

MTRX5700 - Assignment Task 1

Report

Experimental Robotics

SIDs:
470148329
470416309
480436153

Question 1	2
Forward Kinematics - State 1	4
Forward Kinematics - State 2	5
Question 2	7
Question 3	9
Specifications	9
Assumptions	9
Workspace/Configuration Space	10
Forward Kinematics	11
Question 4	13
Appendix	15
Q2 code	15

Question 1

- a. The following section highlights the configuration of a six-axis arm robot (UR5e) with six rotational joints. *Figure 1*, highlights the structural outline of the robotic arm. *Figure 2*, denotes the present configuration in state $q[0, -\frac{3\pi}{2}, \frac{\pi}{2}, 0, 0, 0]$. *Figure 3*, depicts a top and side view of the state configuration, illustrating the 3D arrangement.

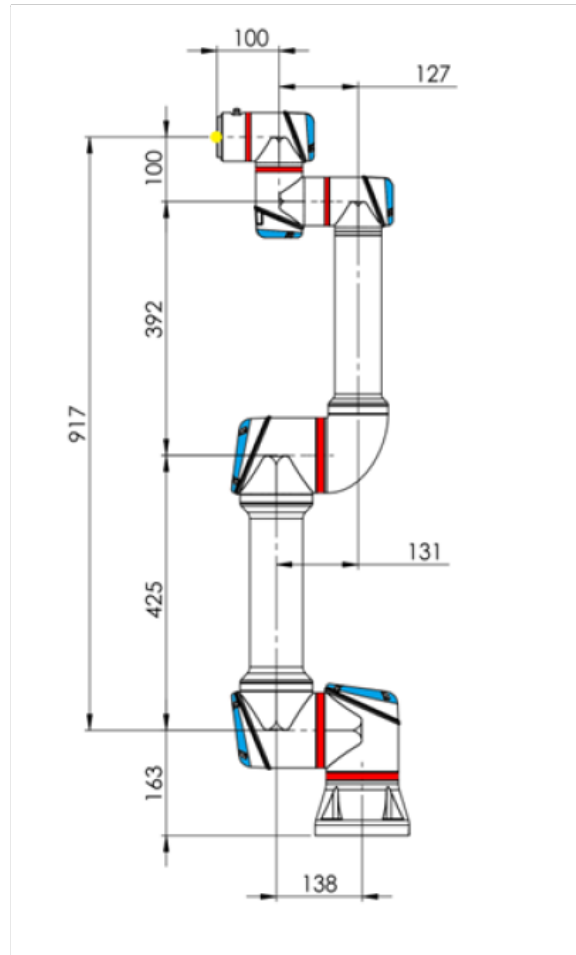


Figure 1 - Structural Representation

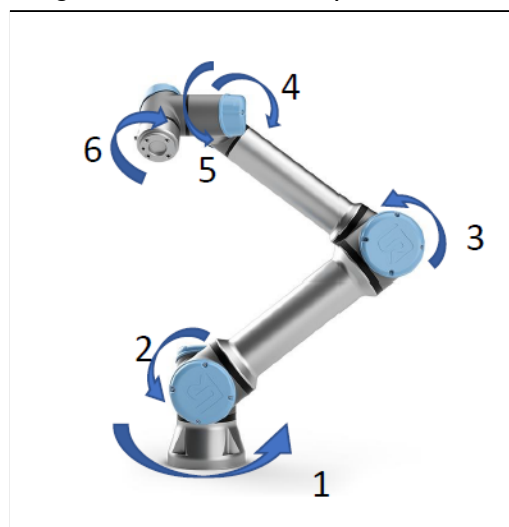


Figure 2 - State Configuration

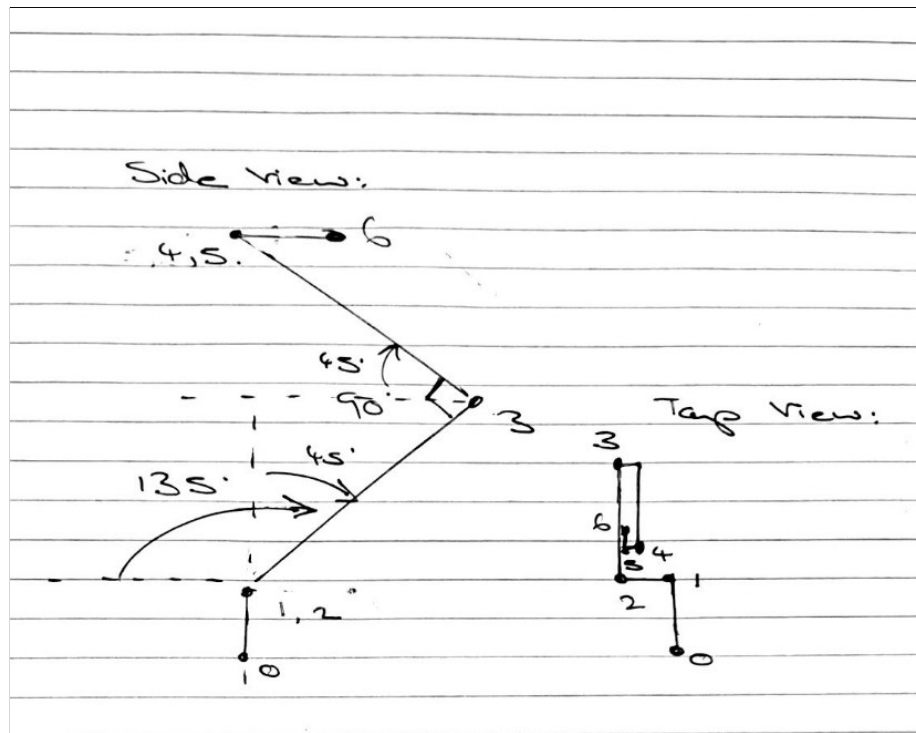


Figure 3 - Top & Side View

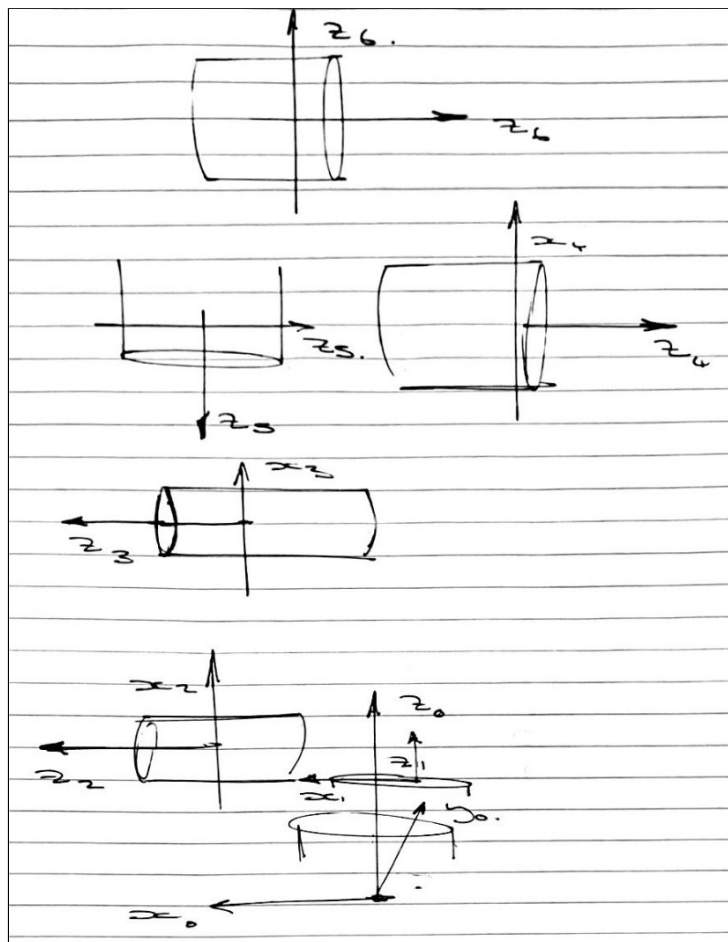


Figure 4 - Defined Frames

Forward Kinematics - State 1

Link (i)	d_i (mm)	θ_i (deg)	a_i (mm)	α_i (deg)
1	163	θ_1	0	$\frac{\pi}{2}$
2	0	θ_2	425	0
3	0	θ_3	392	0
4	127	θ_4	0	0
5	100	θ_5	0	$-\frac{\pi}{2}$
6	100	θ_6	0	$\frac{\pi}{2}$

Table 1 - Denavit-Hartenberg Parameters for State 1 (UR5)

$${}^{i-1}T_i \begin{bmatrix} \cos \theta & -\sin \theta & 0 & \alpha \\ \sin \theta * \cos \alpha & \cos \theta * \cos \alpha & -\sin \alpha & -d * \sin \alpha \\ \sin \theta * \sin \alpha & \cos \theta * \sin \alpha & \cos \alpha & \cos \alpha * d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\therefore {}^0T_6 = {}^0T_1 {}^1T_2 {}^2T_3 {}^3T_4 {}^4T_5 {}^5T_6$$

Equation 1 - Transformation Matrix

- b. The following section highlights the visualisation of the above calculated forward kinematics of state one:

Note: $\theta_2 = \frac{-3\pi}{2}$ & $\theta_3 = \frac{\pi}{2}$.

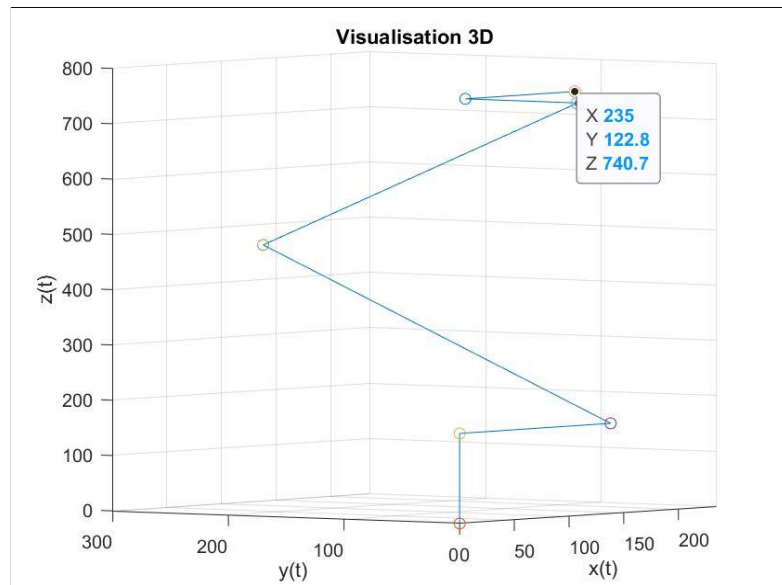


Figure 5 - Visualisation of Kinematic Equations

End Effector: (235, 122.8, 740.7)

- c. The following section highlights the investigation to locate the end effector of the state: $q[0, \frac{-\pi}{2}, 0, \frac{-\pi}{2}, 0, 0]$ also known as home position.

Forward Kinematics - State 2

Link (i)	d_i (mm)	θ_i (deg)	a_i (mm)	α_i (deg)
1	163	θ_1	0	$\frac{\pi}{2}$
2	0	θ_2	425	0
3	0	θ_3	0	0
4	127	θ_4	127	0
5	100	θ_5	0	0
6	100	θ_6	0	0

Table 2 - Denavit-Hartenberg Parameters for State 2 (UR5)

Note: $\theta_2 = \frac{-\pi}{2}$ & $\theta_4 = \frac{-\pi}{2}$.

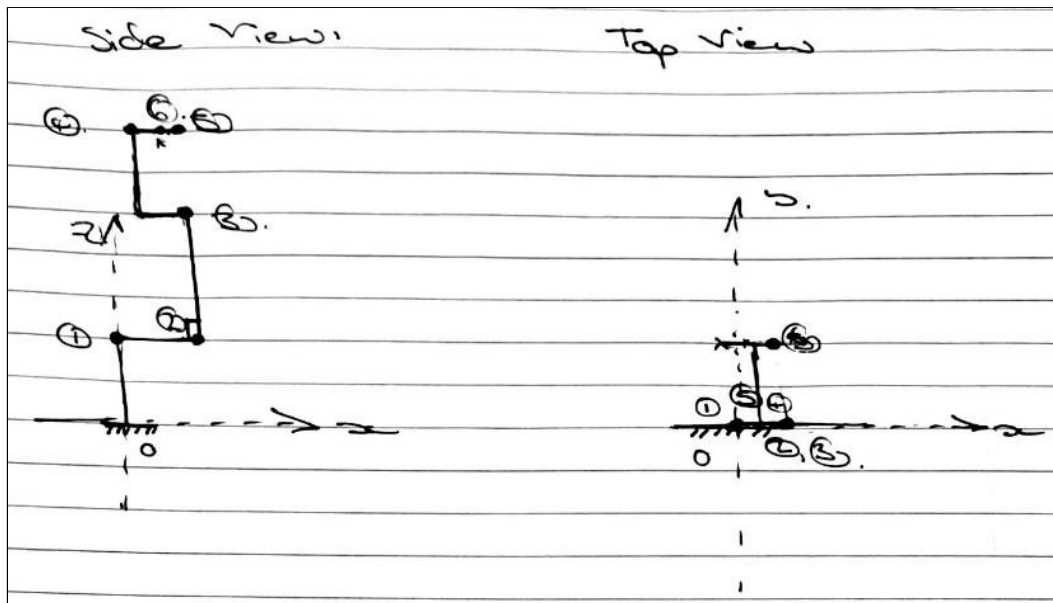


Figure 6 - Top & Side View

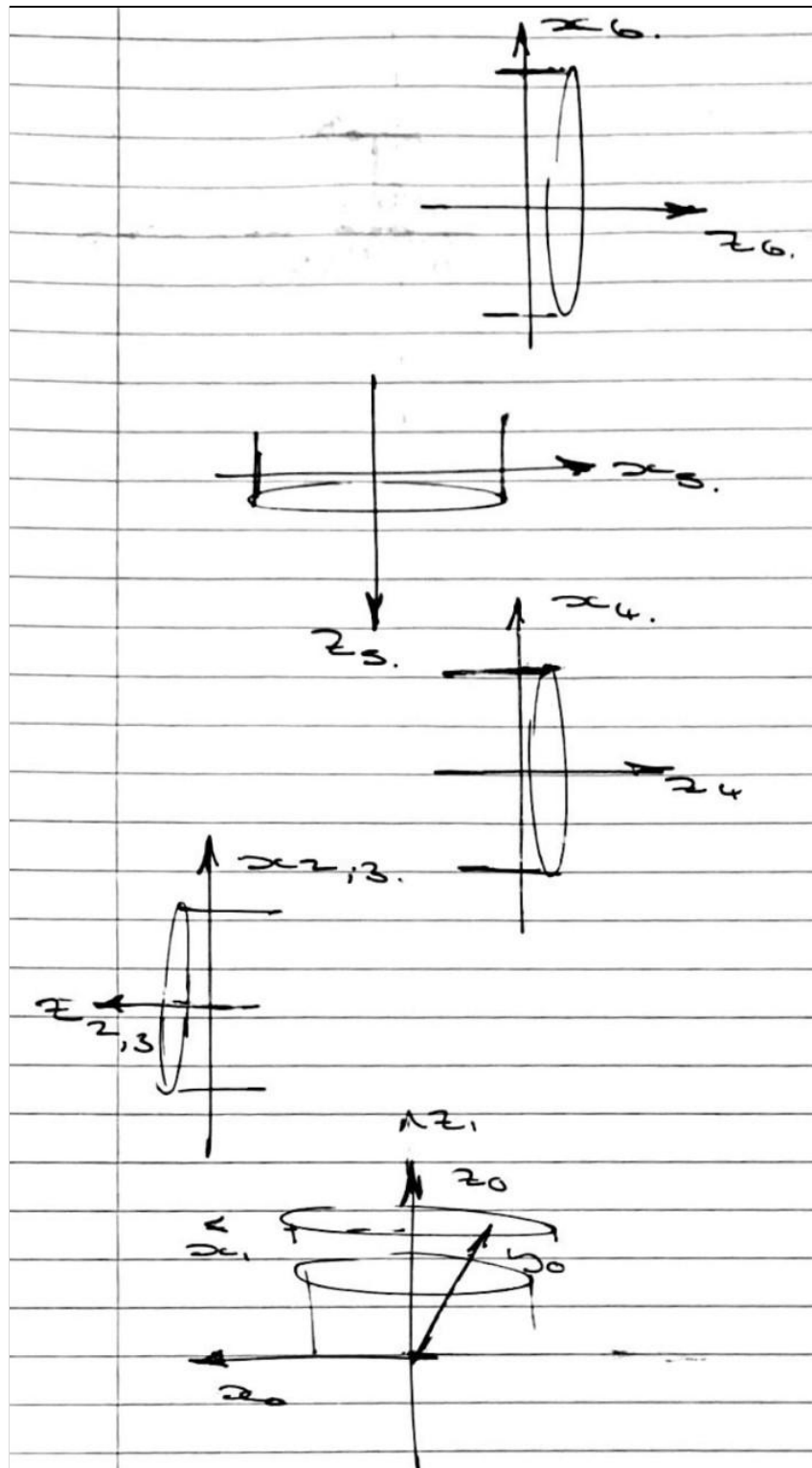


Figure 7 - Defined Frames

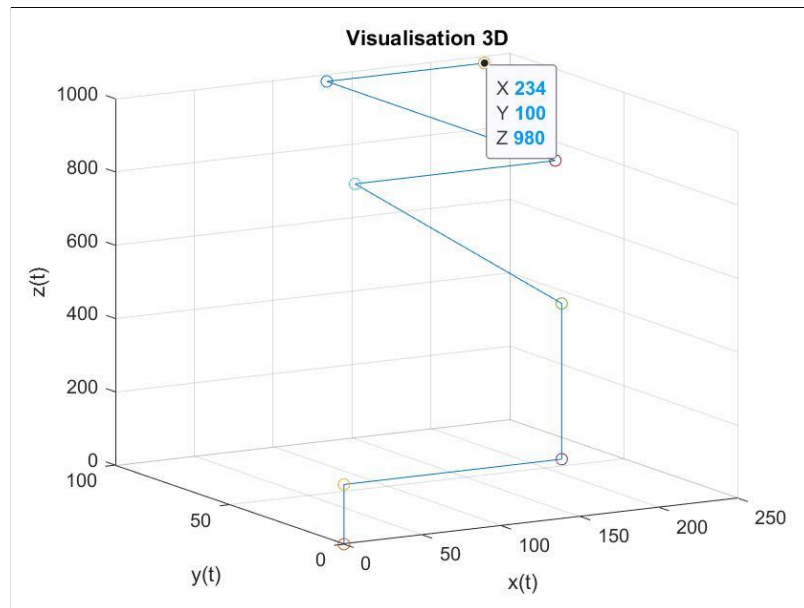


Figure 8 - Visualisation of Kinematic Equations

End Effector: (234, 100, 980)

Question 2

- This question submission is provided with code submission and also in the appendix.
- There are three ways to achieve obstacle avoidance in my code;
 - Straight down straight up:
 - Try to make the robotic arm run straight up and down and also move at the same height as possible to reduce the possibility of oblique movement to reduce random collisions as much as possible.
 - Include the intermedia waypoints:
 - Appropriately add some intermediate waypoints to avoid large horizontal rotation of the robot arm and reduce uncontrollable collision.
 - Adjust the direction of the gripper:
 - In the hardware test, the direction of the gripper should be the same as shown in Figure.2b to prevent accidental collision with the tower when grabbing the block. Obtaining the quaternion of the gripper by calculation, thereby changing the direction of the gripper.

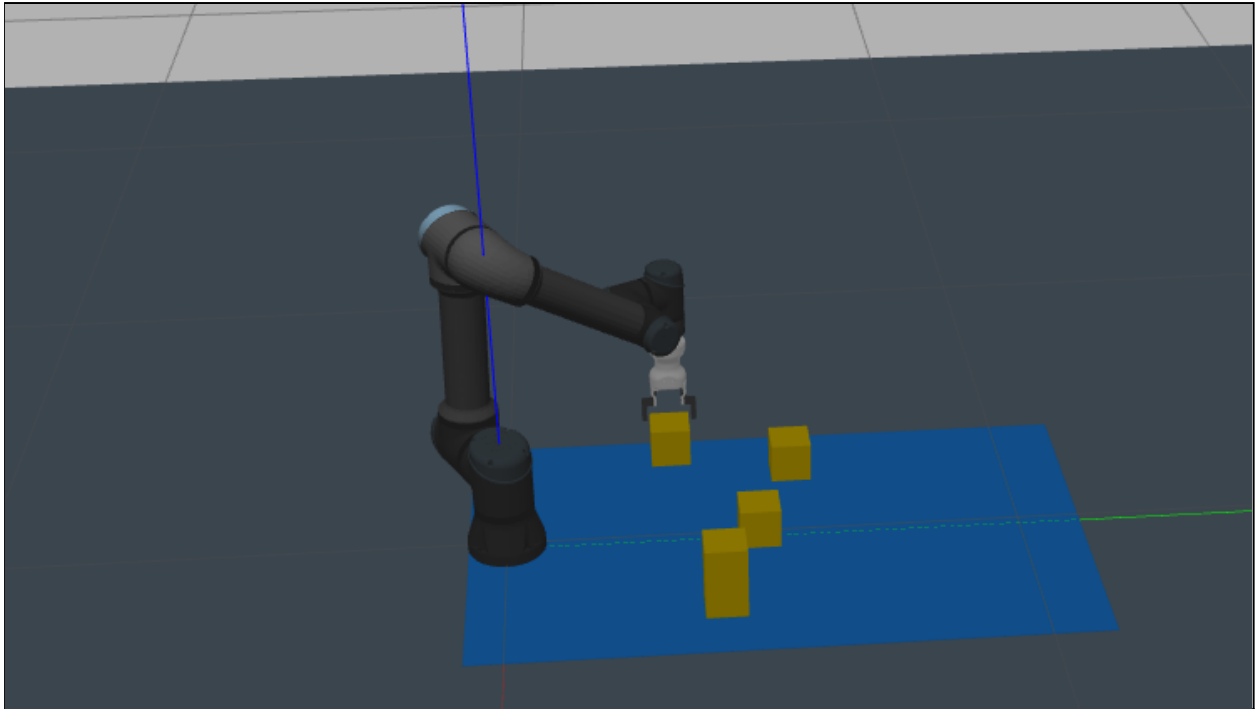


Figure 9 - The Direction of Gripper

```
def move_arm(mgpi, x, y, z):  
  
    print("-----")  
    print("move to:", x, y, z)  
  
    target_pose = geometry_msgs.msg.Pose()  
    target_pose.position.x = x  
    target_pose.position.y = y  
    target_pose.position.z = z  
    target_pose.orientation.x = 0.5  
    target_pose.orientation.y = 0.5  
    target_pose.orientation.z = -0.5  
    target_pose.orientation.w = 0.5  
    mgpi.move_eef_to_pose(target_pose)
```

Figure 10 - The Quaternion of the Gripper

Question 3

- a. The following section outlines the workspace and configuration space of the 3-link planar arm robot.

Specifications

$$L_1 = 300mm$$

$$L_2 = 280mm$$

$$L_3 = 350mm$$

Assumptions

In order to simplify workspace and configuration space calculations we assume the following parameters:

$$0 < s_1 < 150 \text{ mm}$$

$$\frac{-\pi}{3} < \theta_2 < \frac{\pi}{3}$$

$$\frac{-2\pi}{3} < \theta_3 < \frac{2\pi}{3}$$

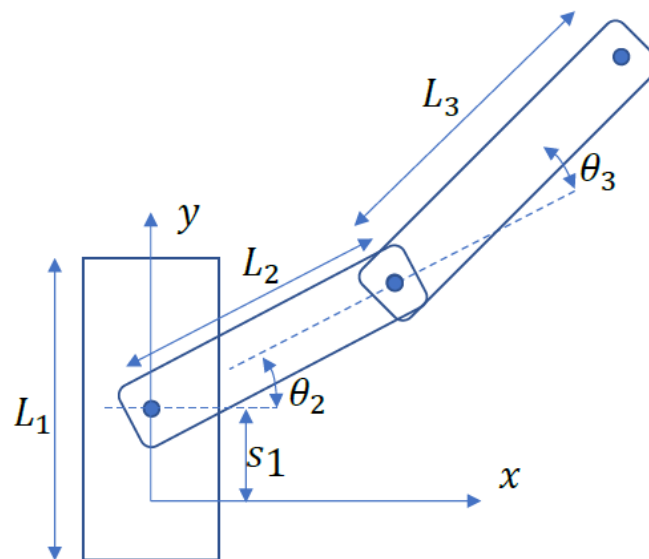


Figure 11 - Visual Representation (Top View)

Workspace/Configuration Space

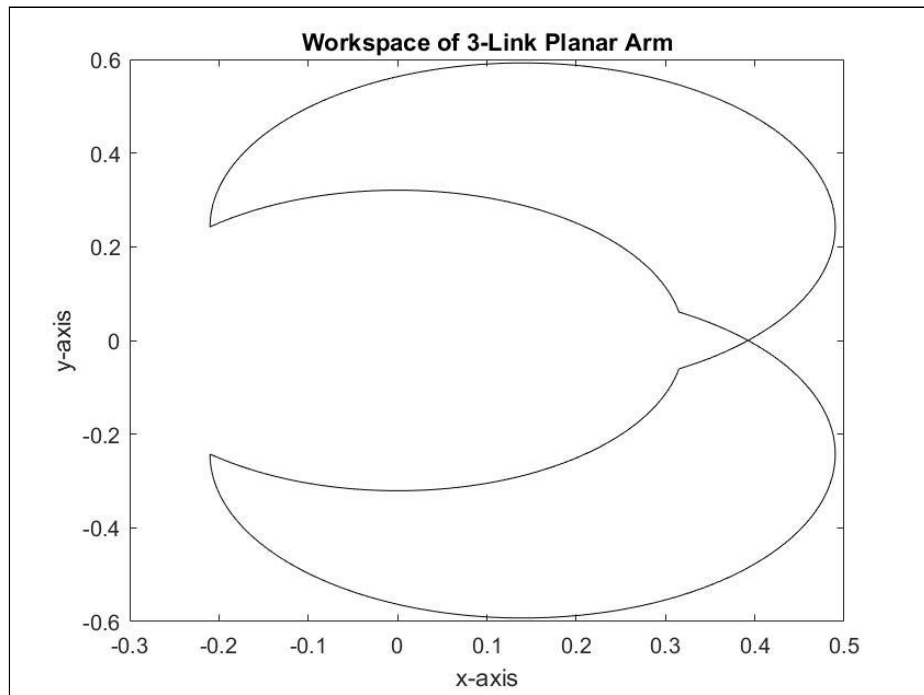


Figure 12 - Workspace - 3-Link Planar Arm

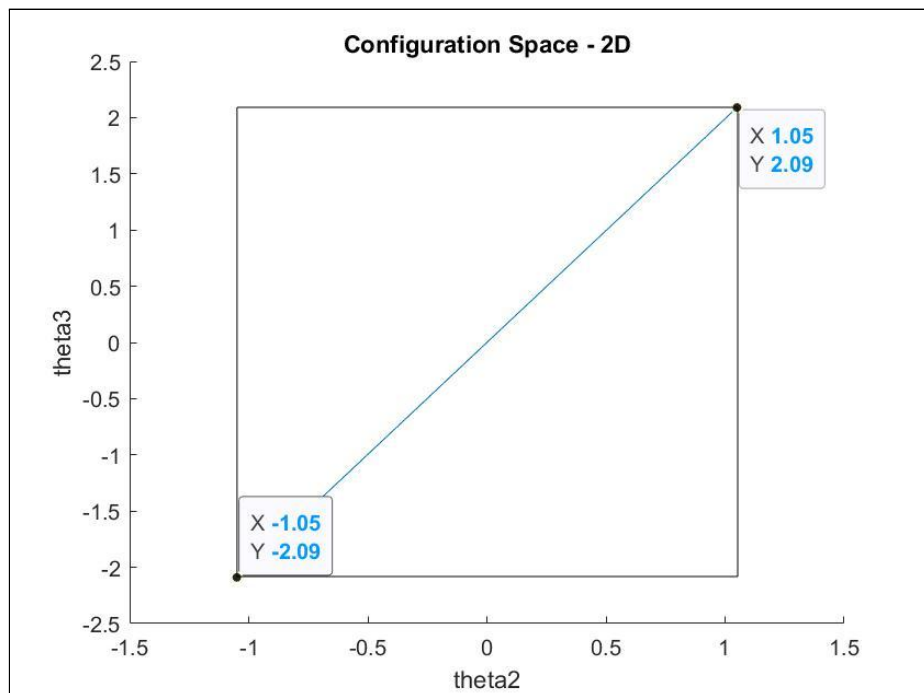


Figure 13 - Configuration Space - 3-Link Planar Arm

Forward Kinematics

- b. The following section outlines the forward kinematics of the manipulator utilizing the assumptions in the previous section *a* for values of the joint variables.

Link (<i>i</i>)	d_i (mm)	θ_i (deg)	a_i (mm)	α_i (deg)
1	0	0	100	0
2	0	θ_2	280	0
3	0	θ_3	350	0

Table 3 - Denavit-Hartenberg Parameters

$${}^{i-1}T_i \begin{bmatrix} \cos \theta & -\sin \theta & 0 & \alpha \\ \sin \theta * \cos \alpha & \cos \theta * \cos \alpha & -\sin \alpha & -d * \sin \alpha \\ \sin \theta * \sin \alpha & \cos \theta * \sin \alpha & \cos \alpha & \cos \alpha * d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\therefore {}^0T_6 = {}^0T_1 {}^1T_2 {}^2T_3$$

Equation 2 - Transformation Matrix

According to such information, the singularities of this robot occur at $\sin \theta_2$ and $\sin \theta_3$ equating to zero. This has been calculated utilizing the determination of the Jacobian matrix. The Jacobian matrix for this 3-link planar arm is outlined below in *Equation 3*.

There are two cases of singularity due to there being two joints of angular manipulation resulting in a full rank Jacobian matrix. This is displayed below:

$$\begin{bmatrix} \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = \frac{1}{L_2 L_3 s_3} \begin{bmatrix} L_3 c_{23} & L_3 s_{23} \\ -L_2 c_2 - L_3 c_{23} & -L_2 s_2 - L_3 s_{23} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$$

Equation 3 - Jacobian Matrix

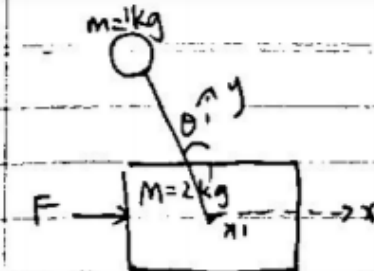
When the equating of θ_2 & θ_3 is either 0 or π , the Jacobian becomes non-invertible. Therefore, a singularity occurs in these two cases. The case of π occurs outside the specifications & workspace therefore, only the first case of both the angles equating to 0 is applicable.

There are an infinite number of ways for the angles to equate to zero, for example, with θ_2 & θ_3 equaling zero, or with the angles having opposite signs (hence, opposite directions of rotation equating to a zero value).

This singularity resembles an arm at full extension and therefore, cannot move radially. This is commonly known as an elbow singularity, causing the elbow joints to lock in position at full extension without possibility of radial movement in the cartesian space.

Question 4

a.



P = Pendulum C = Cart

$$V_p = V_c + V_p^c$$

$$= \frac{d}{dt} x_1 \hat{i} - \frac{d}{dt} (l \sin \theta \hat{i} + \frac{d}{dt} l \cos \theta \hat{j})$$

$$V_p = (\dot{x}_1 - (l \dot{\theta} \sin \theta))^2 + (l \dot{\theta} \cos \theta)^2$$

$$= \dot{x}_1^2 - 2 l \dot{x}_1 \dot{\theta} \cos \theta + l^2 \dot{\theta}^2$$

$$K = K_c + K_p$$

$$= \frac{1}{2} M V_c^2 + \frac{1}{2} m V_p^2 \quad (\text{sub. } M=2\text{kg} \text{ \& } m=1\text{kg})$$

$$= \dot{x}_1^2 + \frac{1}{2} \dot{x}_1^2 - 0.6 \dot{x}_1 \dot{\theta} \cos \theta + 0.18 \dot{\theta}^2$$

$$= 1.5 \dot{x}_1^2 - 0.6 \dot{x}_1 \dot{\theta} \cos \theta + 0.18 \dot{\theta}^2$$

$$P = P_c + P_p$$

$$= mgl \cos \theta$$

$$L = K - P = 1.5 \dot{x}_1^2 - 0.6 \dot{x}_1 \dot{\theta} \cos \theta + 0.18 \dot{\theta}^2 - 5.9 \cos \theta$$

$$\frac{\partial L}{\partial x_1} = 3 \dot{x}_1 - 0.6 \dot{\theta} \cos \theta$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}_1} = 3 \ddot{x}_1 - 0.6 \ddot{\theta} \cos \theta + 0.6 \dot{\theta}^2 \sin \theta$$

$$\frac{\partial L}{\partial x_1} = 0$$

$$F = 3 \ddot{x}_1 - 0.6 \ddot{\theta} \cos \theta + 0.6 \dot{\theta}^2 \sin \theta$$

$$\frac{\partial L}{\partial \theta} = -0.6 \dot{x}_1 \cos \theta + 0.36 \dot{\theta}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}} = 0.36 \ddot{\theta} - 0.6 \dot{x}_1 \sin \theta + 0.6 \dot{x}_1 \dot{\theta} \sin \theta$$

$$\frac{\partial L}{\partial \theta} = 0.6 \dot{x}_1 \sin \theta + 5.9 \sin \theta$$

$$T = 0.36 \ddot{\theta} - 0.6 \dot{x}_1 \cos \theta - 5.9 \sin \theta$$

$$= 0.6 \ddot{\theta} - \dot{x}_1 \cos \theta - 9.81 \sin \theta$$

Equation 4 - Calculations

- b. When simulating the system, the simulator can be used to simulate the cart-pole system behaviour under different external forces and torques. We can also design a PID controller that receives position feedback from the pendulum and varies the cart wheel speed in order to stabilise the pendulum's motion.

As shown in the results in part a), the summation of torque in the system is related to the pendulum angle, θ . By integrating the controller to the system, we can use the simulator to plot the change in the angle with respect to different values of external torque, and find out at what torque value the angle becomes too big and the pendulum tends to fall. With those data, we can select motors that have a torque value within the torque range acquired from the plottings, such that it ensures the pendulum doesn't fall.

Appendix

Q2 code

```
#!/usr/bin/env python
import math,rospy
import rospy
import geometry_msgs.msg
from gripper_control import GripperController
from move_group_interface import MoveGroupPythonInterface
from std_msgs.msg import Float32MultiArray
from gazebo_msgs.msg import ModelState
from gazebo_ros_link_attacher.srv import Attach, AttachRequest, AttachResponse

def print_information(mgpi):

    mgpi.print_useful_info() # Print useful information
    mgpi.print_robot_state() # Print the current state of the robot

def go_to_home_position(mgpi):

    # predefined state "home" is defined in
    universal_robot/ur5_e_moveit_config/config/ur5e.srdf
    mgpi.move_to_joint_state(mgpi.home_joint_angles)

def move_arm(mgpi,x,y,z):

    print("-----")
    print("move to:",x,y,z)

    target_pose = geometry_msgs.msg.Pose()
    target_pose.position.x = x
    target_pose.position.y = y
    target_pose.position.z = z
    target_pose.orientation.x = 0.5
    target_pose.orientation.y = 0.5
    target_pose.orientation.z = -0.5
    target_pose.orientation.w = 0.5
    mgpi.move_eef_to_pose(target_pose)

def Hold_block(mgpi,obj_name):

    mgpi.close_gripper(obj_name)

def Release_block(mgpi,obj_name):
```

```

mgpi.open_gripper(obj_name)

def print_position():

    rospy.sleep(1) #sleep for 1s

    position_sub = rospy.wait_for_message('/gazebo/model_states',
ModelStates,timeout=None) # Recive message one time

    # define the global variables
    global goal_x, goal_y, block_1_x, block_1_y, block_1_z,block_2_x, block_2_y,
block_2_z, block_3_x, block_3_y, block_3_z, block_4_x, block_4_y, block_4_z, block_5_x,
block_5_y, block_5_z

    # print out the position of block and goal
    print("The goal position in x-axis", position_sub.pose[2].position.x ,"The goal position
in y-axis:" ,position_sub.pose[2].position.y, "The goal position in z-axis:",
position_sub.pose[2].position.z)
    goal_x = position_sub.pose[2].position.x
    goal_y = position_sub.pose[2].position.y
    print("The block_1 position in x-axis", position_sub.pose[3].position.x ,"The block_1
position in y-axis:" ,position_sub.pose[3].position.y, "The block_1 position in z-axis:",
position_sub.pose[3].position.z)
    block_1_x = position_sub.pose[3].position.x
    block_1_y = position_sub.pose[3].position.y
    block_1_z = position_sub.pose[3].position.z
    print("The block_2 position in x-axis", position_sub.pose[4].position.x ,"The block_2
position in y-axis:" ,position_sub.pose[4].position.y, "The block_2 position in z-axis:",
position_sub.pose[4].position.z)
    block_2_x = position_sub.pose[4].position.x
    block_2_y = position_sub.pose[4].position.y
    block_2_z = position_sub.pose[4].position.z
    print("The block_3 position in x-axis", position_sub.pose[5].position.x ,"The block_3
position in y-axis:" ,position_sub.pose[5].position.y, "The block_3 position in z-axis:",
position_sub.pose[5].position.z)
    block_3_x = position_sub.pose[5].position.x
    block_3_y = position_sub.pose[5].position.y
    block_3_z = position_sub.pose[5].position.z
    print("The block_4 position in x-axis", position_sub.pose[6].position.x ,"The block_4
position in y-axis:" ,position_sub.pose[6].position.y, "The block_4 position in z-axis:",
position_sub.pose[6].position.z)
    block_4_x = position_sub.pose[6].position.x
    block_4_y = position_sub.pose[6].position.y
    block_4_z = position_sub.pose[6].position.z

```

```

    print("The block_5 position in x-axis", position_sub.pose[7].position.x, "The block_5
position in y-axis:", position_sub.pose[7].position.y, "The block_5 position in z-axis:",
position_sub.pose[7].position.z)

```

```

    block_5_x = position_sub.pose[7].position.x
    block_5_y = position_sub.pose[7].position.y
    block_5_z = position_sub.pose[7].position.z

```

```
def main():
```

```

    try:
        raw_input("Press Enter to build a tower")    # waits for Enter
        print(" 1. Creating and initializing the interface to the robot")
        print("-----")
        mgpi = MoveGroupPythonInterface()           # Create and Initialize the interface
        print("Simulation environment done.")
        raw_input("Press Enter to build a tower")    # waits for Enter
        print(" 2. Print Basic Information")
        print("-----")
        print_information(mgpi)
        print(" 3. Getting the block and goal position")
        print("-----")
        print_position()
        print("Location loading done.")
        go_to_home_position(mgpi)                    # start at home position

```

```
    # Start
```

```
    # First block-----1
```

```
    move_arm(mgpi, block_1_x, block_1_y, 0.4) #Go to the position of the block_1
```

```
    move_arm(mgpi, block_1_x, block_1_y, 0.081) # This height need to change to
```

```
hardware block height 0.061
```

```
    # Open gripper
```

```
    Hold_block(mgpi, "block_1")
```

```
    move_arm(mgpi, block_1_x, block_1_y, 0.4)
```

```
    move_arm(mgpi, goal_x, goal_y, 0.4) #Go to the Goal point
```

```
    move_arm(mgpi, goal_x, goal_y, 0.082) # This height need to change to
```

```
hardware block height 0.064
```

```
    Release_block(mgpi, "block_1") # close gripper
```

```
    move_arm(mgpi, goal_x, goal_y, 0.4)
```

```
    print_position() # update new loaction to minimise the position change by collision
```

```
    # Next block-----2
```

```
    move_arm(mgpi, block_2_x, 0.45, 0.4)
```

```
    move_arm(mgpi, block_2_x, block_2_y, 0.4)
```

```
    move_arm(mgpi, block_2_x, block_2_y, 0.081) # This height need to change to
```

```
hardware block height 0.061
```

```

# Open gripper
Hold_block(mgpi, "block_2")

move_arm(mgpi, block_2_x, block_2_y, 0.4)
move_arm(mgpi, goal_x, goal_y, 0.4)
move_arm(mgpi, goal_x, goal_y, 0.167)    # This height need to change to
hardware block height 0.125
Realease_block(mgpi, "block_2")          # close gripper

move_arm(mgpi, goal_x, goal_y, 0.4)

print_position() # update new loaction to minimise the position change by collision

# Next block-----3
move_arm(mgpi, block_3_x, 0.45, 0.4)
move_arm(mgpi, block_3_x, block_3_y, 0.4)
move_arm(mgpi, block_3_x, block_3_y, 0.081) # This height need to change to
hardware block height 0.061

# Open gripper
Hold_block(mgpi, "block_3")
move_arm(mgpi, block_3_x, block_3_y, 0.4)
move_arm(mgpi, goal_x, goal_y, 0.4)
move_arm(mgpi, goal_x, goal_y, 0.257)    # This height need to change to
hardware block height 0.188
Realease_block(mgpi, "block_3")          # close gripper

move_arm(mgpi, goal_x, goal_y, 0.4)

print_position() # update new loaction to minimise the position change by collision

# Next block-----4
move_arm(mgpi, block_4_x, 0.45, 0.4)
move_arm(mgpi, block_4_x, block_4_y, 0.4)
move_arm(mgpi, block_4_x, block_4_y, 0.081)    # This height need to change to
hardware block height 0.061
# Open gripper
Hold_block(mgpi, "block_4")

move_arm(mgpi, block_4_x, block_4_y, 0.4)
move_arm(mgpi, goal_x, goal_y, 0.4)
move_arm(mgpi, goal_x, goal_y, 0.345)    # This height need to change to
hardware block height 0.249
Realease_block(mgpi, "block_4")          # close gripper

move_arm(mgpi, goal_x, goal_y, 0.4)

```

```

print_position() # update new loaction to minimise the position change by collision

# Next block-----5
move_arm(mgpi, block_5_x, 0.45, 0.4)
move_arm(mgpi, block_5_x, block_5_y, 0.4)
move_arm(mgpi, block_5_x, block_5_y, 0.081)    # This height need to change to
hardware block height 0.061
# Open gripper
Hold_block(mgpi, "block_5")

move_arm(mgpi, block_5_x, block_5_y, 0.43)
move_arm(mgpi, goal_x, goal_y, 0.43)    # This height need to change to
hardware block height 0.31
Realease_block(mgpi, "block_5")    # close gripper

print_position() # Show the final location

print(" Tower finish !!")
print("-----")
except rospy.ROSInterruptException:
return
except KeyboardInterrupt:
return

if __name__ == '__main__':
    main()

```