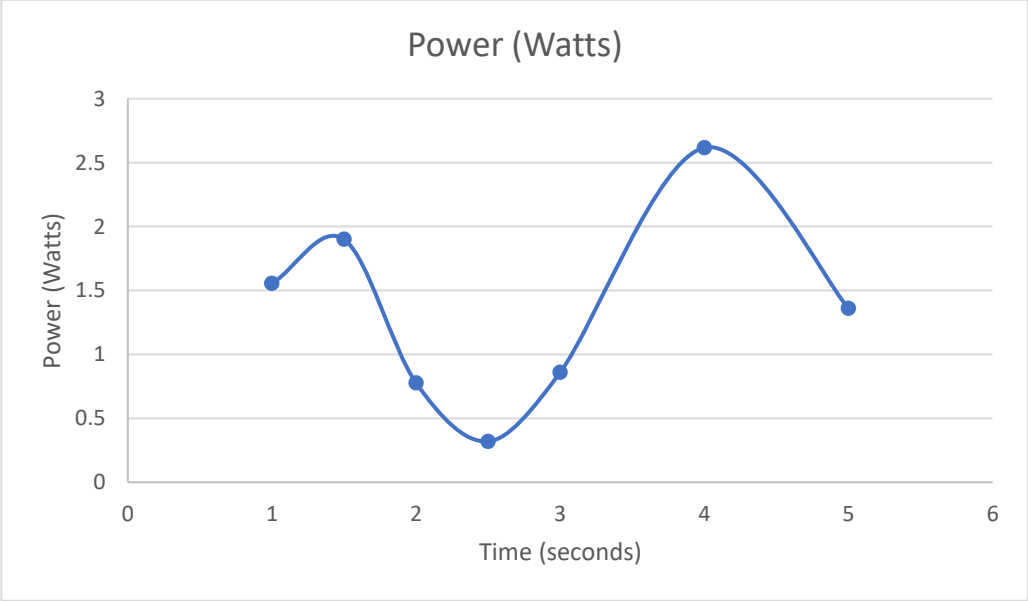

Figure 4.22 Power vs. Time

CHAPTER 5

SOFTWARE FRAMEWORK FOR CONSTRUCTION TESTBED

5.1 ROS Compatible Robotic Arm

ROS (Robot Operating System) is a robot software development platform. It has a huge user community, which makes ROS a preferred platform for roboticists. It has users from all over the world and because of that, robotics companies are inclining towards using ROS. Due to the following properties, many users prefer ROS over other robotics platforms:

- Feature-rich packages: ROS has various high-capability packages for different applications, including for autonomous navigation of mobile robots and motion planning of robotic arms. These features and packages have numerous applications, and code can be modified to create new applications for robots.
- Various functions: ROS has many functionalities for visualization, debugging and simulation. Rviz and Gazebo are widely used simulators in the ROS community. The rqt tool is used for visualization of control outputs, joint behaviors and feedback properties of systems.
- Compatible with sensors and actuators: ROS is compatible with various sensors and actuators. It is easy to communicate with these components via ROS. The Dynamixel servos used in the PhantomX Robotic Arm are also compatible with ROS.
- Different coding languages: ROS operations are based on nodes. The nodes can be programmed in many languages like Python, Matlab, C and C++. Just because of that no other platform is as varied as ROS.

- Modularity: ROS is comparatively more modular than other robotic platforms. The ROS platform is conceptualized based on node coding for different activities; if one node crashes, then other nodes and other activities can still work.
- Dynamic community: The ROS community is developing very rapidly, and is very active and helpful to new ROS users. The innovative applications are shared throughout the community by users. Problems regarding ROS are solved on a ROS wiki page.

5.2 ROS Structure

ROS is a highly structured platform, like other operating systems. ROS has 3 main levels: file system, computational graph and community level. The ROS file system has the following folders: Meta-Packages, Packages, and Messages-service-action-nodes. Here, packages are the most basic element of the ROS platform. ROS packages have configuration files, launch files, scripts, package manifest files and a CMake build file.

The package folder is very important for ROS setup, and some of package sub-folders are described here.

- Config: All configuration files are stored in this folder. Users can assign parameters of robot controllers, joints, and joint limits under configuration files.
- Launch: All launch files are stored in this folder. The nodes for different applications are launched by these files.
- Scripts: The users can store Python scripts in this folder. These executable scripts help to implement functionalities for nodes.
- Src: All source files are stored in this file. The file must be written in C++.

- Msg: All message definitions are stored in this folder.
- Srv: All services are stored in this folder.
- Package.xml: This is the package manifest file of the package.
- CMakeLists.txt: This is the CMake build file of the package.

5.3 Modeling of Robotic Arm in ROS

To validate robot controllers, 3D modeling of robots in software is very important. The modeling of robots saves a great deal of time and money that would be spent on physical robot experiments. There are some packages in ROS which assist in modeling of robots. In this work, modeling of the PhantomX robotic arm is done using a URDF (Unified Robot Description Format) package, a joint state publisher and a robot state publisher. The URDF is the XML file format for describing the robot model.

The joint state publisher package is very important for modeling the robot. The joint state publisher node publishes joint values to nonfixed joints from reading URDF files. This way, it makes it easy for users to understand URDF files. The robot state publisher is also a useful package for publishing 3D poses of the robot. It publishes each robot link in a 3D world using kinematics described in the URDF files. The TF nodes represents relations between coordinates of robot reference frames.

The URDF file of any robot contains joint and link information. The link tag contains properties like size, shape, and geometry, and it can also include mesh properties of the link. The link also has some dynamic properties like inertia and collision of links.

The different types of joints, such as revolute, prismatic, and fixed, are defined under joint tags. The joint limits, velocity limits, and effort limits are also defined under joint

tags. The URDF file creates a 3D model from declarations of joint and link tags. The interface of the hardware with ROS is also defined in URDF files. Simulators like Gazebo and Rviz read URDF files and create 3D models for users.

In this work, a 3D model of the 5-DOF PhantomX Reactor robotic arm was created in the Gazebo and Rviz simulators. The files are on the GitHub page of the Autonomous Collective Systems (ACS) Laboratory (Lab 2019).

5.4 Motion Planning with Moveit!

MoveIt has different packages for manipulation of a robotic arm. It is possible to solve problems of motion planning, pick and place, grasping, and so on with different plugins provided by MoveIt APIs. It has library plugins for manipulation, collision checking, motion planning, control and perception. The customized robots are also compatible with MoveIt with the help of the MoveIt Setup Assistant.

The MoveIt platform is centralized around a move-group node. It connects various services and actions of different applications with robot. The move-group node subscribes to the joint state and the TF topic of the robot. It gathers information about the robot from URDF and configuration files provided by the user.

The user can work with MoveIt in 3 different ways. The user can command the move-group node via C++ APIs, Python APIs or the GUI of MoveIt. The GUI is compatible with the Rviz simulator, in which the user can add plugins associated with their requirements. It is also possible to interlink MoveIt-Rviz with a real robot or Gazebo for experiments.

Below, the MoveIt architecture diagram is shown to illustrate Moveit programming.

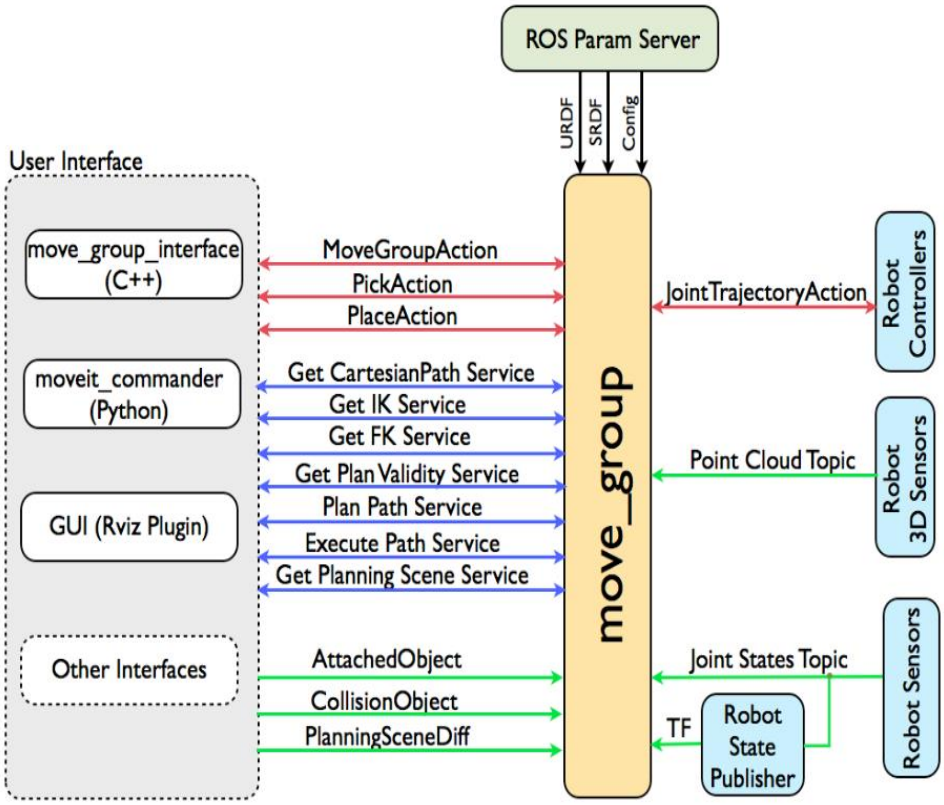

Figure 5.1 MoveIt architecture diagram (Figure borrowed from MoveIt-documentation)

As shown in Figure 5.1, the move-group node collects values from robot sensors and controllers. It processes path planning, collision checking, scene planning and grasping. It operates those functionalities when an action or service is called by the user in the form of Rviz plugins, Python moveit-commander or move-group-interface.

MoveIt uses the Open Motion Planning Library (OMPL) for path planning of robotic arms. The OMPL has several algorithmic plugins for motion planning, so users can change

algorithms in the MoveIt GUI as per their applications. By default, MoveIt uses RRT (Rapidly-exploring Random Trees) and RRT* (optimized RRT) plugins for manipulation.

In this work, the PhantomX Reactor Robotic Arm (Phantomx) was made compatible with MoveIt with the use of the setup assistant and the URDF file of the robot. Motion planning was also performed to obtain the desired trajectories of the robotic arm that were computed using the optimization procedure described in Chapter 4. The PhantomX robot was also inter-linked with Gazebo and physical hardware to test MoveIt package. The files for MoveIt and Gazebo simulations are on the GitHub page of the Autonomous Collective Systems (ACS) Laboratory (Lab 2019).

CHAPTER 6

SIMULATION AND EXPERIMENTAL RESULTS

6.1 Simulation of Computed Torque Controller

The computed torque controller is a model-based controller for robotic arms. As mentioned in Chapter 3, the control law for the CTC is given by:

$$\tau = M(q)\left[\ddot{q}_d + K_d\dot{\tilde{q}} + K_p\tilde{q}\right] + C(q,\dot{q})\dot{q} + g(q)$$

This controller makes the nonlinear robot dynamics into a second-order linear system:

$$\ddot{\tilde{q}} + K_d\dot{\tilde{q}} + K_p\tilde{q} = 0$$

Here, this second-order system is solved using the ODE45 command in Matlab software. To make the system solvable using ODE45, the second-order system can be converted into two first-order differential equations, written in state-space form as follows:

$$\frac{d}{dt}\begin{bmatrix}\tilde{q}\\ \dot{\tilde{q}}\end{bmatrix} = \begin{bmatrix}\dot{\tilde{q}}\\ -K_d\dot{\tilde{q}} - K_p\tilde{q}\end{bmatrix}$$

$$\frac{d}{dt}\begin{bmatrix}\tilde{q}\\ \dot{\tilde{q}}\end{bmatrix} = \begin{bmatrix}0 & I\\ -K_p & -K_d\end{bmatrix}\begin{bmatrix}\tilde{q}\\ \dot{\tilde{q}}\end{bmatrix}$$

The joint positions and velocities are unknown in these equations, and because of that they are considered as symbolic variables in Matlab. The Matlab code for the computed torque controller is on the GitHub page of the Autonomous Collective Systems (ACS) Laboratory (Lab 2019). The proportional and velocity gain matrices are defined as diagonal matrices to make this centralized system decentralized.

After trial and error we set $K_p = K_p1 = \text{diag}\{1.8, 0.65, 2, 0.7, 2.5\}$ and $K_d = K_d1 = \text{diag}\{10, 2, 10, 2, 10\}$. Figures 6.1, 6.2, and 6.3 show the joint positions, velocities, and

error values over time, respectively. These are the outputs from the integration of the robot dynamics with the computed torque controller in Matlab. It is seen in Figure 6.3 that the error values all converge to zero for the given K_p1 and K_d1 matrices.

Next, we set $K_p = K_p2$ = diag {14.8, 7.65, 15, 6.7, 20.5} and $K_d = K_d2$ = diag {10, 2, 10, 2, 10}; i.e., the gains in the K_p matrix are increased. As Figures 6.4-6.6 show, the manipulator has a faster response but exhibits jerkier motion.

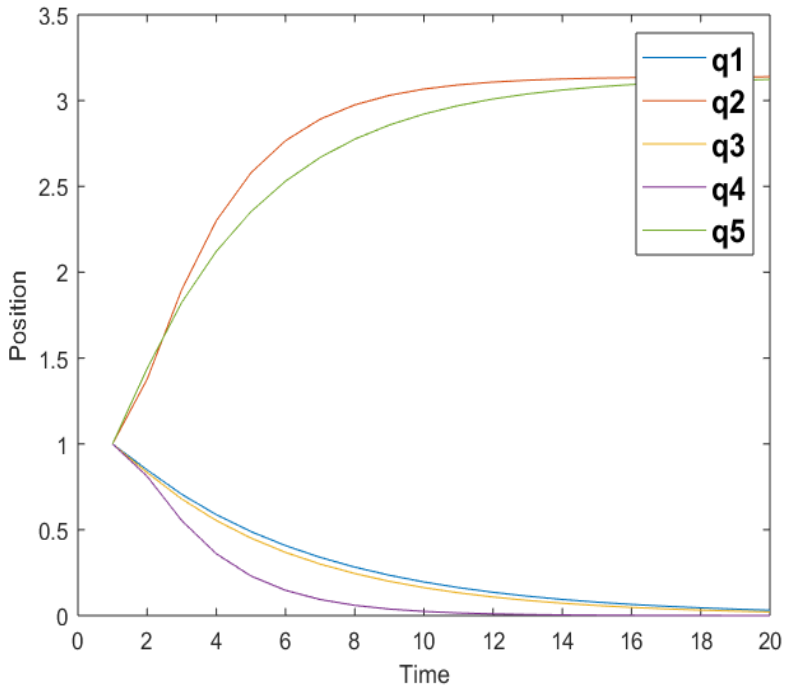

Figure 6.1 Position-Time graph for K_p1 and K_d1(radian-second)

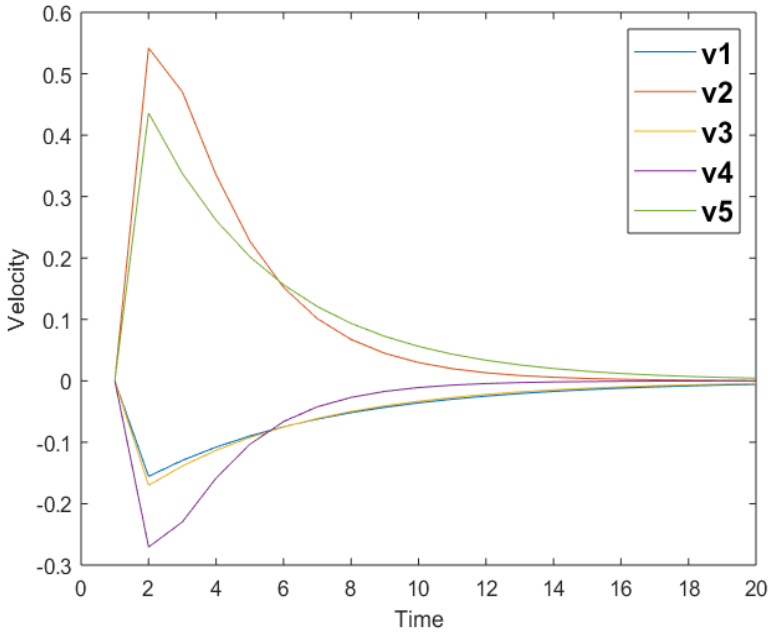

Figure 6.2 Velocity-Time graph for K_p1 and K_d1(radian/sec-second)

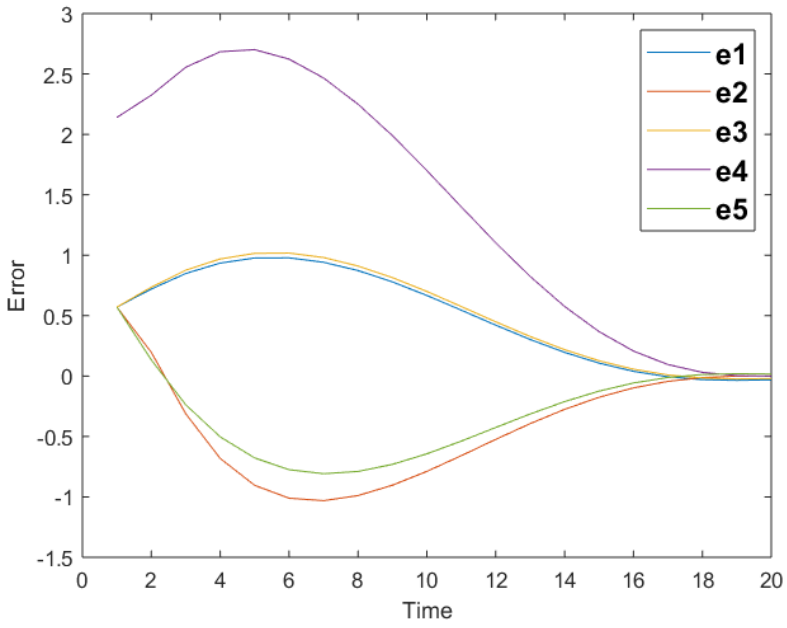

Figure 6.3 Error-Time graph for K_p1 and K_d1 (radian-second)

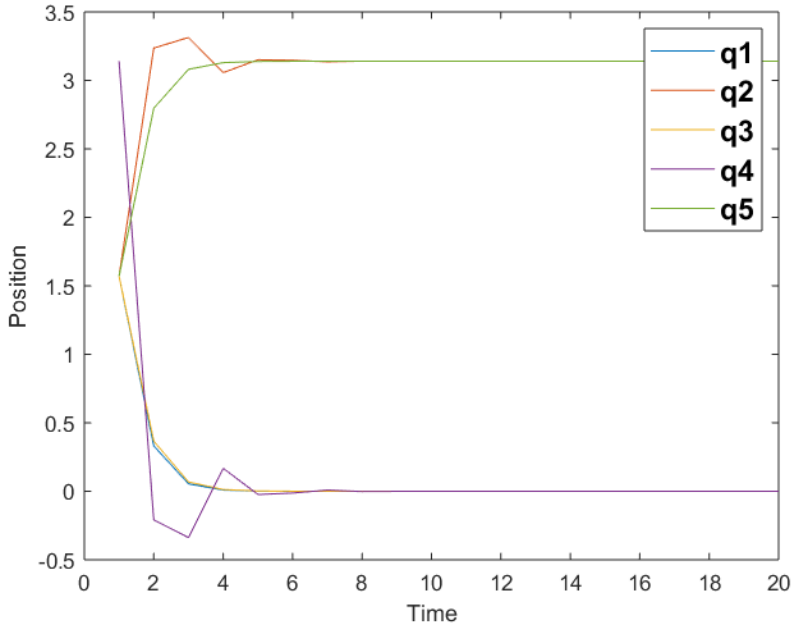

Figure 6.4 Position-Time graph for K_p2 and K_d2(radian-second)

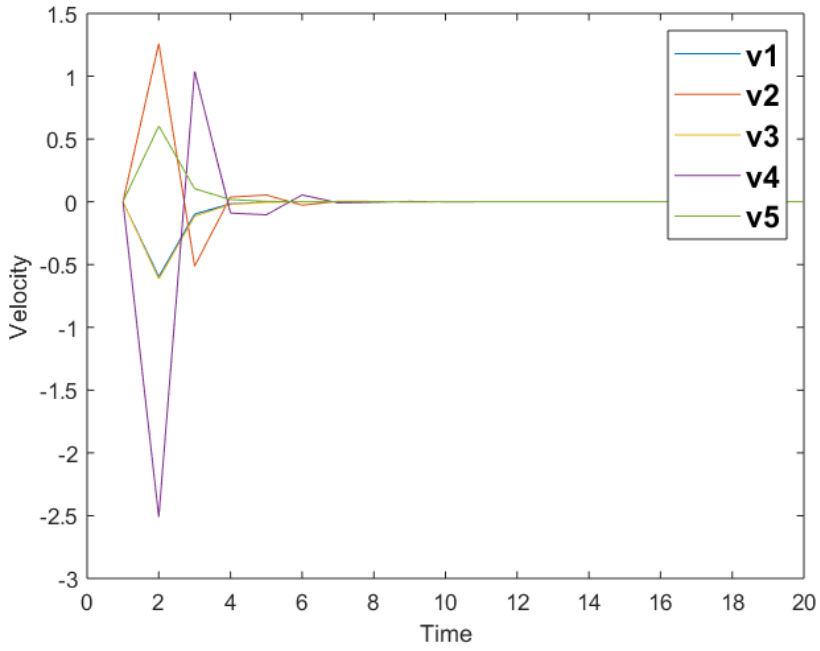

Figure 6.5 Velocity-Time graph for K_p2 and K_d2(radian/sec-second)

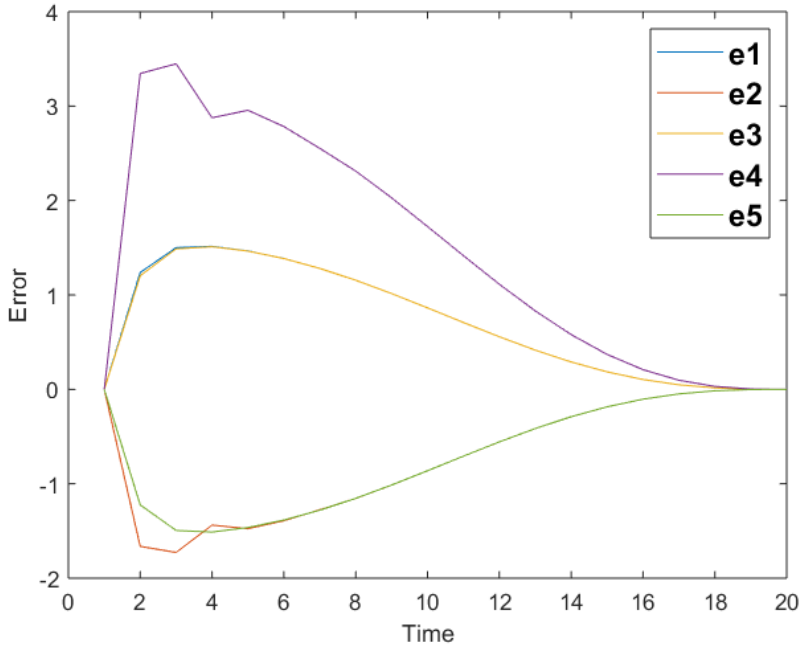

Figure 6.6 Error-Time graph for K_p2 and K_d2(radian-second)

6.2 Experiment Setup

The commercially available 5-DOF PhantomX Reactor Robotic Arm was used here for experiments. It has a static base, and it is possible to make it mobile by attaching it to a mobile robotic platform. The Dynamixel AX-12a servo motors are used in the PhantomX Robotic Arm. These servos have position and velocity feedback, while inputs are only possible to achieve position control. These servos are compatible with ROS, so it is possible to command them from an external computer. The PhantomX robot was modeled in MoveIt using the setup assistant. The setup assistant software is shown in Figure 6.7.

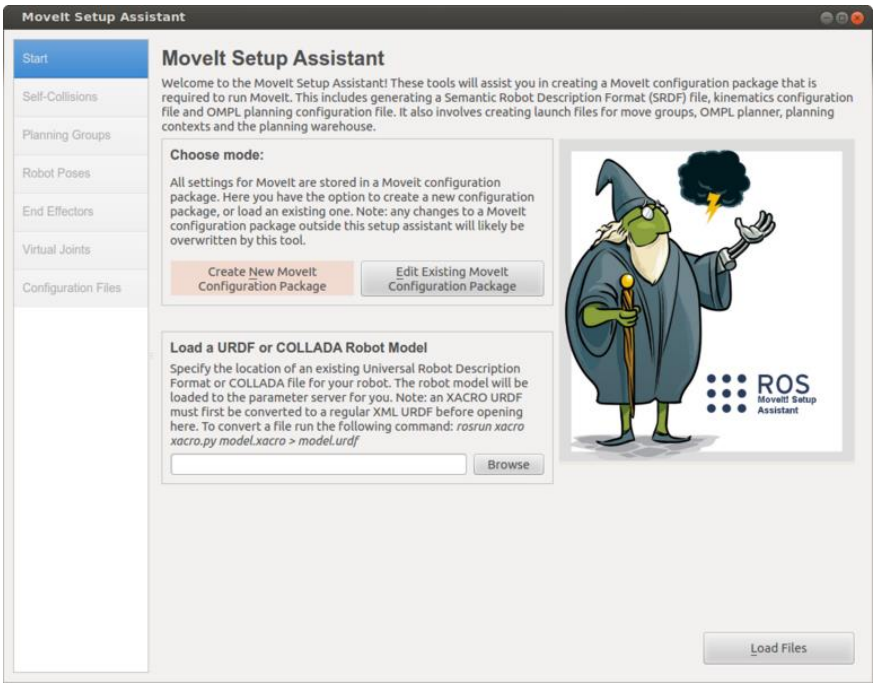

Figure 6.7 MoveIt Setup Assistant (Moveit! n.d.)

After attaching the URDF files of the PhantomX robot, the end-effector configurations and control methods are chosen as per the required applications. Then after setting up the PhantomX robot in MoveIt, it is possible to plan its path with the RRT algorithm. This planned trajectory is commanded to the physical hardware, and the Dynamixel servos attempt to move the robot according to the trajectory generated in MoveIt.

For the MoveIt and physical robot coordination, some files must be modified to achieve similar controllers in MoveIt and Gazebo / the physical robot. The MoveIt simple manager has to be called in a launch file for initiation of the MoveIt controllers. The configure file is required to start controllers in Gazebo. Finally, launching Gazebo and MoveIt will initiate controllers in the computer as well as on the real hardware. In this way, it is possible to coordinate Gazebo and the physical robot with MoveIt. The required files are on the

GitHub page of the Autonomous Collective Systems (ACS) Laboratory (Lab 2019). The MoveIt motion planning GUI is shown in Figure 6.8.

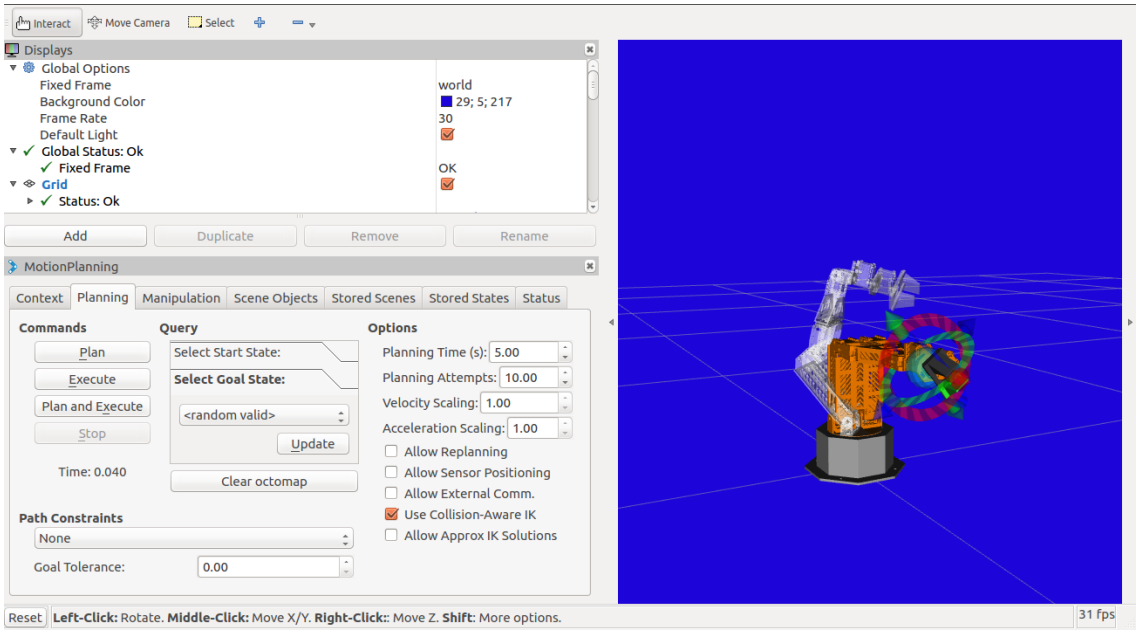

Figure 6.8 MoveIt motion planning GUI in Rviz (Initial robot configuration in Orange and final configuration in white)

6.3 Matlab-ROS interface

Matlab has introduced a robotics toolbox, which can connect Matlab to ROS. The benefit of the ROS-Matlab interface is that the user does not need to code complicated controllers and perception algorithms in c++ coding language; they can directly use Matlab code to send commands and subscribe to the robot.

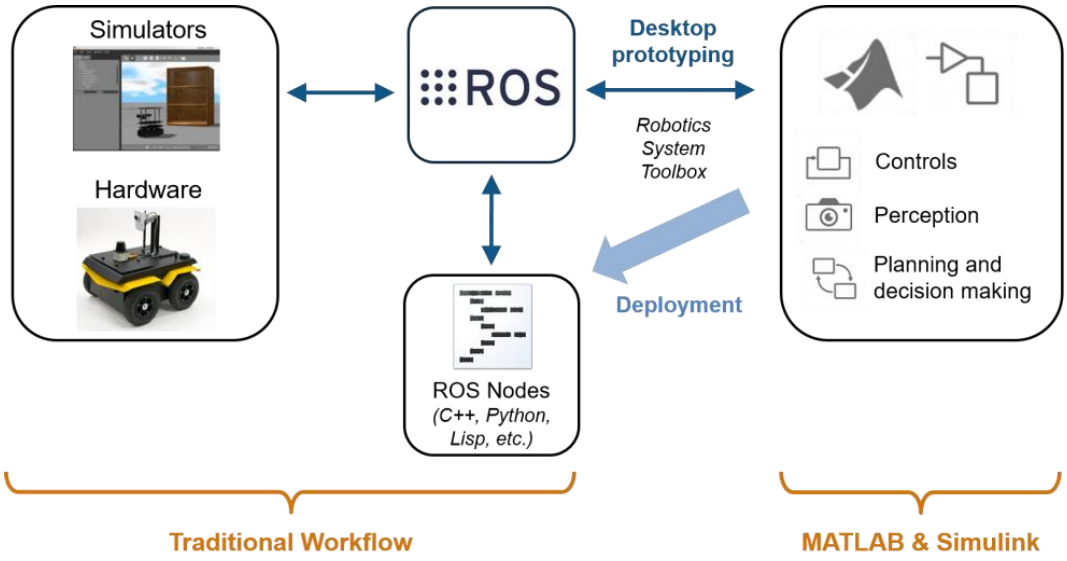

Figure 6.9 Matlab-ROS workflow (Matlab-documentation)

The Matlab-ROS workflow is shown in Figure 6.9. ROS communicates with the robot hardware and simultaneously subscribes and publishes to Matlab algorithms to obtain updated outcomes of joint values from Matlab. This interface allows the user to add new functions to ROS libraries. The user just has to input sensor values from the robot to Matlab code, and then Matlab performs the calculations and publishes the outcome of joint values to the robot via ROS.

Matlab can be connected to a Gazebo simulation or a physical robot by assigning the IP address of the robot to the Matlab node. As per the flowchart in Figure 6.10, Matlab will connect with ROS using the assigned IP address. It will subscribe to ROS topics associated with the physical robot's position and velocity sensors. At every specified time instant, Matlab will solve the ODEs that describe the robot's dynamics to obtain the robot's joint position and velocity values and find the required motor torques needed for that motion. It

will publish the torque values to the robot motor controllers via ROS. The motor controllers are assumed to be effort controllers.

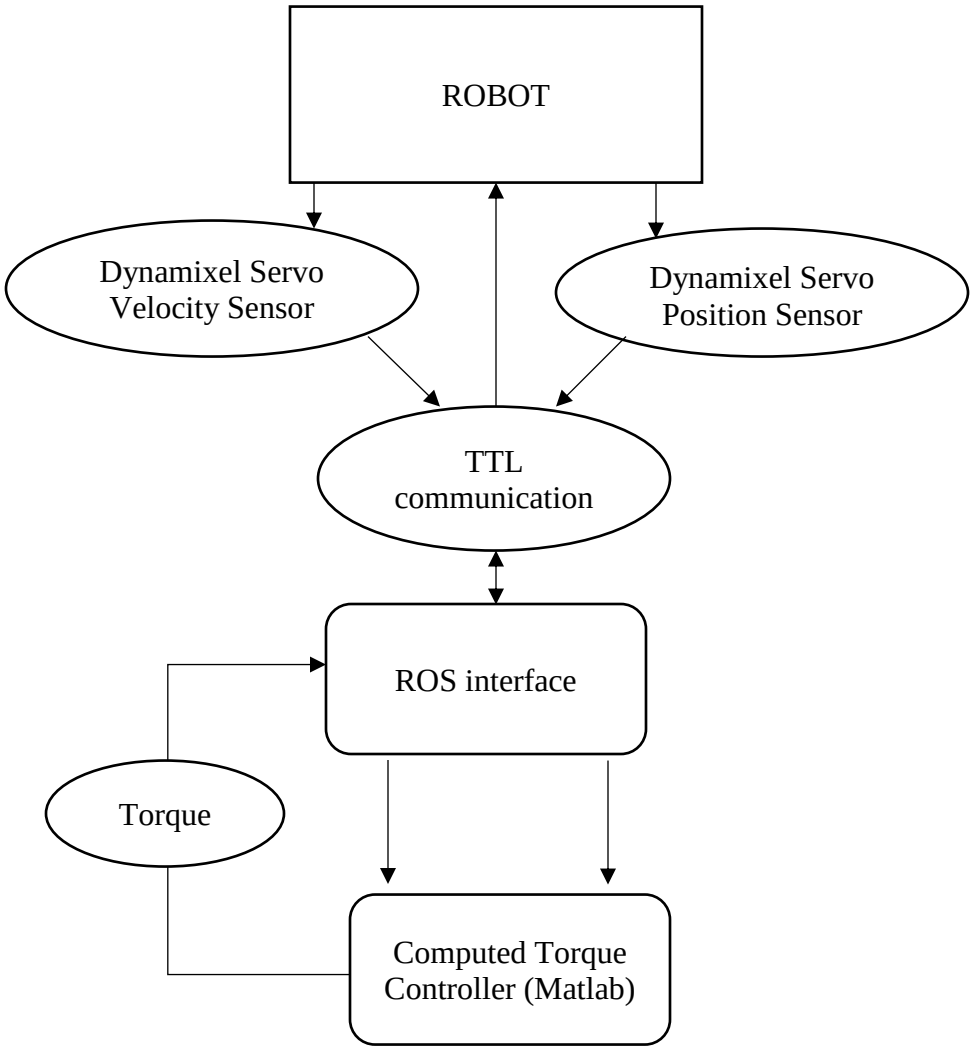

Figure 6.10 Flowchart of computed torque controller with Matlab-ROS interface

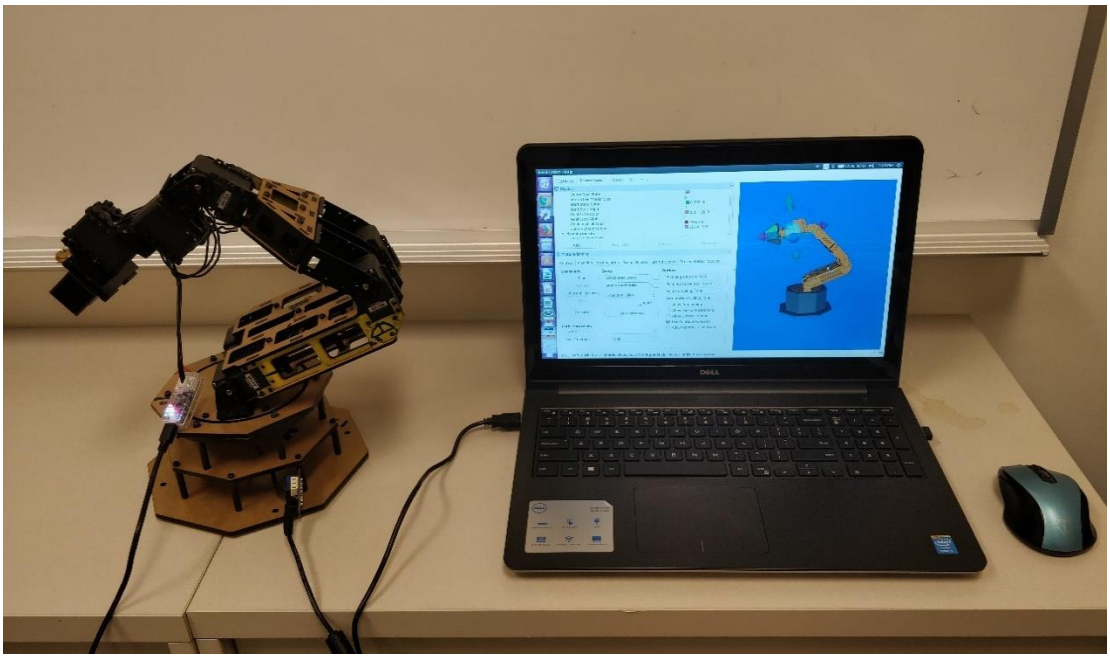

Figure 6.11 Setup of PhantomX Reactor robotic arm

Here, the PhantomX robotic arm was connected to a computer by TTL communication and Dynamixel servo motors (Figure 6.11). The Matlab code for the Gazebo and Matlab interface is presented in Appendix A.

An experiment to implement pick-and-place action, which is essential in bricklaying operations, was performed on the PhantomX Robot with position controller, and a similar action was simulated in Gazebo. Figure 6.12 shows a time sequence of snapshots of a successful pick-and-place action by the physical robot with a cardboard block, as well as a snapshot of the Gazebo model.

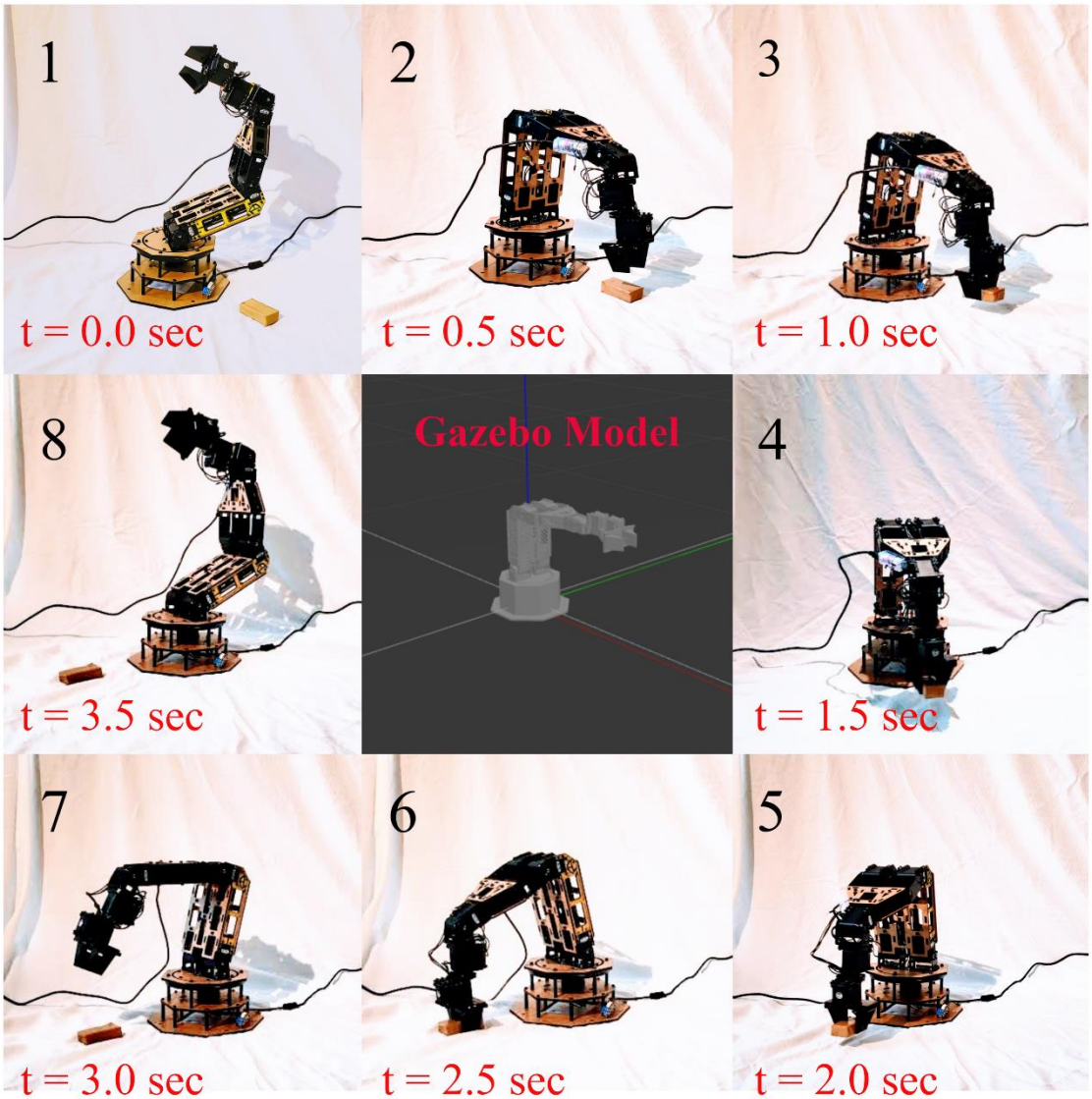

Figure 6.12 Snapshots of pick-and-place action by the PhantomX robotic arm and Gazebo model of PhantomX robotic arm (center)

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

This thesis describes the procedure of designing, simulating, and implementing a controller for a robotic arm to perform movements that are useful for construction activities. The thesis presents the kinematics and dynamics of a 5-DOF robotic manipulator, a trajectory planning procedure that can be used to generate bricklaying motions, controller design for improved performance, a software framework for controlling the robotic arm, and hardware implementation on a 5-DOF robotic arm.

In this thesis, a computed torque controller was designed for better trajectory performance of a robotic arm. The computed torque controller produces improved performance in trajectory following over a conventional PD controller because it accounts for the dynamics of the arm. Although a 5-DOF arm is a nonlinear dynamic system, the computed torque controller transforms it into a linear second-order system. Simulation results of the computed torque controller show that the arm with this controller is asymptotically stable and follows the desired trajectory.

The purpose of the designing the software framework was to make it easier for new users to control a robotic arm. The 5-DOF robotic arm was also modeled in ROS to visualize the actual behavior of the arm in the Gazebo or Rviz simulators. A Matlab-ROS interface was developed so that the computed torque controller can be implemented easily on the robot via Matlab. This Matlab-ROS interface allows users to implement different controllers on this robot just by changing the controller source file. The Moveit-ROS

interface developed in this thesis empowers users to use different motion and path planning algorithms for a robotic arm. The Matlab-ROS and Moveit interfaces were also used to control an actual PhantomX Reactor robotic arm. The Dynamixel servos AX-12a and USB2Dynamixel converter were used for implementing the controller on the hardware.

7.2 FUTURE WORK

There are some possible extensions to this thesis as future work:

(1) The current robotic arm has a static base. It is possible to expand the workspace of the PhantomX robotic arm by attaching it to a mobile robotic platform, such as the TurtleBot3 Waffle robot.

(2) During brick-laying operations, if a brick is of varying or unknown weight, then it is difficult to obtain satisfactory performance from PD and CTC controllers. As future work, adaptive and robust controllers can be designed for loads with unknown parameters.

(3) The Dynamixel servos AX-12a have proportional controllers, and they can only sense joint position and velocity. These servos could be replaced by MX-64 servos, which have torque controllers that are possible to directly command using the CTC controller's output.

REFERENCES

A. Balestrino, G. De Maria, L. Sciavicco. *An adaptive model following control.* ASME Journal of Dynamic Systems, Measurement and Control , vol. 105, pp. 143–151, 1983.

Alia N, Barakatl, Khaled A.Gouda, Kenz A. Bozed. "Kinematics Analysis and Simulation of a Robotic Arm." *Proceedings of 2016 4th International Conference on Control Engineering & Information Technology.* 2016.

Anil Mahtani, Luis Sánchez, Enrique Fernández, Aaron Martinez. *Effective Robotics Programming with ROS.* Packt Publishing, 2016.

Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics-Modelling, Planning and Control.* Springer, 2009.

Carol Fairchild, Dr. Thomas L. Harman. *ROS Robotics By Example.* Packt Publishing, 2016.

Corke, Peter. "Robot Manipulator Capability in MATLAB." *IEEE ROBOTICS & AUTOMATION MAGAZINE* , 2017.

—. *Robotics,Vision and Control.* Springer, 2012.

"Dynamixel_controllers." *ros-wiki.* n.d. http://wiki.ros.org/dynamixel_controllers/Tutorials.

"dynamixel_motor joint control." *GitHub.* n.d. https://github.com/arebgun/dynamixel_motor/blob/master/dynamixel_controllers/src/dynamixel_controllers/joint_trajectory_action_controller.py.

Fenton, X. Shi and N.G. "A complete and general solution to the forward kinematics problem of platform-type robotic manipulators." *IEEE International Conference on Robotics and Automation proceedings.* 1994. pp. 3055-3062.

"Gazebo-ROS control." *Gazebo.* n.d. http://gazebosim.org/tutorials/?tut=ros_control.

"Getting started with ROS-Matlab." *Matlab documentaion.* n.d. https://www.mathworks.com/matlabcentral/fileexchange/64534-getting-started-with-robot-operating-system-ros.

Joseph, Lentin. *Learning Robotics Using Python.* Packt Publishing Ltd., 2014.

—. *Mastering ROS for Robotics Programming.* Packt Publishing Ltd., 2015.

Lab, ACS. *ACS Lab GitHub Page.* 03 28, 2019. https://github.com/ACSLaboratory (accessed 3 28, 2019).

Marvi, Hamidreza. *MAE-547 Modeling and Control of Robots Lecture Notes.* 2018.

Melchiorri, Claudio. "Control of robot manipulators." Masters Thesis, n.d.

Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS.* 2016: O'Reilly Media, Inc., n.d.

"Move-group Python Interface." *Moveit.* n.d. https://ros-planning.github.io/moveit_tutorials/doc/move_group_python_interface/move_group_python_interface_tutorial.html.

"Moveit!" *Moveit concept documentation.* n.d. https://moveit.ros.org/documentation/concepts/.

Pivac, Mike. *X, Hadrian.* 2016. https://www.fbr.com.au/view/hadrian-x.

PODKAMINER, NATE. *Construction Robotics SAM100.* 2016. https://www.construction-robotics.com/sam100/.

Quirke, Joe. *CIOB.* november 15, 2018. http://www.globalconstructionreview.com/news/hadrian-bricklaying-robot-builds-complete-house-th/.

QUOCTRUNG BUI, ROGER KISBY. *Bricklayers Think They're Safe From Robots.* New York, March 06, 2018.

R. Kelly, V. Santibáñez and A. Loría. *Control of Robot Manipulators in Joint Space.* Springer, 2005.

Ren, Yi. *Design informatics Lab.* 2018. http://designinformaticslab.github.io/index.html.

Rick. *Trossen Robotics - Phantomx Robotic Reactor Arm* . 2017. https://www.trossenrobotics.com/p/phantomx-ax-12-reactor-robot-arm.aspx.

Shahab, Mohammad. "2DOF Robotic Manipulator." Report, 2008.

"wiki-ros." *ROS actionlib.* n.d. http://wiki.ros.org/actionlib.

Y. Antonio, S. Victor and M.V. Javier. "Global asymptotic stability of the classical PID controller by considering saturation effects in industrial robots." *International Journal of Advanced Robotic Systems, Vol. 8,*, 2011: pp. 34-42.

APPENDIX A

MATLAB - GAZEBO INTERFACE

```
rosshutdown;
#connects with robot with following ip-address
rosinit('ip-address')

#Declares publisher with trajectory controller
[myPub,pubMsg] =
rospublisher('SG_robot/arm_joint_controller/command');

#Defines msg type associated with trajectory controller
joint_send =
rosmessage('trajectory_msgs/JointTrajectoryPoint');

load('x_pid_2');
joint_send.Positions = zeros(5,1);
joint_send.Velocities = zeros(5,1);
pubMsg.Points = joint_send;
pubMsg.JointNames =
{'shoulder_yaw_joint','shoulder_pitch_joint','elbow_pit
ch_joint','wrist_pitch_joint','wrist_roll_joint','gripp
er_revolute_joint'}

i=1; j=1;
while(i<6)
    pubMsg.Points.Positions =
[Q1(4*i,1);Q1(4*i,2);Q1(4*i,3);Q1(4*i,4);Q1(4*i,5)]

    if i == 1
        pubMsg.Points.TimeFromStart.Sec =
           round((0.49*((i-1)/2)),0);
        pubMsg.Points.TimeFromStart.Nsec = 240099009
    elseif i == 2
        pubMsg.Points.TimeFromStart.Sec =
           round((0.49*((i-1)/2)),0);
        pubMsg.Points.TimeFromStart.Nsec = 490099009
    elseif i == 3
        pubMsg.Points.TimeFromStart.Sec =
           round((0.49*((i-1)/2)),0);
        pubMsg.Points.TimeFromStart.Nsec = 740099009
    else i == 4
        pubMsg.Points.TimeFromStart.Sec =
           round((0.49*((i-1)/2)),0);
        pubMsg.Points.TimeFromStart.Nsec = 990099009
```

```
    end

# Publishes pubMsg to actual robot from Matlab
    send(myPub,pubMsg);
         for t= 0:0.001:1
             j = i+t;
         end
     i = j;
end
```