WidowX 200 Robotic Arm User Manual

Jeff Grasberger, Ritish Shailly, Burhannudin Bharmal, Shardul Amrite

August 2020

Contents

1 Introduction

This documents serves as a user manual for three different models of the WidowX 200 Robotic Arm: the physical robot itself, a Gazebo simulation, and a Simulink simulation, as well as a graphical user interface (GUI) that can be used to operate the three models. More information on the WidowX 200 Robotic Arm can be found at the Trossen Robotics website: `https://www.trossenrobotics.com/widowx-200-robot-arm.aspx`. First, the theory behind the methods used to control the arm is explained. Then, for each of the three models, the procedures required for setup, tuning, running, and troubleshooting are outlined within this manual.

2 Underlying Theory

Figure 1 is the block diagram of the system we are developing.

- Inverse Kinematics (IK): The user can specify the end effector position. The IK model gives the required joint angles q_d.
- Trajectory Generator: Provides the required joint velocities and accelerations to reach the required joint positions.
- Controller: This block calculates the required torque according to the error and the system parameters and a command signal is sent to the Manipulator system.

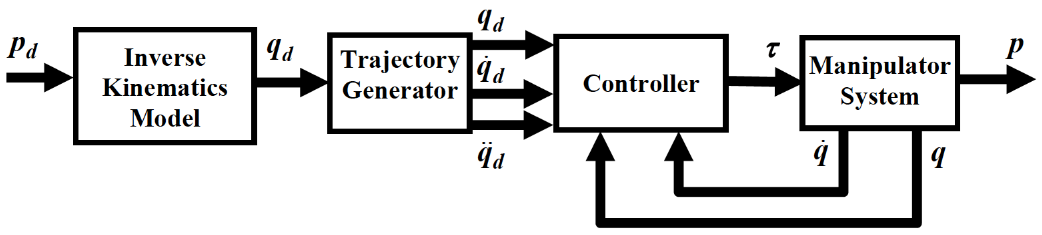


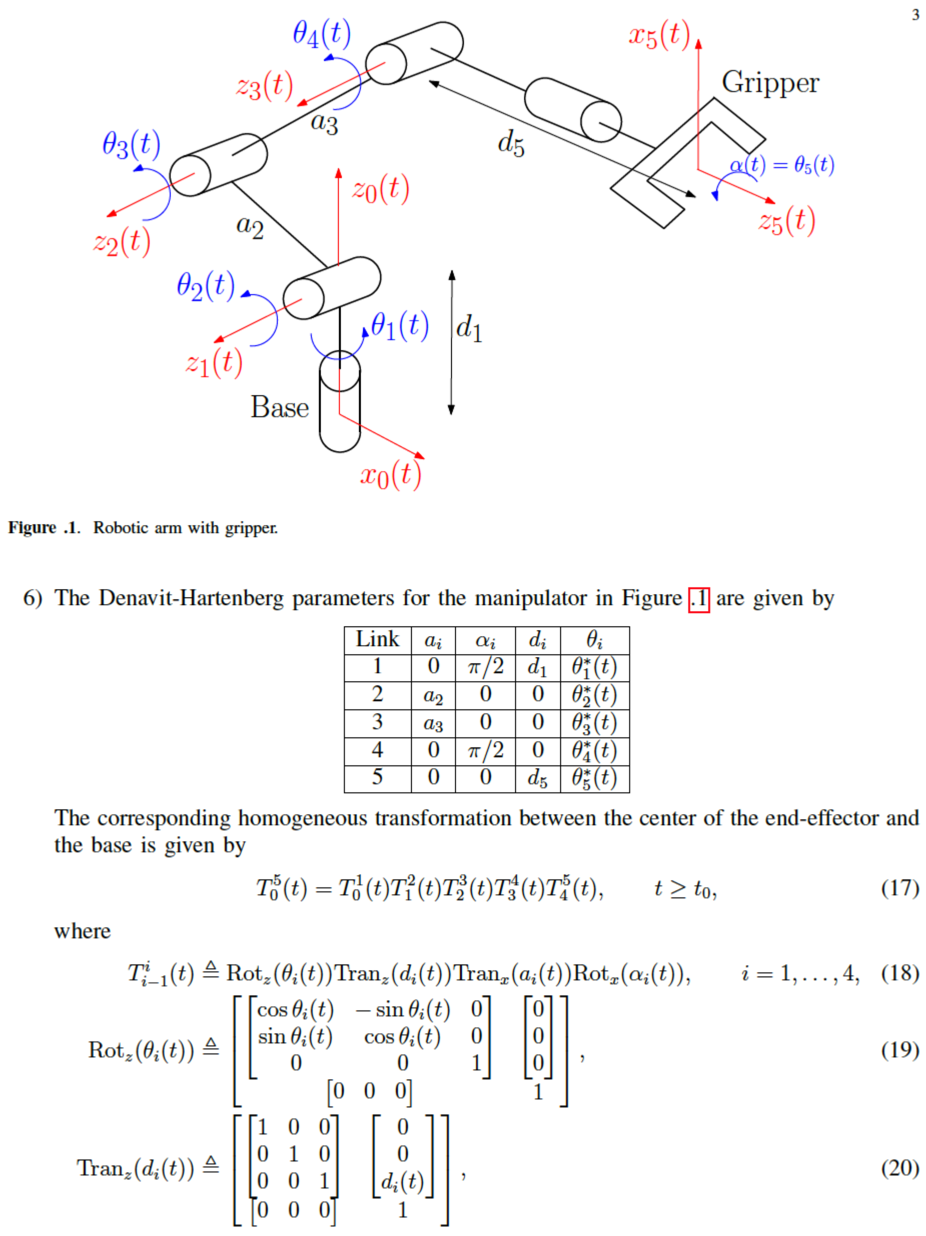
Figure 1: Block diagram of the control system used in each model

2.1 Inverse Kinematics

Inverse Kinematics is a process utilized by each of the three models to determine the joint angles required to meet a desired end effector position and orientation. In the case of the WidowX 200 robotic arm, the following parameters are specified by the user:

- p: vector of the form [x;y;z] – specifies the desired end effector position.
- alpha: angle in radians – specifies the desired gripper angle.
- z_5: vector of the form [x;y;z] – specifies the desired axis of the end effector. For example, a [0;0;1] indicates the end effector will be pointing straight upward (parallel to the z axis). To reach further positions, it is recommended to set the axis as the same or similar to the p vector. To reach closer positions, it is recommended to set the axis as more upward or downward with relatively high positive or negative z-values.

Based on the specified parameters, the Denavit-Hartenberg convention for inverse kinematics computes and returns the required joint angles. This method and the math behind it is further outlined in Figures 2, 3, 4, and 5 below.



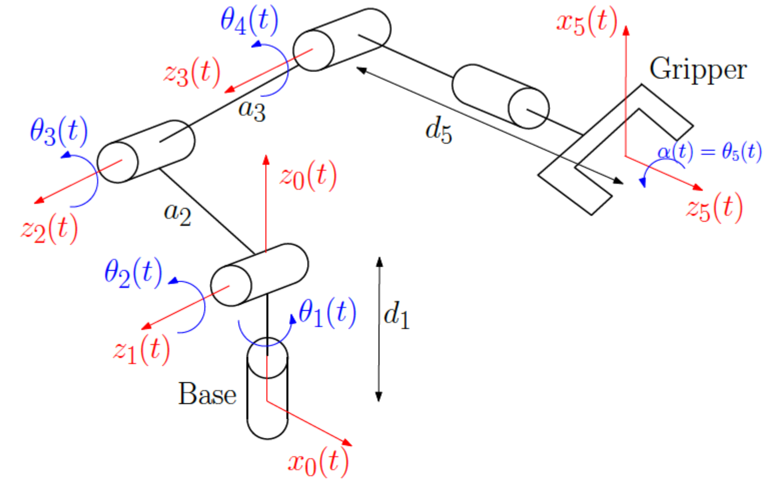


Figure .1. Robotic arm with gripper.

6) The Denavit-Hartenberg parameters for the manipulator in Figure .1 are given by

Link	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	d_1	$\theta_1^*(t)$
2	a_2	0	0	$\theta_2^*(t)$
3	a_3	0	0	$\theta_3^*(t)$
4	0	$\pi/2$	0	$\theta_4^*(t)$
5	0	0	d_5	$\theta_5^*(t)$

The corresponding homogeneous transformation between the center of the end-effector and the base is given by

$$T_0^5(t) = T_0^1(t)T_1^2(t)T_2^3(t)T_3^4(t)T_4^5(t), \qquad t \geq t_0, \tag{17}$$

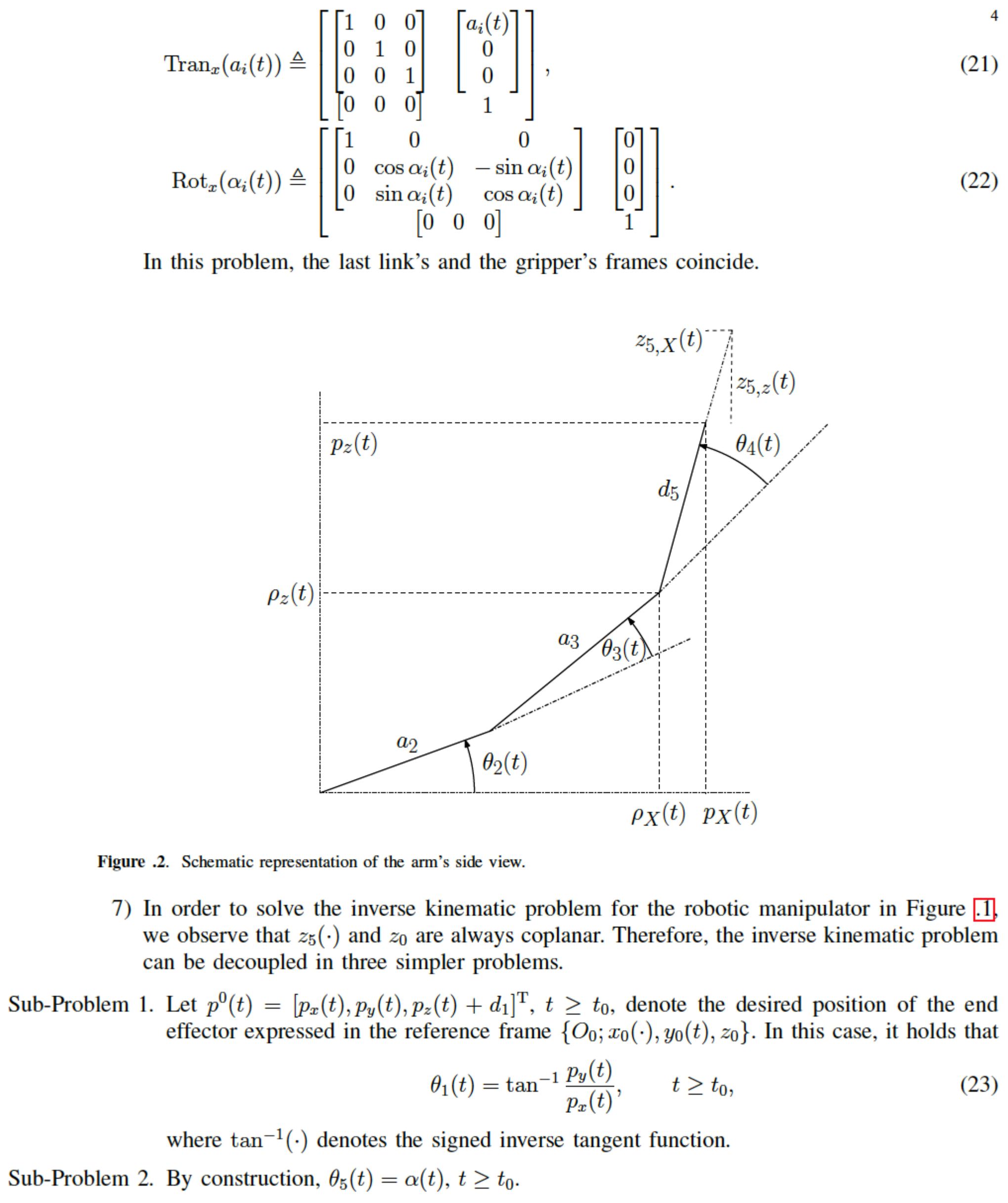
where

$$T_{i-1}^i(t) \triangleq \text{Rot}_z(\theta_i(t))\text{Tran}_z(d_i(t))\text{Tran}_x(a_i(t))\text{Rot}_x(\alpha_i(t)), \qquad i = 1, \ldots, 4, \tag{18}$$

$$\text{Rot}_z(\theta_i(t)) \triangleq \begin{bmatrix} \begin{bmatrix} \cos\theta_i(t) & -\sin\theta_i(t) & 0 \\ \sin\theta_i(t) & \cos\theta_i(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} & 1 \end{bmatrix}, \tag{19}$$

$$\text{Tran}_z(d_i(t)) \triangleq \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ d_i(t) \\ 1 \end{bmatrix} \end{bmatrix}, \tag{20}$$

Figure 2: IK1



4

$$\mathrm{Tran}_x(a_i(t)) \triangleq \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} a_i(t) \\ 0 \\ 0 \end{bmatrix} \\ [0 \;\; 0 \;\; 0] & 1 \end{bmatrix}, \tag{21}$$

$$\mathrm{Rot}_x(\alpha_i(t)) \triangleq \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_i(t) & -\sin\alpha_i(t) \\ 0 & \sin\alpha_i(t) & \cos\alpha_i(t) \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ [0 \;\; 0 \;\; 0] & 1 \end{bmatrix}. \tag{22}$$

In this problem, the last link's and the gripper's frames coincide.

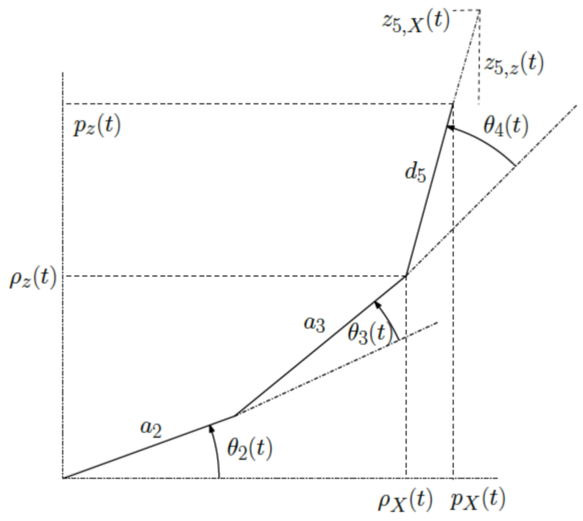


Figure .2. Schematic representation of the arm's side view.

7) In order to solve the inverse kinematic problem for the robotic manipulator in Figure .1, we observe that $z_5(\cdot)$ and z_0 are always coplanar. Therefore, the inverse kinematic problem can be decoupled in three simpler problems.

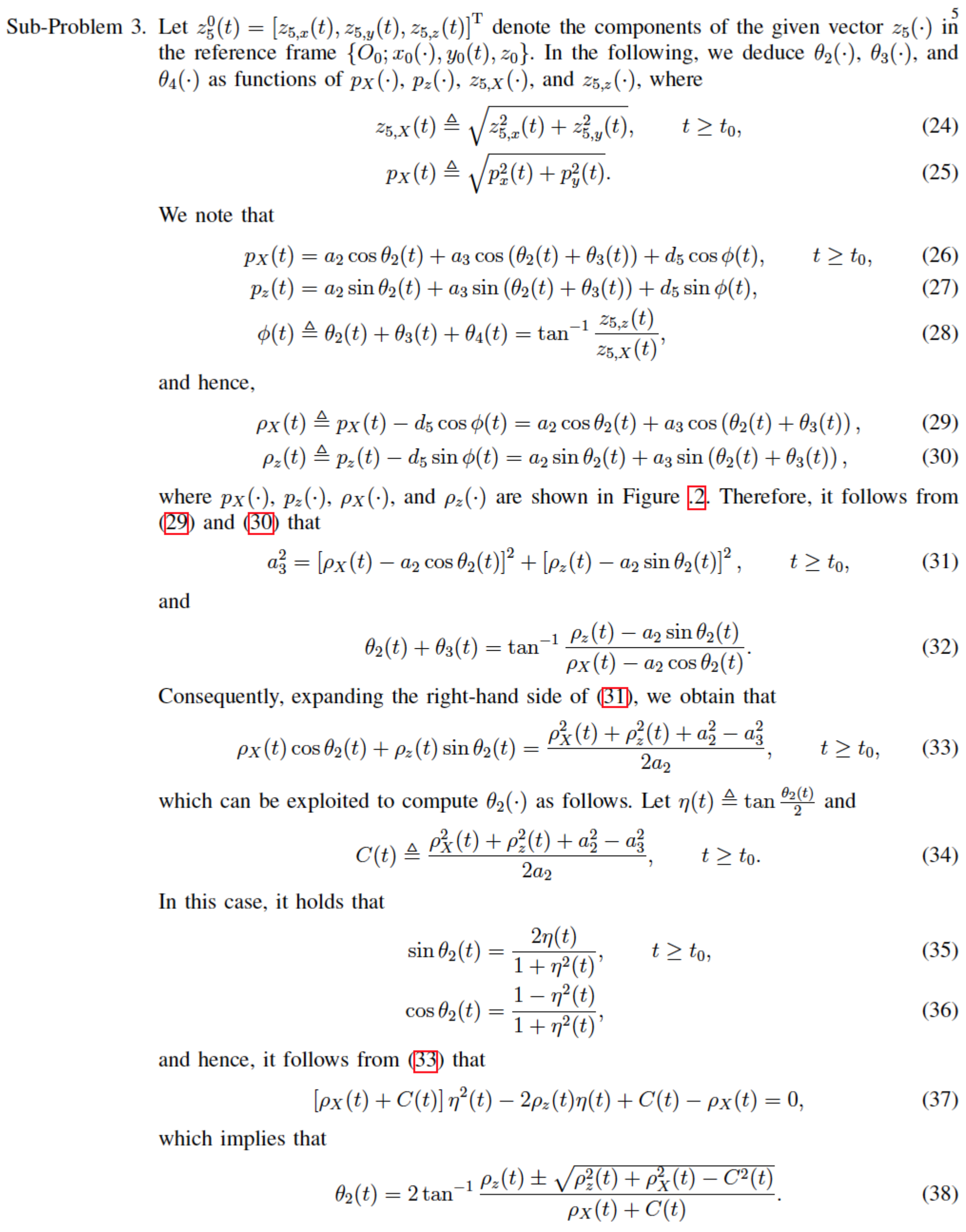
Sub-Problem 1. Let $p^0(t) = [p_x(t), p_y(t), p_z(t) + d_1]^{\mathrm{T}}$, $t \geq t_0$, denote the desired position of the end effector expressed in the reference frame $\{O_0; x_0(\cdot), y_0(t), z_0\}$. In this case, it holds that

$$\theta_1(t) = \tan^{-1}\frac{p_y(t)}{p_x(t)}, \qquad t \geq t_0, \tag{23}$$

where $\tan^{-1}(\cdot)$ denotes the signed inverse tangent function.

Sub-Problem 2. By construction, $\theta_5(t) = \alpha(t)$, $t \geq t_0$.

Figure 3: IK2



Sub-Problem 3. Let $z_5^0(t) = [z_{5,x}(t), z_{5,y}(t), z_{5,z}(t)]^{\mathrm{T}}$ denote the components of the given vector $z_5(\cdot)$ in the reference frame $\{O_0; x_0(\cdot), y_0(t), z_0\}$. In the following, we deduce $\theta_2(\cdot)$, $\theta_3(\cdot)$, and $\theta_4(\cdot)$ as functions of $p_X(\cdot)$, $p_z(\cdot)$, $z_{5,X}(\cdot)$, and $z_{5,z}(\cdot)$, where

$$z_{5,X}(t) \triangleq \sqrt{z_{5,x}^2(t) + z_{5,y}^2(t)}, \qquad t \geq t_0, \tag{24}$$

$$p_X(t) \triangleq \sqrt{p_x^2(t) + p_y^2(t)}. \tag{25}$$

We note that

$$p_X(t) = a_2 \cos\theta_2(t) + a_3 \cos(\theta_2(t) + \theta_3(t)) + d_5 \cos\phi(t), \qquad t \geq t_0, \tag{26}$$

$$p_z(t) = a_2 \sin\theta_2(t) + a_3 \sin(\theta_2(t) + \theta_3(t)) + d_5 \sin\phi(t), \tag{27}$$

$$\phi(t) \triangleq \theta_2(t) + \theta_3(t) + \theta_4(t) = \tan^{-1}\frac{z_{5,z}(t)}{z_{5,X}(t)}, \tag{28}$$

and hence,

$$\rho_X(t) \triangleq p_X(t) - d_5 \cos\phi(t) = a_2 \cos\theta_2(t) + a_3 \cos(\theta_2(t) + \theta_3(t)), \tag{29}$$

$$\rho_z(t) \triangleq p_z(t) - d_5 \sin\phi(t) = a_2 \sin\theta_2(t) + a_3 \sin(\theta_2(t) + \theta_3(t)), \tag{30}$$

where $p_X(\cdot)$, $p_z(\cdot)$, $\rho_X(\cdot)$, and $\rho_z(\cdot)$ are shown in Figure 2. Therefore, it follows from (29) and (30) that

$$a_3^2 = [\rho_X(t) - a_2 \cos\theta_2(t)]^2 + [\rho_z(t) - a_2 \sin\theta_2(t)]^2, \qquad t \geq t_0, \tag{31}$$

and

$$\theta_2(t) + \theta_3(t) = \tan^{-1}\frac{\rho_z(t) - a_2 \sin\theta_2(t)}{\rho_X(t) - a_2 \cos\theta_2(t)}. \tag{32}$$

Consequently, expanding the right-hand side of (31), we obtain that

$$\rho_X(t)\cos\theta_2(t) + \rho_z(t)\sin\theta_2(t) = \frac{\rho_X^2(t) + \rho_z^2(t) + a_2^2 - a_3^2}{2a_2}, \qquad t \geq t_0, \tag{33}$$

which can be exploited to compute $\theta_2(\cdot)$ as follows. Let $\eta(t) \triangleq \tan\frac{\theta_2(t)}{2}$ and

$$C(t) \triangleq \frac{\rho_X^2(t) + \rho_z^2(t) + a_2^2 - a_3^2}{2a_2}, \qquad t \geq t_0. \tag{34}$$

In this case, it holds that

$$\sin\theta_2(t) = \frac{2\eta(t)}{1 + \eta^2(t)}, \qquad t \geq t_0, \tag{35}$$

$$\cos\theta_2(t) = \frac{1 - \eta^2(t)}{1 + \eta^2(t)}, \tag{36}$$

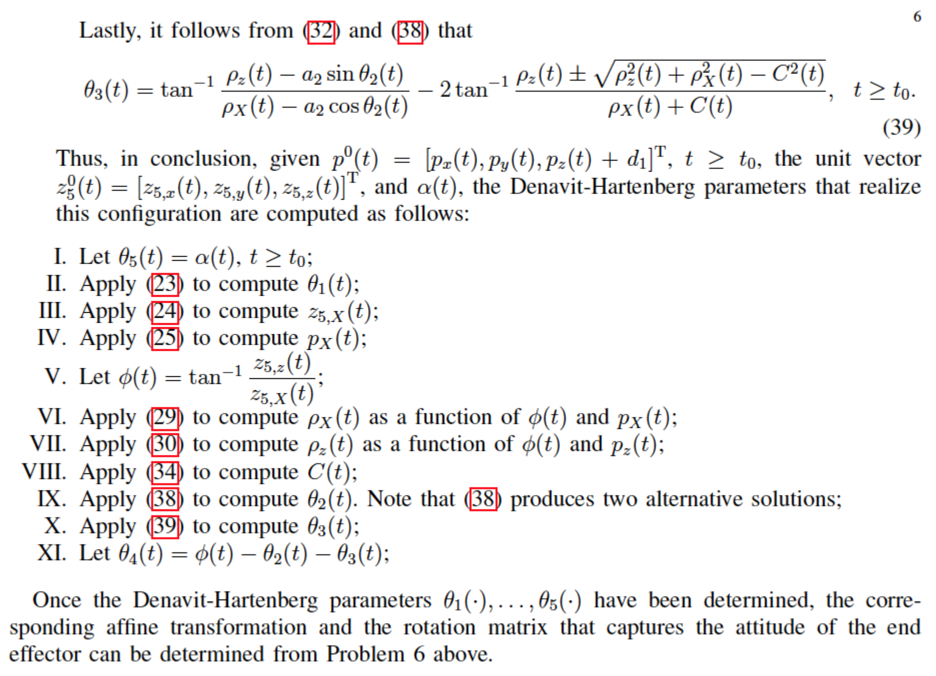
and hence, it follows from (33) that

$$[\rho_X(t) + C(t)]\,\eta^2(t) - 2\rho_z(t)\eta(t) + C(t) - \rho_X(t) = 0, \tag{37}$$

which implies that

$$\theta_2(t) = 2\tan^{-1}\frac{\rho_z(t) \pm \sqrt{\rho_z^2(t) + \rho_X^2(t) - C^2(t)}}{\rho_X(t) + C(t)}. \tag{38}$$

Figure 4: IK3



6

Lastly, it follows from (32) and (38) that

$$\theta_3(t) = \tan^{-1}\frac{\rho_z(t) - a_2 \sin\theta_2(t)}{\rho_X(t) - a_2\cos\theta_2(t)} - 2\tan^{-1}\frac{\rho_z(t) \pm \sqrt{\rho_z^2(t) + \rho_X^2(t) - C^2(t)}}{\rho_X(t) + C(t)}, \quad t \geq t_0. \tag{39}$$

Thus, in conclusion, given $p^0(t) = [p_x(t), p_y(t), p_z(t) + d_1]^{\mathrm{T}}$, $t \geq t_0$, the unit vector $z_5^0(t) = [z_{5,x}(t), z_{5,y}(t), z_{5,z}(t)]^{\mathrm{T}}$, and $\alpha(t)$, the Denavit-Hartenberg parameters that realize this configuration are computed as follows:

I. Let $\theta_5(t) = \alpha(t)$, $t \geq t_0$;
II. Apply (23) to compute $\theta_1(t)$;
III. Apply (24) to compute $z_{5,X}(t)$;
IV. Apply (25) to compute $p_X(t)$;
V. Let $\phi(t) = \tan^{-1}\dfrac{z_{5,z}(t)}{z_{5,X}(t)}$;
VI. Apply (29) to compute $\rho_X(t)$ as a function of $\phi(t)$ and $p_X(t)$;
VII. Apply (30) to compute $\rho_z(t)$ as a function of $\phi(t)$ and $p_z(t)$;
VIII. Apply (34) to compute $C(t)$;
IX. Apply (38) to compute $\theta_2(t)$. Note that (38) produces two alternative solutions;
X. Apply (39) to compute $\theta_3(t)$;
XI. Let $\theta_4(t) = \phi(t) - \theta_2(t) - \theta_3(t)$;

Once the Denavit-Hartenberg parameters $\theta_1(\cdot), \ldots, \theta_5(\cdot)$ have been determined, the corresponding affine transformation and the rotation matrix that captures the attitude of the end effector can be determined from Problem 6 above.

Figure 5: IK4

2.2 Trajectory Planning

This section outlines the two methods used for trajectory planning: linear segments with parabolic blends (LSPB – Section 2.2.1) and quintic polynomial (Section 2.2.2). Both of these trajectory planning methods require the input of the initial and final positions and the total time. The LSPB method also requires the time at each time step, while the quintic polynomial method requires the initial and final velocities and accelerations as well. From these parameters, both methods create a trajectory to proceed from the initial positions to final positions within the specified amount of time. When operating either one of the simulations or the actual robot, the user may choose to use either the LSPB or quintic polynomial method.

2.2.1 Linear Segments with Parabolic Blends (LSPB) Method (page 192 of Spong's book "Robot Modeling and Control")

The LSPB method focuses on implementing motion with a constant velocity. The trajectory created using this method steadily increases velocity until a time specified as the blend time. Then, the velocity is held constant before steadily decreasing when the current time is equal to the final time minus the blend time. This method is applied to each joint and leads to joint position trajectories with parabolic portions before and after the blend time limits and linear segments in the middle of the trajectory. The position, velocity, and acceleration curves generated using this method are shown in Figure 6.

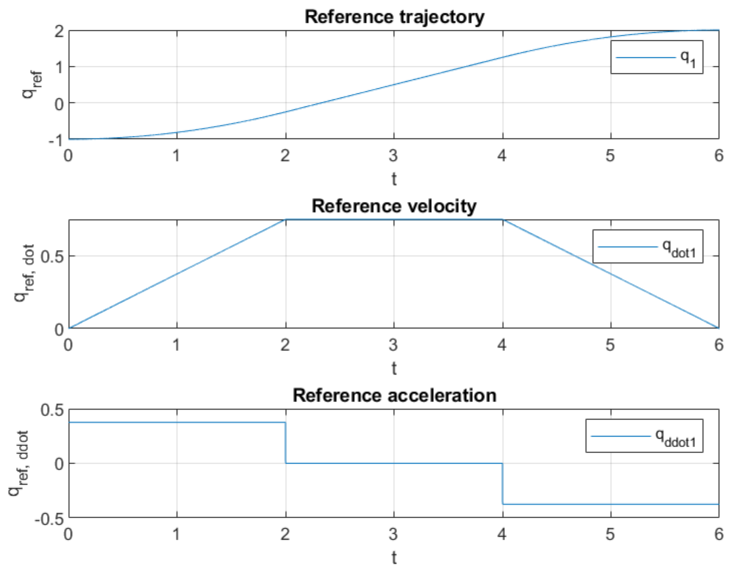


Figure 6: Position, velocity, and acceleration trajectories using the LSPB method

2.2.2 Quintic Polynomial Method (page 191 of Spong's book "Robot Modeling and Control")

The quintic polynomial method computes the trajectory of each joint using a fifth order polynomial. This method creates a slightly smoother trajectory than the LSPB method. The equations to be solved are shown in Figure 7. After solving these equations, the reference trajectories are created as displayed in Figure 8.

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ \alpha_0 \\ q_f \\ v_f \\ \alpha_f \end{bmatrix}$$

Figure 7: Quintic polynomial equations in matrix form

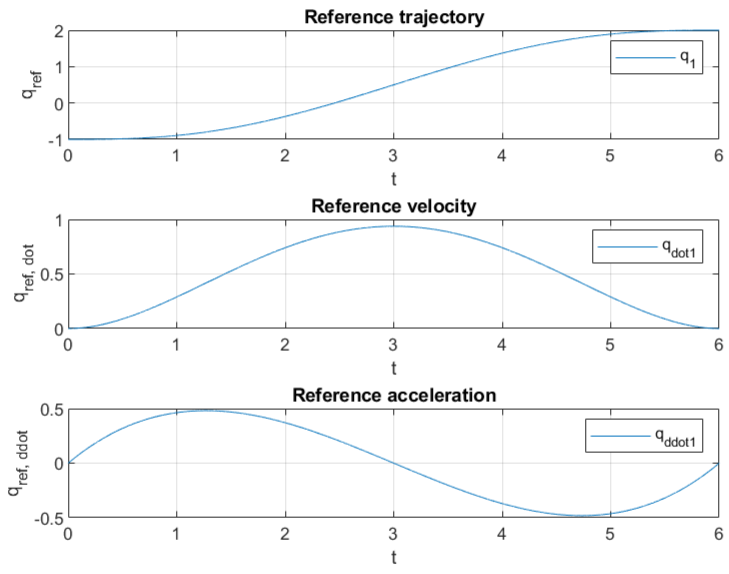


Figure 8: Position, velocity, and acceleration trajectories using the quintic polynomial method

2.3 PID Control

Proportional-integral-derivative (PID) control is a control system that implements a feedback loop to determine the torque to be applied at each time step. With the WidowX 200 models, the PID controller inputs the error, integral of error, and derivative of the error. The P, I, and D gains (specified by user) then multiply with the respective error to produce the torque inputs. This process occurs at each time step, ensuring the arm follows the trajectory closely. The effects of changing each of the gains are delineated in Figure 9.

$$u = M(q)[\ddot{q}_{ref} - K_p e - K_d \dot{e} - K_i e_{integral}] + C(q, \dot{q})(\dot{q}) + K_g + U_{lqr}(e, \dot{e})$$

K_p = Proportional Gain
K_d = Derivative Gain
K_i = Integral Gain

$K_p = \omega^2$
$K_d = 2\zeta\omega$

K_p is directly proportional to rise and settling time.
K_d value affects the overshoot behaviour of the system.

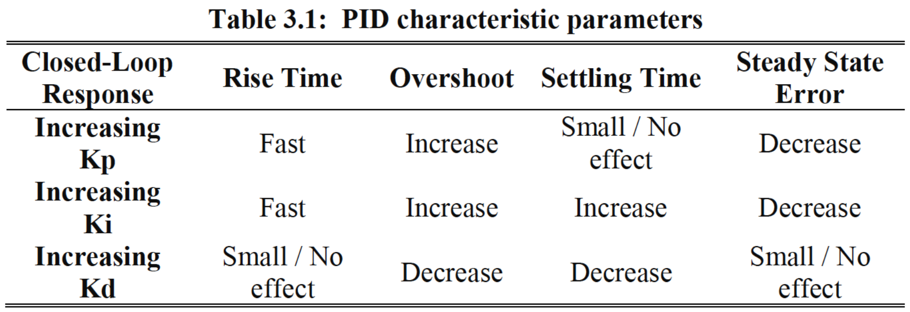


Table 3.1: PID characteristic parameters

Closed-Loop Response	Rise Time	Overshoot	Settling Time	Steady State Error
Increasing Kp	Fast	Increase	Small / No effect	Decrease
Increasing Ki	Fast	Increase	Increase	Decrease
Increasing Kd	Small / No effect	Decrease	Decrease	Small / No effect

Figure 9: Effect of tuning PID gains

2.4 LQR Method

Optimal Control enables an intuitive way to develop a controller with the help of a cost function. The cost function returns a set of gains which penalize either the error of each state or penalize the amount of control input required.

The variables in which we can use to gain control over our system are: r_{11}, r_{12}, r_2.

- r_{11} represents the penalty applied on the proportional error (position). Higher values will give higher convergence time (more error) for the respective states (position and velocity).
- r_{12} represents the penalty applied on the derivative error (velocity).
- r_2 represents the penalty on the control force (torque).

Higher values of these variables will give higher convergence times(for r_{11}, r_{12}) and lower control force(for r_2)

2.5 Torque Equation (page 295 of Spong's book "Robot Modeling and Control")

$$u = M(q)[\ddot{q}_{ref} - K_p e - K_d \dot{e} - K_i e_{integral}] + C(q, \dot{q})(\dot{q}) + K_g + U_{lqr}(e, \dot{e})$$

- u = Joint Torque
- $M(q)$ = Generalised Mass Matrix
- $C(q, \dot{q})$ = Centripetal and Coriolis matrix
- K_g = Gravitational Force Matrix
- e = error in actual joint position and ref position
- $\dot{e}$ = error in actual joint velocity and ref velocity
- $e_{integral}$ = integral of the position errors
- $\ddot{q}_{ref}$ = reference joint acceleration
- K_p = proportional gain
- K_d = Derivative gain
- K_i = Integral Gain
- U_{lqr} = Additional Torque from LQR

3 Virtual Models

3.1 Gazebo

3.1.1 Introduction to Gazebo Model

This section outlines the process of creating the workspace and running the ROS Gazebo model for the WidowX 200 robotic arm. Inverse Kinematics (Section 2.1) is used to calculate the reference trajectory from the user defined end effector position and gripper angle. Based on the calculated reference trajectory and current position and velocity readings from the joints, a PD controller (Section 2.3) with LQR control (Section 2.4) calculates the necessary torques to follow the reference trajectory. The torques are applied to the respective joints, which actuates the Gazebo model shown in Figure 10.

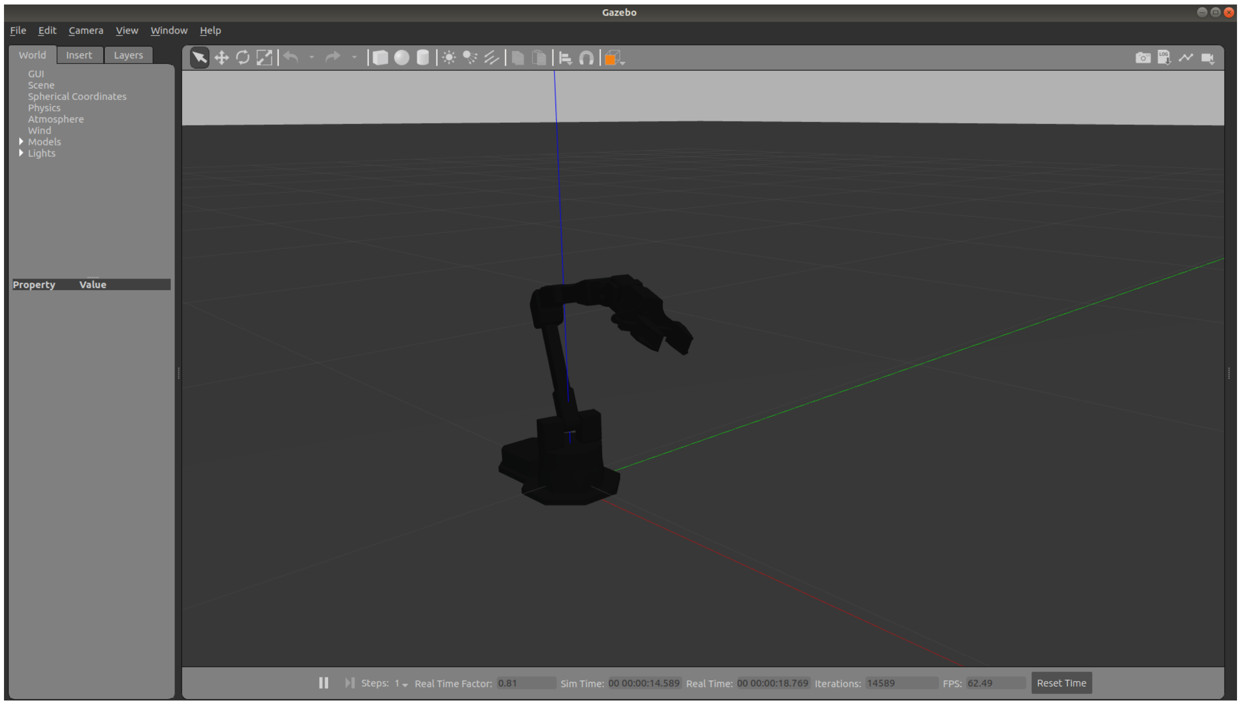

Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.81 Sim Time: 00 00:00:14.589 Real Time: 00 00:00:18.769 Iterations: 14589 FPS: 62.49
Reset Time

Figure 10: ROS Gazebo model of WidowX 200 Robotic Arm

3.1.2 Building the workspace

This model was created using ROS Melodic, which should be installed on your Linux machine prior to accessing the model. ROS Melodic can be installed by following the steps given in this link: `http://wiki.ros.org/melodic/Installation/Ubuntu`.

Before creating the workspace, the necessary packages need to be installed.

The instructions for downloading and installing the repository and the necessary packages are outlined below:

Step 1: Open a terminal on your Linux machine to carry out the necessary commands.

Step 2: Before building the workspace, the Gnuplot and Eigen Library packages should be installed.

```
$ sudo apt-get install -y gnuplot
$ sudo apt-get install libeigen3-dev
```

Step 3: After installing the two libraries, create a new catkin workspace called interbotix_ws. It is recommended that you create the workspace in the home directory.

```
$ mkdir -p ~/interbotix_ws/src
$ cd ~/interbotix_ws/
$ catkin_make
```

Step 4: Now, move to the src folder.

```
$ cd src
```

Step 5: Next, copy the Interbotix_src folder from the drive in the src folder.

Step 6: The next commands ensure that the workspace is sourced every time a terminal is opened.

```
$ cd ~/interbotix_ws/
$ source ~/interbotix_ws/devel/setup.bash
$ echo "source ~/interbotix_ws/devel/setup.bash" >> ~/.bashrc
```

Step 7: Before doing another catkin_make, make sure that all required dependencies are installed. Rosdep will be used to do this efficiently.

```
$ rosdep update
$ rosdep install --from-paths src --ignore-src -r -y
```

Step 8: Now that all the dependencies are installed, it's time to build!

```
$ catkin_make
$ source ~/.bashrc
```

3.1.3 Running The Simulation

Running the simulation is relatively simple by following these steps:

Step 1: Navigate to the interbotix_ws folder if not already there.

```
$ cd ~/interbotix_ws
```

Step 2: Before beginning the simulation, certain parameters can be changed to alter the performance of the controller. These parameters do not need to be changed, but can be if a certain controller performance is desired. If you would like to change these parameters, follow the steps below. Otherwise, continue to the next step.

- Navigate to the src/Interbotix_src/Gazebo/Data_log folder and edit the data_to_read.txt file.

  ```
  $ cd src/Interbotix_src/Gazebo/Data_log/
  $ gedit data_to_read.txt
  ```

- This will open the text file in edit mode. In the text file, you will find the omega and zeta values for each joint. These can be changed to control the natural frequency (omega) and damping ratio (zeta) of the response. Parameters such as the LQR values and the simulation time can also be changed within this text file.
- After changing the desired parameters, navigate back to the parent folder to run the model.

  ```
  $ cd ~/interbotix_ws
  ```

- To update the model with any changes made, do another catkin_make.

  ```
  $ catkin_make
  ```

Step 3: The following command runs the model in the standalone mode.

```
$ roslaunch interbotix_gazebo gazebo.launch
```

Step 4: Manipulate the arm by changing the end effector position p_x, p_y, p_z, end effector axis z_5x, z_5y, z_5z and gripper rotation alpha in the data_to_read.txt file. z_5 is the z axis of the last link or the gripper as you can see in the image in the section 2.1. So, the values associated with z_5 are it's direction cosines with respect to the ground frame(frame of reference of the base). For example, if the gripper is pointing upwards as seen from the ground then it will be parallel to z_0 from the image and hence z_5's direction cosines will be just [0 0 1]. In the image, it is somewhat parallel to x_0, so for this configuration, the direction cosines can be taken as [1 0 0] for example. So values for z_5x,z_5y,z_5z would be 1, 0, 0 respectively for this example.

Step 5: The arm can also be manipulated using keyboard: the reference position for all the joints can be changed by entering the following character in the ubuntu terminal.

```
char 'q' :  waist_ref =  waist_ref + 0.010
char 'a' :  waist_ref =  waist_ref - 0.010
char 'w' :  shoulder_ref =  shoulder_ref + 0.010
char 's' :  shoulder_ref =  shoulder_ref - 0.010
char 'e' :  elbow_ref =  elbow_ref + 0.010
char 'd' :  elbow_ref =  elbow_ref - 0.010
char 'r' :  wrist_angle_ref =  wrist_angle_ref + 0.010
char 'f' :  wrist_angle_ref =  wrist_angle_ref - 0.010
char 't' :  wrist_rotation_ref =  wrist_rotation_ref + 0.010
char 'g' :  wrist_rotation_ref =  wrist_rotation_ref - 0.010
```

Step 6: Use the command ctrl+c in the terminal to quit the program. Quitting the program will cause Gazebo to close and will open 5 different plots. Each of the plots displays the actual and the reference trajectory for one of the joints. An example plot of the shoulder trajectory is shown in Figure 11 below.

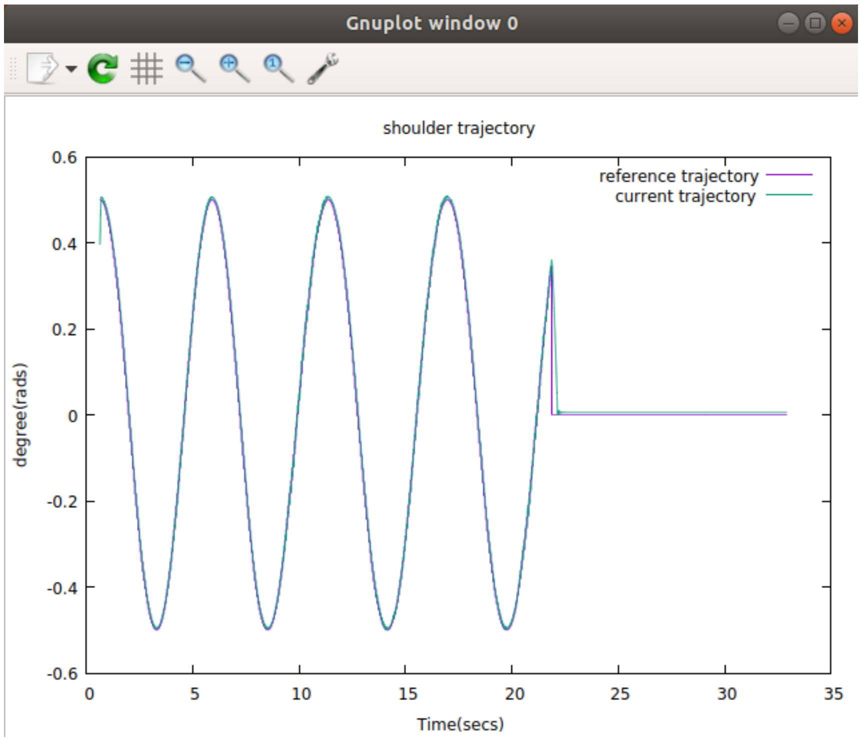


Figure 11: Plot of the shoulder joint actual and reference trajectory

3.1.4 Troubleshooting for Gazebo Simulation

- If the updated end effector position which is updated in the text file does not reflect in the gazebo model , try choosing different end effector position, this end effector position might be out of range. Refer to the figure 18 for the end effector range.
- While launching the gazebo file, If you get an error related to the log file exceeded the 1GB space. You need to clear the log file in the ros folder using the following steps:

– move to the ros log folder

```
$ cd ~/.ros/log
```

– Clear the log files

```
$ rm -rf *
```

3.2 Simulink

3.2.1 Introduction to Simulink Model

This section outlines the process of running the Simulink model for the WidowX 200 robotic arm. The user can input an end effector position, axis, and gripper angle into the widowx200_main.m script (Step 9 of Section 3.2.2). Based on these parameters, the inverse_kinematics.m function computes the necessary joint angles as described in Section 2.1. The user is also able to input the desired trajectory planning method using the which_path variable within the widowx200_main.m script. When which_path equals zero, the LSPB method (Section 2.2.1) is used, and when which_path equals one, the Quintic Polynomial Method (Section 2.2.2) is used. Within the widowx200_main.m script, the parameters for PID control (Section 2.3) and the LQR method (Section 2.4) can also be altered before running the simulation. These parameters are further described below in Step 11 of Section 3.2.2. When widowx200_main.m is run, the Simulink model is called, which runs the simulation and a MATLAB function to compute the necessary torques at each time step and apply them to the model. The Simulink simulation creates a Simscape Multibody model and plots the trajectories and torque inputs to each joint as shown in Figure 17.

3.2.2 Simulink Setup

Step 1: Download and unzip the folder containing the Simulink model and respective MATLAB codes.

Step 2: Open the widowx200_sim_model.slx file in Simulink. Before running the simulation, the path to the STEP files will need to be specified.

Step 3: Each of the link blocks are highlighted in Figure 12 below. Steps 4-6 will need to be applied to each of these blocks.

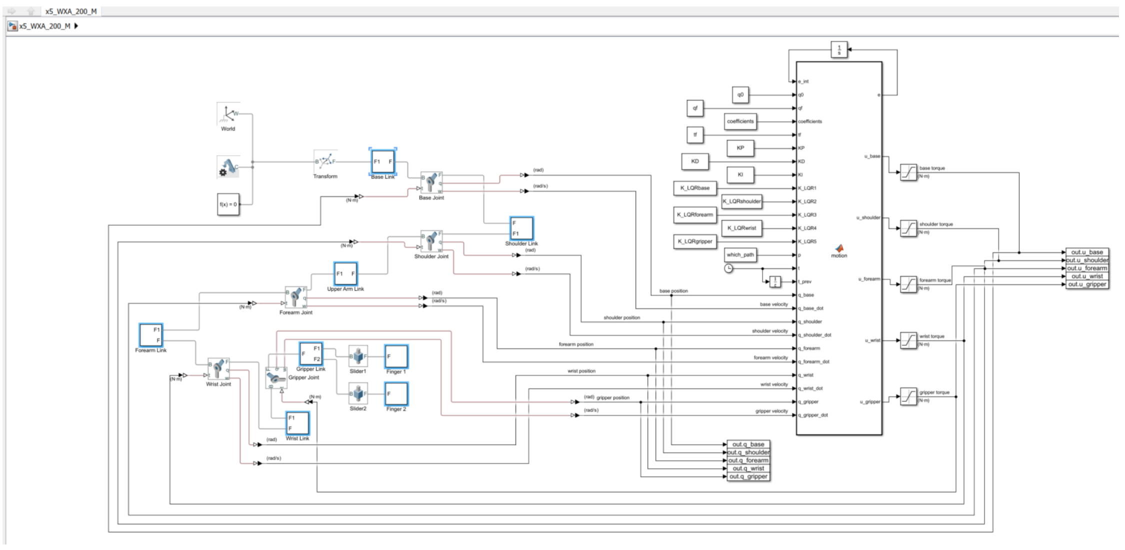


Figure 12: Carry out steps 4 through 6 for each of the links selected (highlighted in blue)

Step 4: Double clicking one of the link blocks will open the following subsystem (Figure 13).

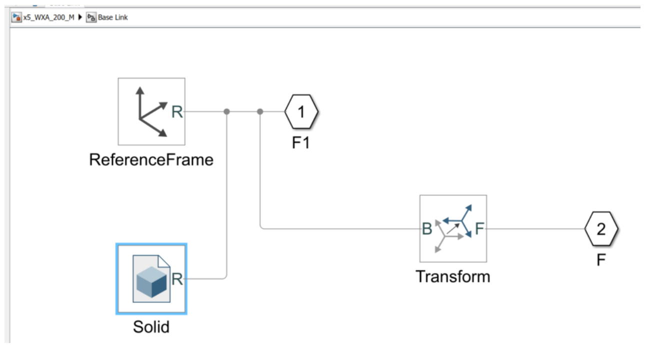
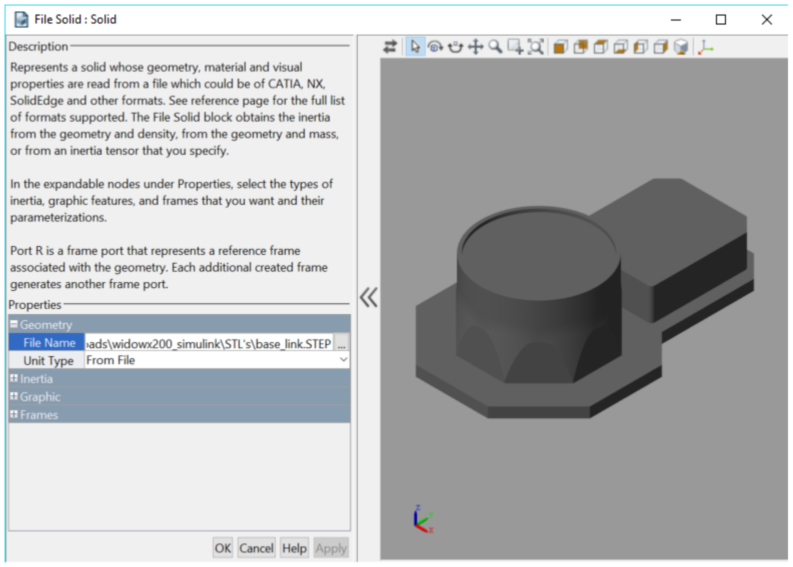


Figure 13: Base link subsystem

Step 5: Double click on the solid block to open the pop-up below (Figure 14). Then, within the Geometry drop-down, go to File Name, and click on the ellipsis to open the file explorer.


File Solid : Solid

Description

Represents a solid whose geometry, material and visual properties are read from a file which could be of CATIA, NX, SolidEdge and other formats. See reference page for the full list of formats supported. The File Solid block obtains the inertia from the geometry and density, from the geometry and mass, or from an inertia tensor that you specify.

In the expandable nodes under Properties, select the types of inertia, graphic features, and frames that you want and their parameterizations.

Port R is a frame port that represents a reference frame associated with the geometry. Each additional created frame generates another frame port.

Properties

Geometry

File Name	›ads\widowx200_simulink\STL's\base_link.STEP
Unit Type	From File

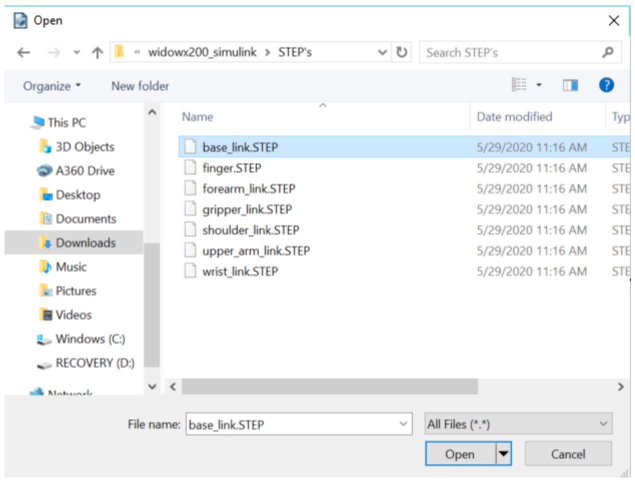
Inertia

Graphic

Frames

Figure 14: Solid block pop-up. Click on the ellipsis to open file explorer

Step 6: Within the file explorer (Figure 15), go to the downloaded folder and open the ”STEP’s” folder, which contains all of the 3d STEP files. Select the STEP file corresponding to the link block you have opened.



Name	Date modified	Typ
base_link.STEP	5/29/2020 11:16 AM	STE
finger.STEP	5/29/2020 11:16 AM	STE
forearm_link.STEP	5/29/2020 11:16 AM	STE
gripper_link.STEP	5/29/2020 11:16 AM	STE
shoulder_link.STEP	5/29/2020 11:16 AM	STE
upper_arm_link.STEP	5/29/2020 11:16 AM	STE
wrist_link.STEP	5/29/2020 11:16 AM	STE

File name: base_link.STEP

Figure 15: Select the STEP file with the same name as the link block

Step 7: After completing steps 4-6 for one of the link blocks, navigate back to the main model by clicking on the yellow highlighted tab below in Figure 16. Then, repeat steps 4-6 for all of the link blocks selected in Figure 12.

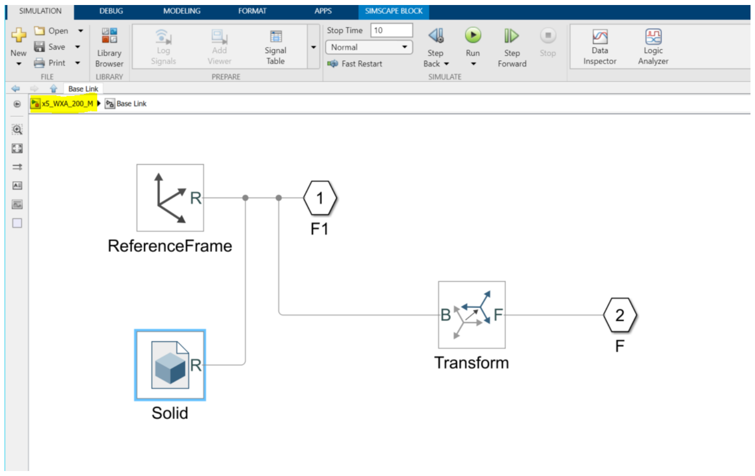


Figure 16: Click on the highlighted tab to go back to the overall Simulink model

Step 8: Now that all of the 3d models for the links have been selected, open the widowx200_main.m script from the downloaded folder.

Step 9: The desired position and orientation of the end effector should be selected by altering the p vector (position), alpha angle (gripper angle), and z_5 vector (end effector axis) as desired. Further explanation of these parameters can be found in section 2.1.

Step 10: Select the trajectory method by setting the which_path variable to either 0 (LSPB – Section 2.2.1) or 1 (Quintic Polynomial – Section 2.2.2). The detailed reference trajectory options can be found in the LSPB.m and the quintic_polynomial.m functions.

Step 11: The omega, zeta, Ki, r11, r22, and r2 values for each joint can be altered within the script to tune the PID controller (Section 2.3) and LQR method (Section 2.4) to meet your needs.

- Omega: natural frequency

- Zeta: damping ratio
- K_i: penalizes error history
- r_{11}: increases effort to reach position
- r_{22}: increases effort to reach velocity
- r_2: decreases steady state error

3.2.3 Running the Simulink Simulation

Once the trajectory method has been chosen and the controller tuned, the widowx200_main.m script can be run to produce the simulation. The simulation contains a Simscape Multibody model of the WidowX 200 following the respective trajectory as well as a plot of the actual and reference trajectories and the torques for each joint. The end effector position with respect to the range of the robotic arm is also plotted (Figure 18). If the desired position is outside the range, an error message will be displayed. The Simscape Multibody model and the example trajectory of one of the joints can be seen in Figure 17 below. If any issues occur, refer to Section 3.2.4 for troubleshooting tips.

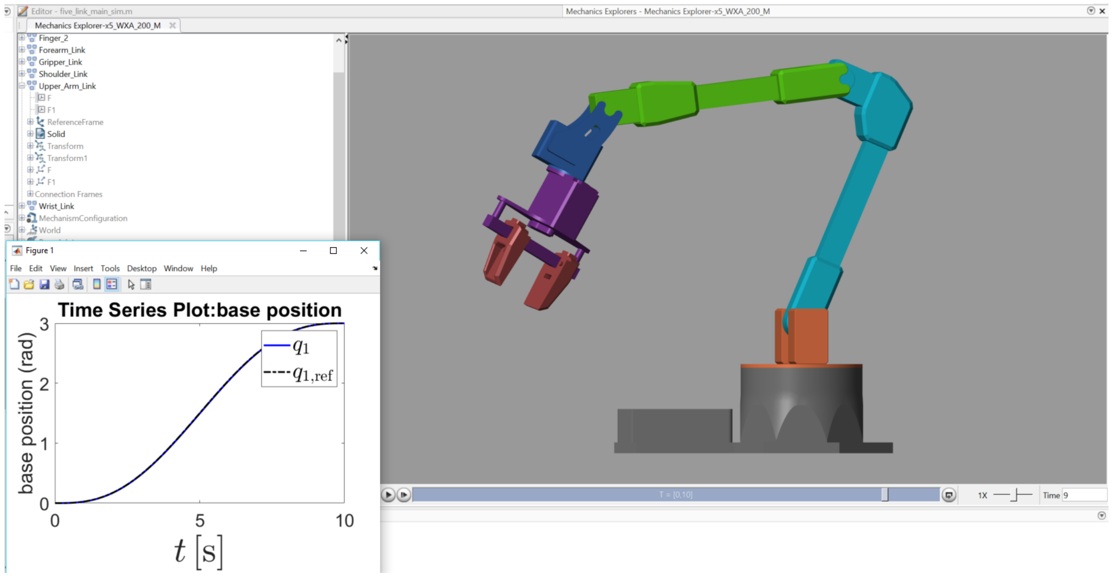


Figure 17: Simscape Multibody model and plot of wrist joint angle with reference trajectory

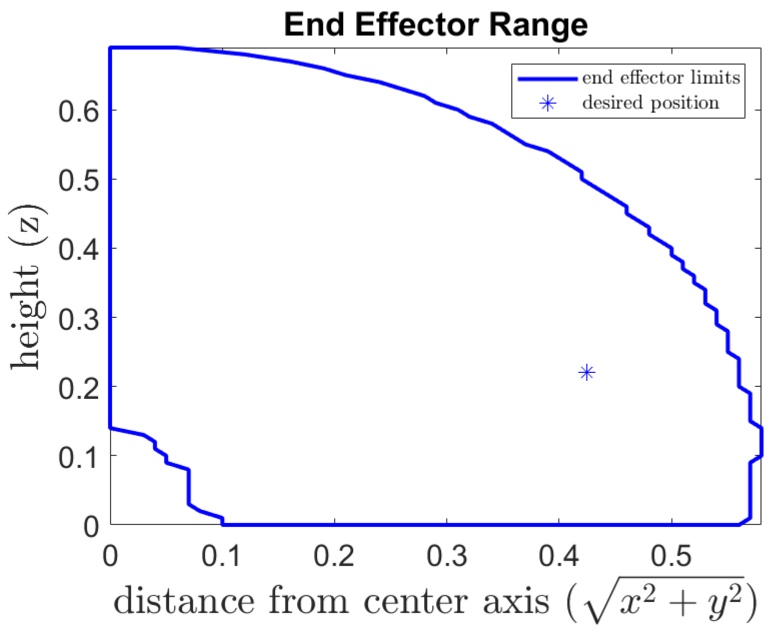


Figure 18: Graph showing the range of the end effector of the WidowX 200 Robotic Arm

3.2.4 Troubleshooting the Simulink Simulation

If you are encountering errors running the Simulink simulation, please read the error messages before reading this section. If the errors suggest choosing a different end effector position, check the end effector range graph shown in Figure 18 above and choose a point within the polygon. Keep in mind that the x-axis of the graph is the distance from the central axis including both the x and y components of the position.

If you are encountering an error suggesting a new end effector axis, try setting the axis as the same or similar to the p vector for further distances from the base or setting it as more upward or downward with relatively high positive or negative z-values for positions closer to the base.

If the any of the joints are having difficulty following the reference trajectory, check the plot of the torque for that joint. If it is reaching 8.2 Nm for the shoulder joint or 4.1 Nm for any other joint, the torque is at a maximum. This may mean that the arm has difficulty reaching the position because it is near the end of the range or the simulation time is too low. If the torque is not at a maximum, the PID and LQR parameters should be altered to tune the response.

3.2.5 Code Explanation

The code for the Simulink model is comprised of one main script and a set of MATLAB functions used to carry out different aspects of the simulation. The widowx200_main.m script first allows for the user to enter desired parameters such as the desired trajectory method, end effector position and orientation, and PID and LQR gain parameters.

When the main script is run, it calls the check_position.m function, which determines whether the desired position is within the range of the robotic arm or has any conflicts with the base or microcontroller box. The range is defined by the simulink_model_bounds.mat file. If the desired position is outside the range, an error is displayed and the simulation is terminated, and if it is near the end of the range, a warning is displayed before running the simulation. If the desired position is well within the range, a blue plot is displayed and the simulation runs normally.

After checking the position, the main function calls the inverse_kinematics.m function, which computes the joint angles required to meet the desired end effector position and orientation. If any of the angles are outside the range of motion of a joint, an error message is displayed and the simulation terminated.

Next, the main script runs the Simulink model itself. The Simulink model contains the 3d files of the links and a MATLAB function (motion.m) that is run at each time step.

The motion.m function calls the widowx200_dynamic_model.m which, at each time step, computes the parameters for the equations of motion, defines the reference trajectories using the LSPB.m or quintic_polynomial.m function, and computes the LQR parameters. The widowx200_dynamic_model.m function also calls the control_law_perfect_cancellation.m function, which uses PID control to calculate the torque to be applied to each joint.

After computing the torques, the widowx200_dynamic_model.m function sums the torques from the PID controller and LQR method and passes them back to the motion.m function, which passes the torques back to the Simulink model.

The Simulink model applies the calculated torques to each of the joints and determines the response using the ODE15s variable step, variable order solver.

This process is repeated until the total time is equal to the specified simulation time. After completing the simulation, the widowx200main.m script creates plots of the actual and reference trajectory of each joint and the torque applied to each joint.

4 Actual Robot

4.1 Setup - Widow X 200

This section contains the detailed Steps on how to setup the Widow X 200 Robotic Arm for Ubuntu and Windows.

The Library setup of how to build the library in C and use the library in MatLab, Ubuntu, and Windows is as per the steps given in manual from the following link.

`https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/#dynamixel-sdk`

4.1.1 Setup for Ubuntu

Step 1: Download the Dynamixel SDK folder from the following link. Download the DynamixelSDK 3.7.21 release.

```
https://github.com/ROBOTIS-GIT/DynamixelSDK/releases
```

Step 2: Copy the downloaded DynamixelSDK-3.7.21 folder in the home folder. This step is important for actual robot to load the Dynamixel library.

Step 3: Check for the gcc compiler version. The gcc version should be GNU gcc ver. 5.4.0 20160609 or higher.

```
$ gcc -v
```

Step 4: To download the GCC compiler run the following command in the terminal.

```
$ sudo apt-get install gcc-5
```

Step 5: To install the dependent packages run the following command in the terminal.

```
$ sudo apt-get install gcc-multilib g++-multilib
```

Step 6: In order to build the library, traverse to the folder depending on the format to built(32 or 64 bit).
For example traverse to [DynamixelSDK folder]/c/build/linux64] for 64 bit platform. Open the terminal and build the make file.
To install the library to the root directory run the following command.

```
$ sudo make install
```

Step 7: After installation of the library open MATLAB. Go to the Home tab and click on the set path button under environment.
Click on the button add with subfolders and add the DynamixelSDK folder to the path list

Step 8: To make the port usable, run the following command in the terminal.

```
$ sudo chmod a+rw /dev/ttyUSB0
```

4.1.2 Setup for Window

`https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/library_setup/c_windows/#c-windows`

MATLAB API References

`https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/api_reference/matlab/matlab_porthandler/#matlab`

Include connections and library setups.

4.2 Code Explanation

To understand how to control the Dynamixel XM and XL motors, it is important to go through the data sheets of these motors. The following are the links for XM and XL data sheets:

XM

`https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/#control-table-data-address`

XL

`https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/#control-table`

Refer the MATLAB APIs in the links given in the setup portion to see how to Read/Write commands.

To control the motor's torque, It has to be set in the 'Current' mode by performing the following steps:

Basic read and write for the motors can be done as follows:

- Read:
  read1ByteTxRx(PortNum, PROTOCOLVERSION, ID,ADDRESS);

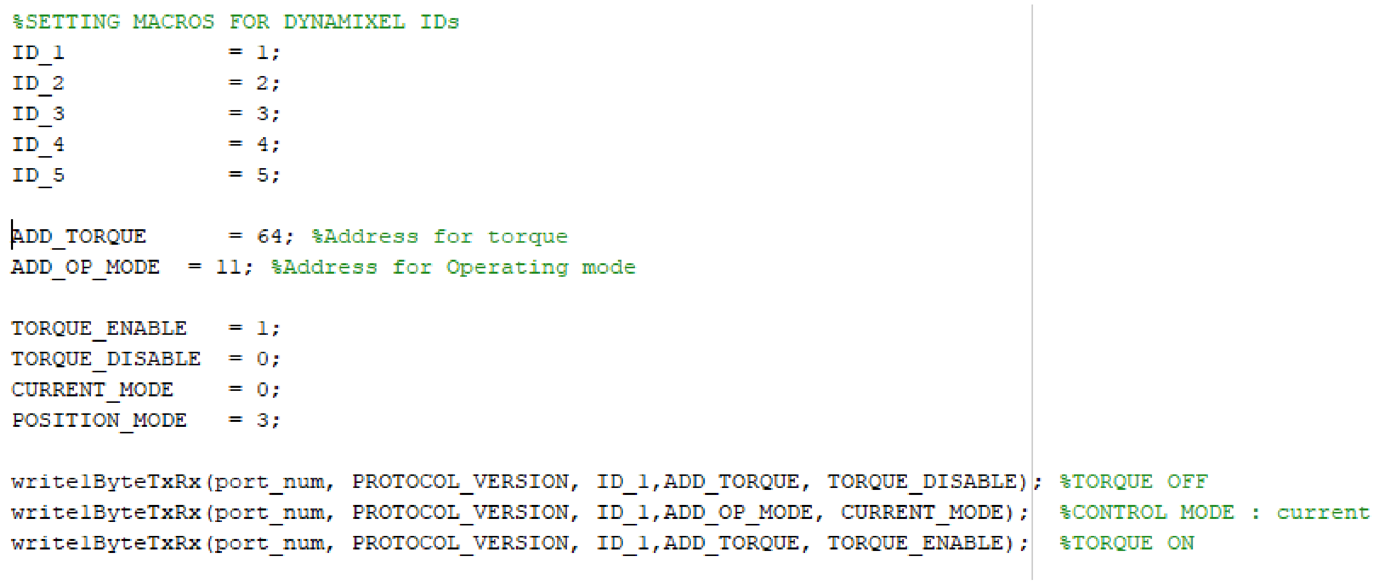
  This command reads the number of bytes in the address specified in the command. The number of bytes to be read in the command changes as per the quantity to be measured and can be found in the control table.

- Write:
  write1ByteTxRx(PortNum, PROTOCOLVERSION, ID,ADDRESS,COMMAND);

  This command writes the specified 'COMMAND' in the specified Address of the Dynamixel given by the ID. Writing different values requires different numbers of bytes to be written which can be changed in the command. Refer to the control table to check the number of bytes to be written.

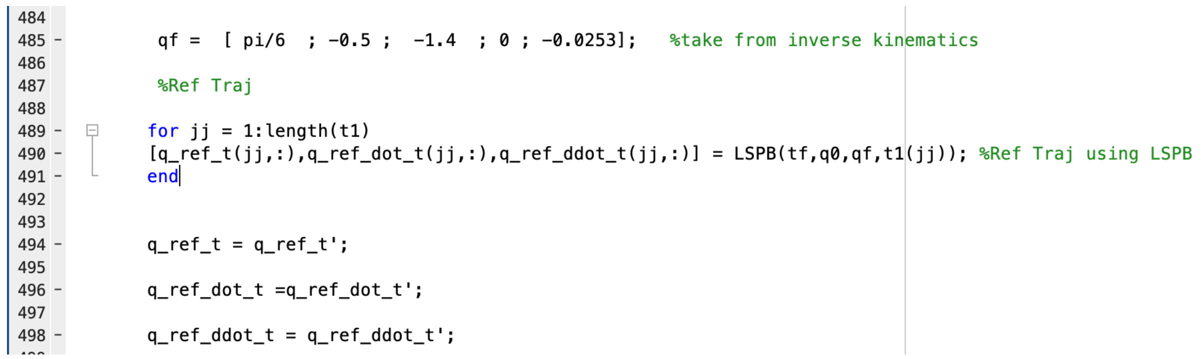
Initializing motor to get required torque :

- Turn motor torque off (addr: 64, length: 1 byte , value: 0)
- Set operating mode to 'current' (addr: 11, length: 1 byte , value: 0)
- Turn motor torque on (addr: 64, length: 1 byte , value: 1)
- Provide the required current for proportional torque (addr: 102, length: 2 byte , value: acc to torque)



```
%SETTING MACROS FOR DYNAMIXEL IDs
ID_1            = 1;
ID_2            = 2;
ID_3            = 3;
ID_4            = 4;
ID_5            = 5;

ADD_TORQUE      = 64; %Address for torque
ADD_OP_MODE  = 11; %Address for Operating mode

TORQUE_ENABLE   = 1;
TORQUE_DISABLE  = 0;
CURRENT_MODE    = 0;
POSITION_MODE   = 3;

write1ByteTxRx(port_num, PROTOCOL_VERSION, ID_1,ADD_TORQUE, TORQUE_DISABLE); %TORQUE OFF
write1ByteTxRx(port_num, PROTOCOL_VERSION, ID_1,ADD_OP_MODE, CURRENT_MODE);  %CONTROL MODE : current
write1ByteTxRx(port_num, PROTOCOL_VERSION, ID_1,ADD_TORQUE, TORQUE_ENABLE);  %TORQUE ON
```

Figure 19: Setup for initializing a motor in current mode for torque control

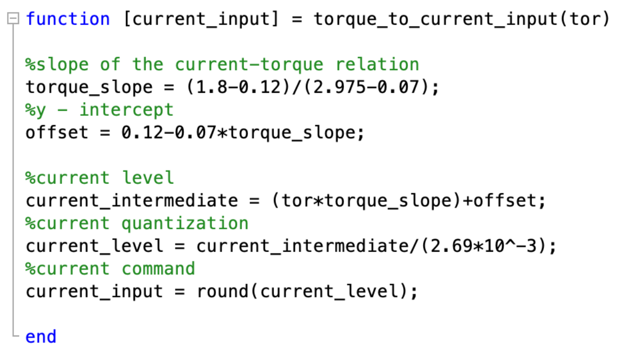


```
     qf =  [ pi/6  ; -0.5 ;  -1.4  ; 0 ; -0.0253];   %take from inverse kinematics

     %Ref Traj

    for jj = 1:length(t1)
    [q_ref_t(jj,:),q_ref_dot_t(jj,:),q_ref_ddot_t(jj,:)] = LSPB(tf,q0,qf,t1(jj)); %Ref Traj using LSPB
    end


    q_ref_t = q_ref_t';

    q_ref_dot_t =q_ref_dot_t';

    q_ref_ddot_t = q_ref_ddot_t';
```

Figure 20: Setting the final position for motors according to the inverse kinematics output, acquiring the reference inputs from LSPB

In the continuous loop, the following tasks are performed:

- Check current positions, and convert the position readings into radians.
- Calculate joint velocities
- Calculate position error and velocity error $(e, \dot{e})$
- Calculate and update $M(q), C(q, \dot{q}), K_g$
- Calculate u_{lqr}
- Calculate joint torque
- Convert joint torque into an input current signal to the motors. This is done with the 'torque to current input' function which returns a current value for an input torque. (Code snippet in Figure 21 below)



```
function [current_input] = torque_to_current_input(tor)

%slope of the current-torque relation
torque_slope = (1.8-0.12)/(2.975-0.07);
%y - intercept
offset = 0.12-0.07*torque_slope;

%current level
current_intermediate = (tor*torque_slope)+offset;
%current quantization
current_level = current_intermediate/(2.69*10^-3);
%current command
current_input = round(current_level);

end
```

Figure 21: Function used to convert required Torque to Current input byte for Dynamixel

$$u = M(q)[\ddot{q}_{ref} - K_p e - K_d \dot{e} - Kie_{integral}] + C(q, \dot{q})(\dot{q}) + K_g + U_{lqr}(e, \dot{e})$$

4.3 Trouble Shooting

- Motor Overload: If the motor LED flashes at any point, the motor must be reset. It can be done by unplugging the power from the motor. It can also be done by connecting DynamixelWizard and resetting the motor.
- Failed to open port: If the program gives an error 'Failed to open Port', there are a couple of issues which could be causing this.
  1: Com port error: the wrong COM port has been initiated in the code. Check the 'DEVICENAME' and make sure the correct USB port is being used.

  2: Unplug the power cable and the USB cable and restart the program.
- If at any point, the robotic arm starts to vibrate uncontrollably, unplug the power chord.

5 Using the App

The app brings all the three simulations together in a user friendly interface.

5.1 GUI Setup

To Use the App. The application has to be setup using the following steps.

- The interbotix_ws worskapce has to be created as shown in the section 3.1.2. This creates the workspace and also does the setup of running the gazebo model using GUI.
- For simulink to work properly, the path of all the STEP files needs to be specified. Follow the steps given in section 3.2.2. The Simulink folder is already downloaded in the Interbotix_src folder used in the workspace setup. This folder contains all the STEP files and the simulink file.
- For running the actual robotic arm. The Dynamixel SDK folder needs to be installed and built in the home folder. Follow the steps given in section 4.1.1 to build the dynamixel library.

5.2 App Navigation

- Open the terminal and type:

```
$ matlab -r "cd ~/interbotix_ws/src/Interbotix_src/GUI/"
```

- The Matlab app will open up in ~/interbotix_ws/src/interbotix_src/GUI folder and now open 'RoboticArmMain.mlapp'.
- The App Designer will open up. Navigate to the toolstrip and click the 'Run' button.
- An App will open up as shown in Figure 22 below:

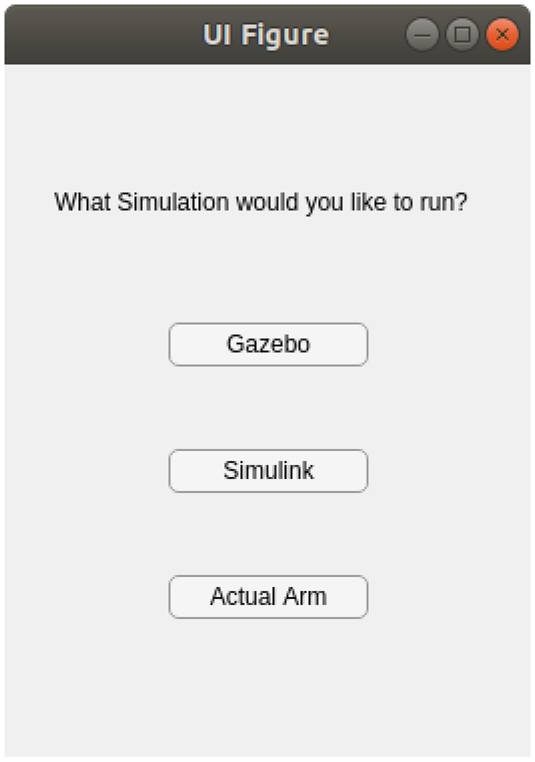
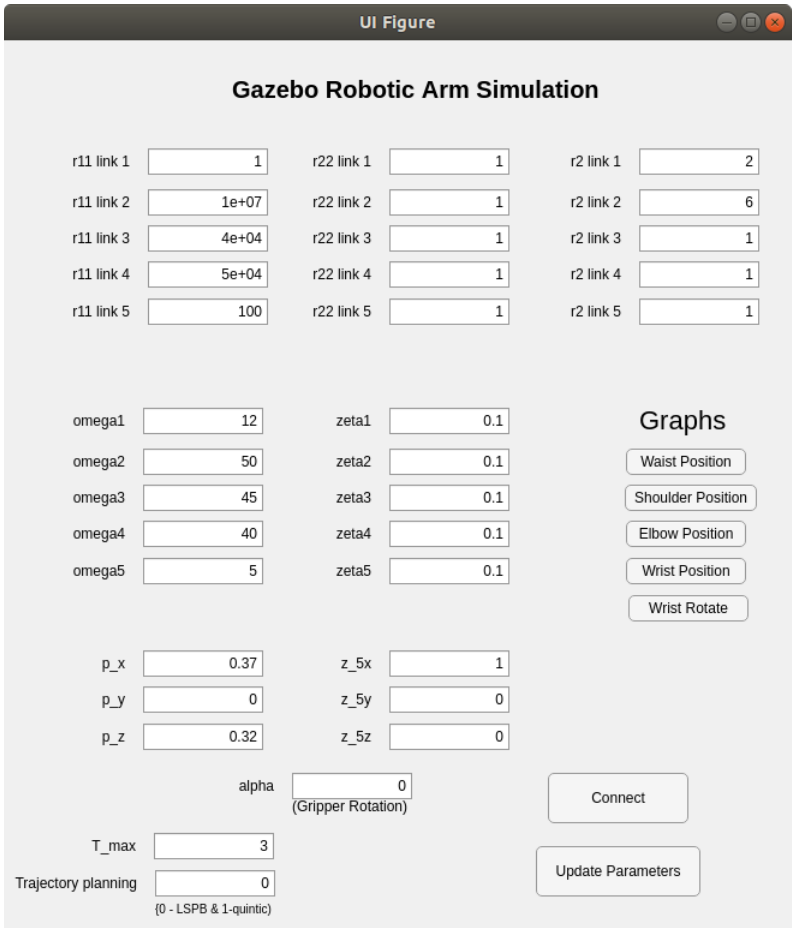


Figure 22: Robotics arm Simulation app

- This app has three Simulation options: Gazebo, Simulink, and Actual Arm. Click on a button to navigate to the user interface for their corresponding simulations

5.3 Gazebo

- If 'Gazebo' button is clicked from the RoboticArmMain app, another user interface will open up as shown in Figure 23 below. This interface have several parameters which can be manipulated to change the simulation of the Gazebo Robotic arm model.



Gazebo Robotic Arm Simulation

r11 link 1	1	r22 link 1	1	r2 link 1	2
r11 link 2	1e+07	r22 link 2	1	r2 link 2	6
r11 link 3	4e+04	r22 link 3	1	r2 link 3	1
r11 link 4	5e+04	r22 link 4	1	r2 link 4	1
r11 link 5	100	r22 link 5	1	r2 link 5	1
omega1	12	zeta1	0.1		
omega2	50	zeta2	0.1		
omega3	45	zeta3	0.1		
omega4	40	zeta4	0.1		
omega5	5	zeta5	0.1		
p_x	0.37	z_5x	1		
p_y	0	z_5y	0		
p_z	0.32	z_5z	0		

alpha 0 (Gripper Rotation)

T_max 3

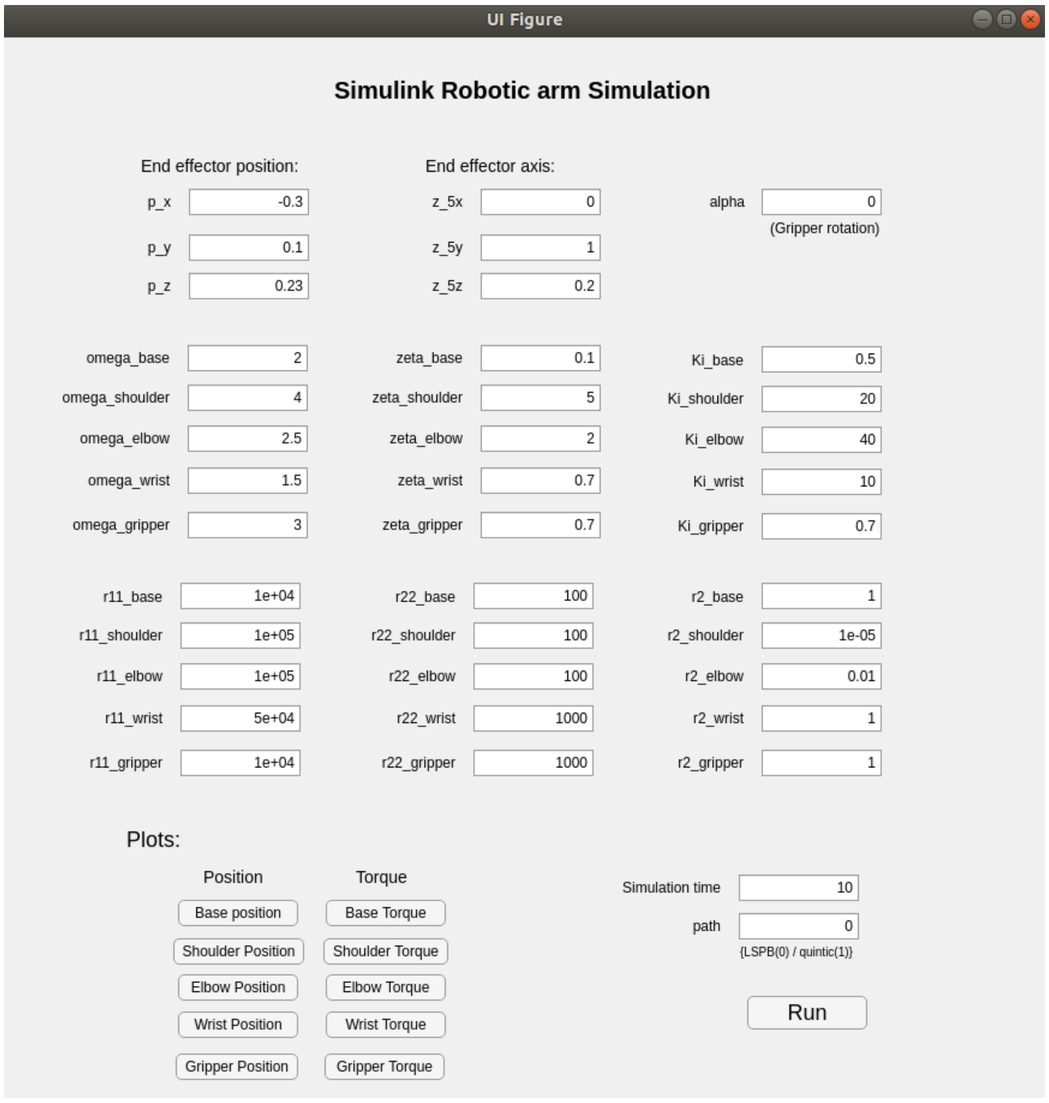
Trajectory planning 0 {0 - LSPB & 1-quintic}

Figure 23: Gazebo Simulation app

- Set the end effector position (p_x, p_y, and p_z) and axes (z_5x, z_5y, and z_5z).
- Specify the trajectory planning (0 for LSPB and 1 for quintic polynomial)
- Specify the T_max time, T_max is the time in which the change of position should be made. Its the time used in Trajectory planning.
- Click the 'Connect' button
- A gazebo model simulating the robot arm will open up. A user can observe a robotic arm moving to the specified end effector position.
- To move and arm to another position, change the end effector parameters and click on 'Update Parameters' button.
- It can now be observed that the robotic arm moves to a new specified position.
- Click on the position graphs buttons to see the trajectory of the joint angles.

5.4 Simulink

- If 'Simulink' button is clicked from the RoboticArmMain app, another user interface will open up as shown in the Figure 24 below.


UI Figure

Simulink Robotic arm Simulation

End effector position:

End effector axis:

p_x	-0.3	z_5x	0	alpha	0
				(Gripper rotation)	
p_y	0.1	z_5y	1		
p_z	0.23	z_5z	0.2		
omega_base	2	zeta_base	0.1	Ki_base	0.5
omega_shoulder	4	zeta_shoulder	5	Ki_shoulder	20
omega_elbow	2.5	zeta_elbow	2	Ki_elbow	40
omega_wrist	1.5	zeta_wrist	0.7	Ki_wrist	10
omega_gripper	3	zeta_gripper	0.7	Ki_gripper	0.7
r11_base	1e+04	r22_base	100	r2_base	1
r11_shoulder	1e+05	r22_shoulder	100	r2_shoulder	1e-05
r11_elbow	1e+05	r22_elbow	100	r2_elbow	0.01
r11_wrist	5e+04	r22_wrist	1000	r2_wrist	1
r11_gripper	1e+04	r22_gripper	1000	r2_gripper	1

Simulation time 10

path 0

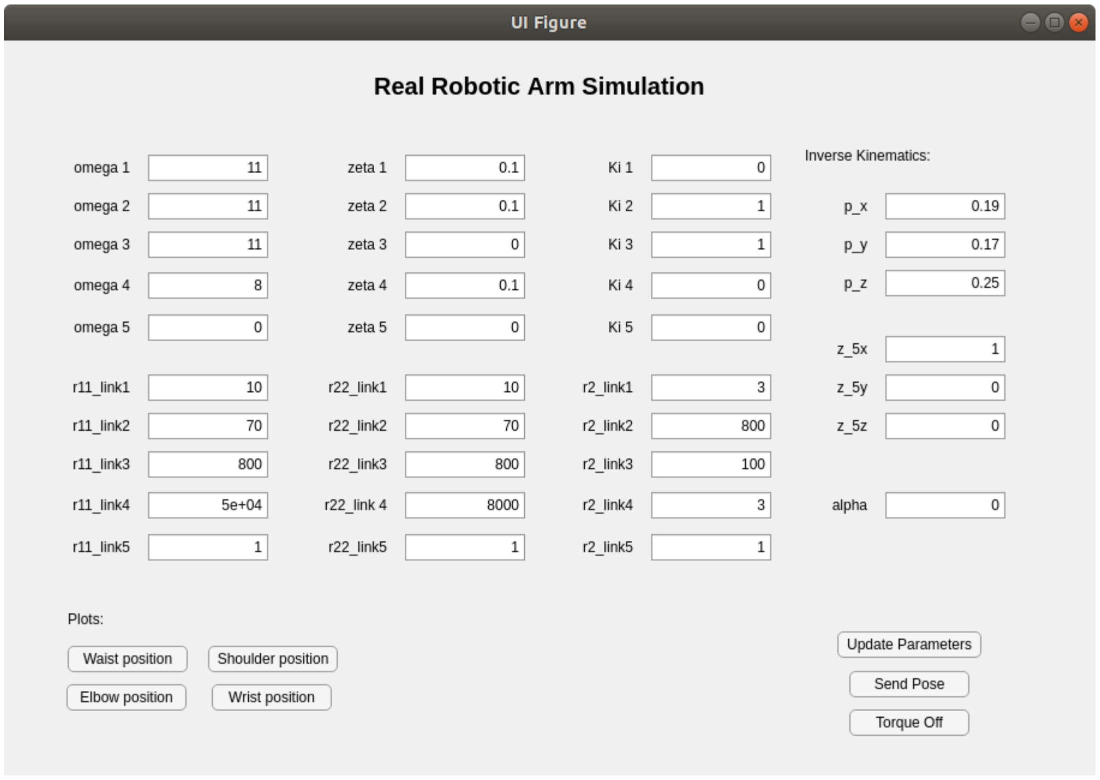
{LSPB(0) / quintic(1)}

Figure 24: Simulink Simulation app

- Set the end effector positions, axes, alpha, simulation time, path preference and other parameters.
- Click the 'Run' button.
- A simulink simulation representing the Robotic arm moving to the specified end effector position will open up.
- After the simulation is complete, click on the position and torque plot buttons to see the respected trajectories.

5.5 Actual Robot

- If 'Actual Arm' button is clicked, another user interface will open up as shown in Figure 25 below.



omega 1	11	zeta 1	0.1	Ki 1	0		
omega 2	11	zeta 2	0.1	Ki 2	1	p_x	0.19
omega 3	11	zeta 3	0	Ki 3	1	p_y	0.17
omega 4	8	zeta 4	0.1	Ki 4	0	p_z	0.25
omega 5	0	zeta 5	0	Ki 5	0	z_5x	1
r11_link1	10	r22_link1	10	r2_link1	3	z_5y	0
r11_link2	70	r22_link2	70	r2_link2	800	z_5z	0
r11_link3	800	r22_link3	800	r2_link3	100		
r11_link4	5e+04	r22_link 4	8000	r2_link4	3	alpha	0
r11_link5	1	r22_link5	1	r2_link5	1		

Figure 25: Real Arm Simulation app

- Set the omega, zeta, Ki, r11, r22, r2, and end effector parameters. Connect the power supply to the robot .Connect the micro USB cable from the Computer's USB port to the Robotic arm port
- Click the 'Update Parameters' button and then click the 'Send Pose' button to move the robotic arm.
- The real robotic arm moves to the specified position and stops there providing the necessary torque to stay at the specified position.
- To turn the torque off, click the 'Torque Off' button.
- After the simulation is complete, click the position plots button to see the respective trajectories of the dynamixel's positions.

5.6 App Troubleshoot

- Sometimes, the app does not lets a user to edit parameters in the edit field. If that happens, click on the title bar of the app and try to edit the parameters again.
- ERROR : 'atan2 invalid result' : In this case, the desired position of the end effector is out of bounds, input a new position and update parameters.
- 'TORQUE OFF' can only be used once the arm has reached its final position. In case of any undesired motion, UNPLUG the power immediately.