

# Project Final Submission evaluation criteria

Course Code:	MCT344	Course Name:	Industrial Robotics	Project Final Submission	.....	Date:	.....				
Student Name:	Omar Makhous Eltantawy		Student ID:	19P1060							
	A (89-100)		B (76-88)		C (67-75)		D (60-66)	F (0-59)			
	100 96	92 89	88 84	80 76	75 72	69 67	66 64	62 60	59 40	40 20	0 0
Report (50%)	<ul style="list-style-type: none"> <li>Clear and relevant research methodology with complete implementation</li> <li>Excellent analysis of results and complete relevant conclusions</li> </ul>	<ul style="list-style-type: none"> <li>Clear and relevant research methodology missing few components</li> <li>Good analysis of results missing some minor conclusions</li> </ul>	<ul style="list-style-type: none"> <li>Clear research methodology missing several components some structure.</li> <li>Normal analysis of results missing some basic conclusions</li> </ul>	<ul style="list-style-type: none"> <li>Inappropriate research methodology Structure not clear.</li> <li>Incomplete analysis or results with some conclusions</li> </ul>	<ul style="list-style-type: none"> <li>Lack of clear research methodology</li> <li>Missing proper analysis or results and no conclusions at all.</li> </ul>						
	<input checked="" type="checkbox"/>										
Simulation Quality (50%)	<ul style="list-style-type: none"> <li>A perfectly running simulation with no errors,</li> <li>Express a deep understanding of each step to prepare the simulation,</li> <li>Applying the learned knowledge to prepare a perfectly running simulation with no errors.</li> </ul>	<ul style="list-style-type: none"> <li>A good running simulation,</li> <li>Express a very good understanding of each step to prepare the simulation,</li> <li>Applying most of the learned knowledge to prepare a good running simulation.</li> </ul>	<ul style="list-style-type: none"> <li>A moderate ability of preparing the simulation,</li> <li>Express a good understanding of the steps to prepare the simulation,</li> <li>Applying most of the learned knowledge, correctly, in the prepared solution.</li> </ul>	<ul style="list-style-type: none"> <li>A fair ability of preparing the simulation,</li> <li>Express a fair understanding of the steps to prepare the simulation,</li> <li>Applying some of the learned knowledge, correctly, in the proposed solution.</li> </ul>	<ul style="list-style-type: none"> <li>An inability of preparing the simulation,</li> <li>Failed to define the correct parameters of preparing the simulation,</li> <li>Failed to correctly apply the learned knowledge for proposing a valid simulation.</li> </ul>						
	<input checked="" type="checkbox"/>										
1 <sup>st</sup> marker Total	28	1 <sup>st</sup> marker Signature		ASU Agreed Mark	28						
2 <sup>nd</sup> Marker Total	28	2 <sup>nd</sup> marker Signature		UEL Agreed Mark	28						

Comments and Feedback:

UEL Grading System	Agreed Mark Range	ASU Grading Scale	
% equivalent at UEL		% at ASU	Grade
95% and higher		97% and higher	A+
82% to less than 95%	✓	93% to less than 97%	A
70% to less than 82%		89% to less than 93%	A-
66% to less than 70%		84% to less than 89%	B+
63% to less than 66%		80% to less than 84%	B
60% to less than 63%		76% to less than 80%	B-
56% to less than 60%		73% to less than 76%	C+
53% to less than 56%		70% to less than 73%	C
50% to less than 53%		67% to less than 70%	C-
45% to less than 50%		64% to less than 67%	D+
40% to less than 45%		60% to less than 64%	D
Less than 40%		Less than 60%	F

# MCT344- Industrial Robotics



TEAM 15

Supervised by:

Dr. Shady Ahmed Maged

Dr Diaa Emad

Eng. Mina Yousry

**Submitted by:**

Kirolos Thabet Fouad                    19P6754

Omar Mahrous Eltoutongy                19P1060

## Table of Contents

Introduction: .....	3
Screenshots of code: .....	3
DH parameters explaining: .....	5
Screenshot for the robot: .....	6
Screenshots for the outputs: .....	7
Screenshot for the published topic: .....	8
Video link: .....	9
References: .....	9

## Table of Figures:

Figure 1: Code part1.....	3
Figure 2: code Part 2 .....	4
Figure 3: DH Parameters tables.....	5
Figure 4: Robot with axis.....	5
Figure 5: the Initial Position of the robot .....	6
Figure 6: moved position of the robot .....	6
Figure 7: outputs with terminal .....	7
Figure 8: Topic lists .....	8

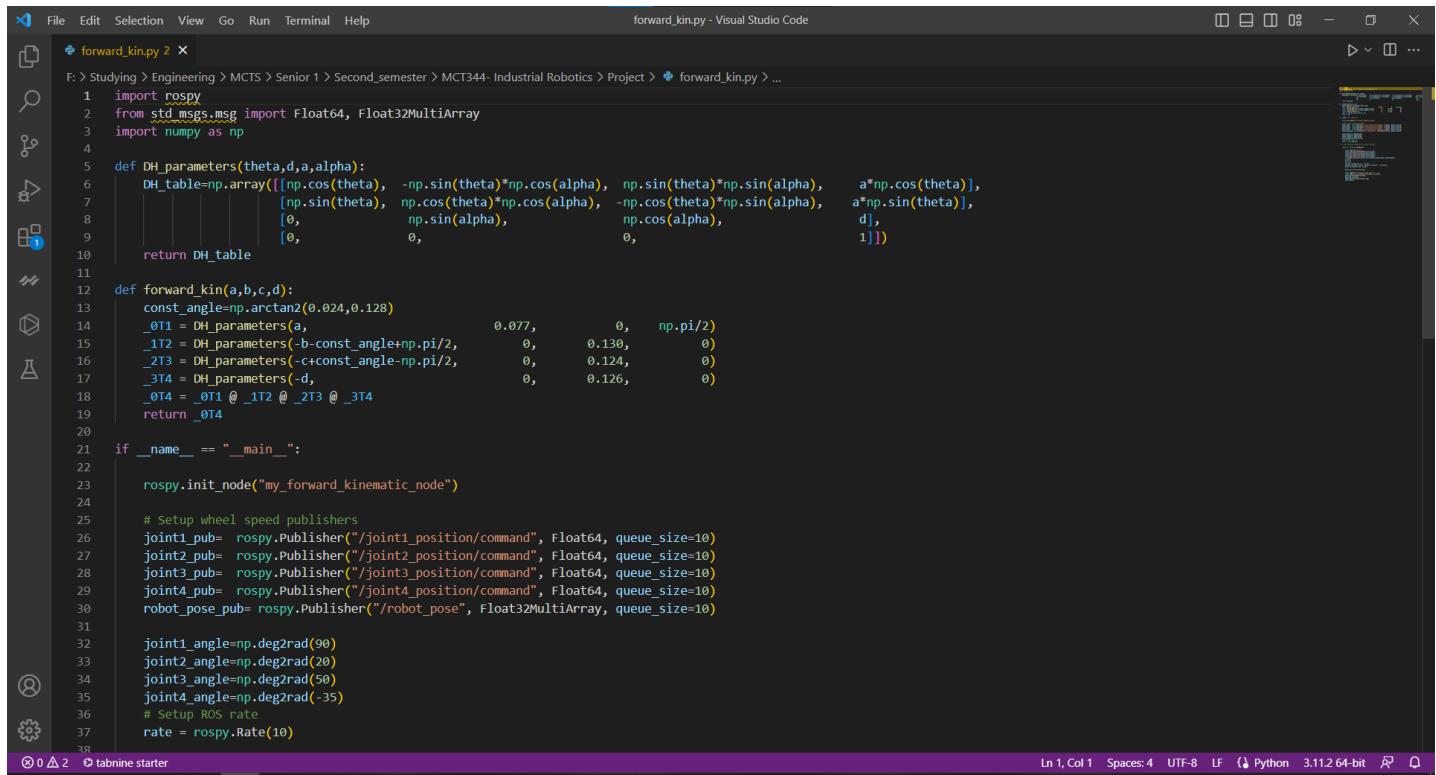
## Introduction:

In this documentation for this project milestone1, we will outline the steps required to program a robot to move according to a set of joint angles and publish its end effector position. This project milestone focuses on using ROS (Robot Operating System) to control a robotic arm and calculate the end effector position based on its joint angles.

To achieve this goal, we will first need to choose a set of four reasonable joint angles and move the robot using a ROS node that publishes on joint command topics. Then, we will calculate the DH (Denavit-Hartenberg) parameters and use them to determine the forward kinematics matrix to the base of the end effector. With this information, we can calculate the end effector position and publish it to a ROS topic in the form of a std\_msgs/Float32MultiArray.

Throughout this documentation, we will go through the steps required to complete this project milestone in detail, including the necessary background information and code examples. We assume that you have a basic understanding of ROS and programming concepts such as matrices and arrays. By following the steps in this documentation, you will gain a deeper understanding of how to control a robotic arm using ROS and calculate its end effector position based on its joint angles.

## Screenshots of code:



The screenshot shows a Visual Studio Code window with the file "forward\_kin.py" open. The code implements forward kinematics for a robotic arm using DH parameters. It includes imports for rospy, std\_msgs.msg, and numpy, defines DH parameters, calculates joint angles, and sets up ROS publishers for each joint and the robot's pose.

```
File Edit Selection View Go Run Terminal Help
F: > Studying > Engineering > MCTS > Senior 1 > Second.semester > MCT344- Industrial Robotics > Project > forward_kin.py ...
forward_kin.py - Visual Studio Code

1 import rospy
2 from std_msgs.msg import Float64, Float32MultiArray
3 import numpy as np
4
5 def DH_parameters(theta,d,a,alpha):
6     DH_table=np.array([[np.cos(theta), -np.sin(theta)*np.cos(alpha), np.sin(theta)*np.sin(alpha), a*np.cos(theta)],
7                        [0, np.sin(theta), np.cos(theta)*np.cos(alpha), -np.cos(theta)*np.sin(alpha)],
8                        [0, 0, np.sin(alpha), np.cos(alpha)],
9                        [0, 0, 0, 1]])
10    return DH_table
11
12 def forward_kin(a,b,c,d):
13     const_angle=np.arctan2(0.024,0.128)
14     _0T1 = DH_parameters(a, 0.077, 0, np.pi/2)
15     _1T2 = DH_parameters(b-const_angle+np.pi/2, 0, 0.130, 0)
16     _2T3 = DH_parameters(c+const_angle-np.pi/2, 0, 0.124, 0)
17     _3T4 = DH_parameters(-d, 0, 0.126, 0)
18     _0T4 = _0T1 @ _1T2 @ _2T3 @ _3T4
19     return _0T4
20
21 if __name__ == "__main__":
22
23     rospy.init_node("my_forward_kinematic_node")
24
25     # Setup wheel speed publishers
26     joint1_pub= rospy.Publisher("/joint1_position/command", Float64, queue_size=10)
27     joint2_pub= rospy.Publisher("/joint2_position/command", Float64, queue_size=10)
28     joint3_pub= rospy.Publisher("/joint3_position/command", Float64, queue_size=10)
29     joint4_pub= rospy.Publisher("/joint4_position/command", Float64, queue_size=10)
30     robot_pose_pub= rospy.Publisher('/robot_pose', Float32MultiArray, queue_size=10)
31
32     joint1_angle=np.deg2rad(90)
33     joint2_angle=np.deg2rad(20)
34     joint3_angle=np.deg2rad(50)
35     joint4_angle=np.deg2rad(-35)
36
37     # Setup ROS rate
38     rate = rospy.Rate(10)

Ln 1, Col 1  Spaces:4  UTF-8  LF  Python  3.11.2 64-bit  ⚙  ⌂
```

Figure 1: Code part1

```
File Edit Selection View Go Run Terminal Help
forward_kin.py - Visual Studio Code
F: > Studying > Engineering > MCTS > Senior 1 > Second_semester > MCT344- Industrial Robotics > Project > forward_kin.py > ...
34 joint3_angle=np.deg2rad(50)
35 joint4_angle=np.deg2rad(-35)
36 # Setup ROS rate
37 rate = rospy.Rate(10)
38
39 # rospy.loginfo("simulation has just started")
40
41 while not rospy.is_shutdown():
42
43     rospy.loginfo("Starting.....")
44     joint1_pub.publish(Float64(joint1_angle))
45     joint2_pub.publish(Float64(joint2_angle))
46     joint3_pub.publish(Float64(joint3_angle))
47     joint4_pub.publish(Float64(joint4_angle))
48     T =forward_kin(joint1_angle,joint2_angle,joint3_angle,joint4_angle)
49     x=T[0,3]
50     y=T[1,3]
51     z=T[2,3]
52     roll=np.arctan2(T[2,1], T[2,2])
53     pitch=np.arctan2(-T[2,0], np.sqrt(T[2,1]**2 + T[2,2]**2))
54     yaw=np.arctan2(T[1,0], T[0,0])
55
56     pose=[x,y,z,roll,pitch,yaw]
57
58     rospy.loginfo("calculated robot pose is: ")
59     rospy.loginfo(np.array(pose).astype(np.float16))
60     pose_msg=Float32MultiArray()
61     pose_msg.data=pose
62     robot_pose_pub.publish(pose_msg)
63     rate.sleep()
```

Ln 1, Col 1 | Spaces: 4 | UTF-8 | LF | Python | 3.11.2 64-bit | ⚙️ | ⌂

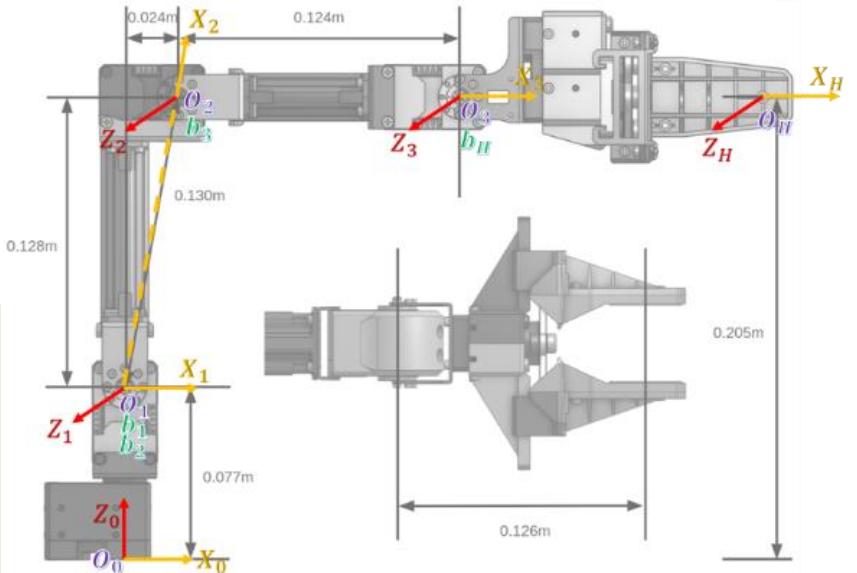
Figure 2: code Part 2

This Python script is designed to control a robotic arm's forward kinematics and publish the robot's position to ROS topics. The DH\_parameters() function calculates the DH transformation matrix based on the input values of theta, d, a, and alpha. The forward\_kin() function calculates the transformation matrix between the robot's base frame and end-effector frame based on the DH parameters and the joint angles of the four robot joints.

The script initializes ROS nodes and publishers for joint angle and robot pose data. The joint angles are initialized and published to ROS topics. The forward kinematics function is called to calculate the robot's position and orientation. The calculated position and orientation are published to a ROS topic as a Float32MultiArray message. The script runs continuously at a rate of 10 Hz using the rospy.Rate() function.

# DH parameters:

Joint	$\theta$	$d$	$a$	$\alpha$
1	$\theta_1$	0.077	0	90
2	$-\theta_2 + 90 - c$	0	0.130	0
3	$-\theta_3 - 90 + c$	0	0.124	0
4	$-\theta_4$	0	0.126	0



$$C = \tan^{-1}\left(\frac{0.024}{0.128}\right) \approx 11$$



Figure 3: DH Parameters tables

## DH parameters explaining:

This is the [link](#) to the video explaining the DH parameters in depth. Please download it to listen to the explanation.

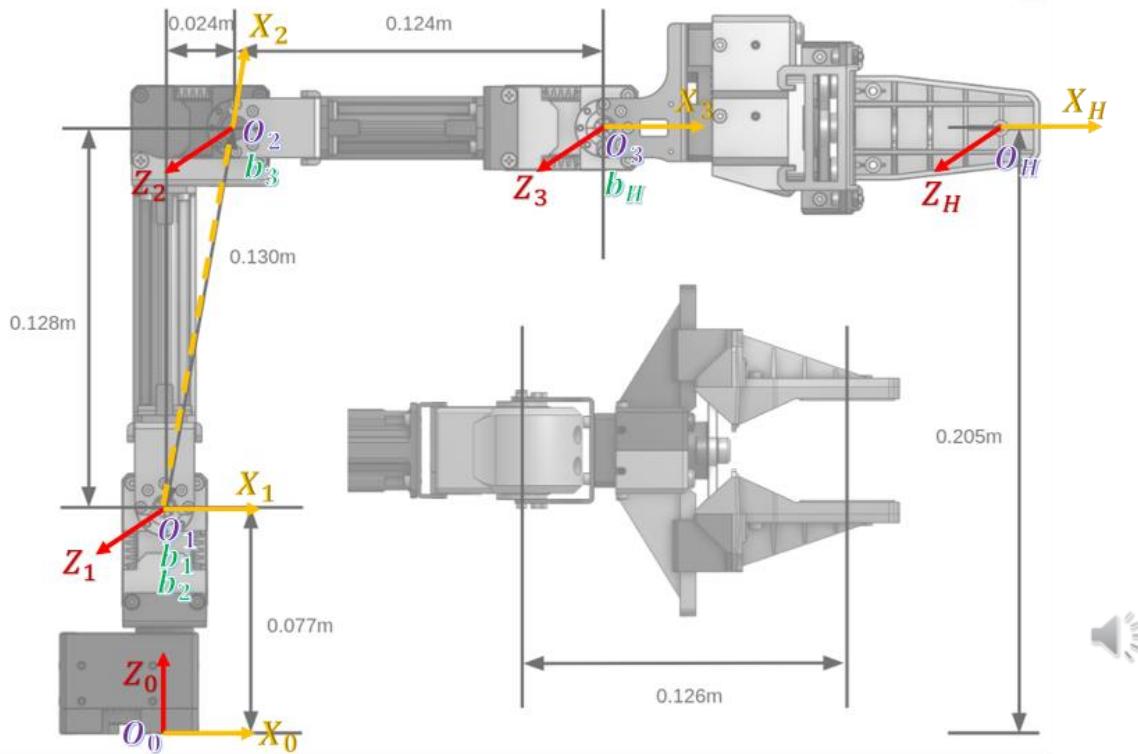


Figure 4: Robot with axis

## Screenshot for the robot:

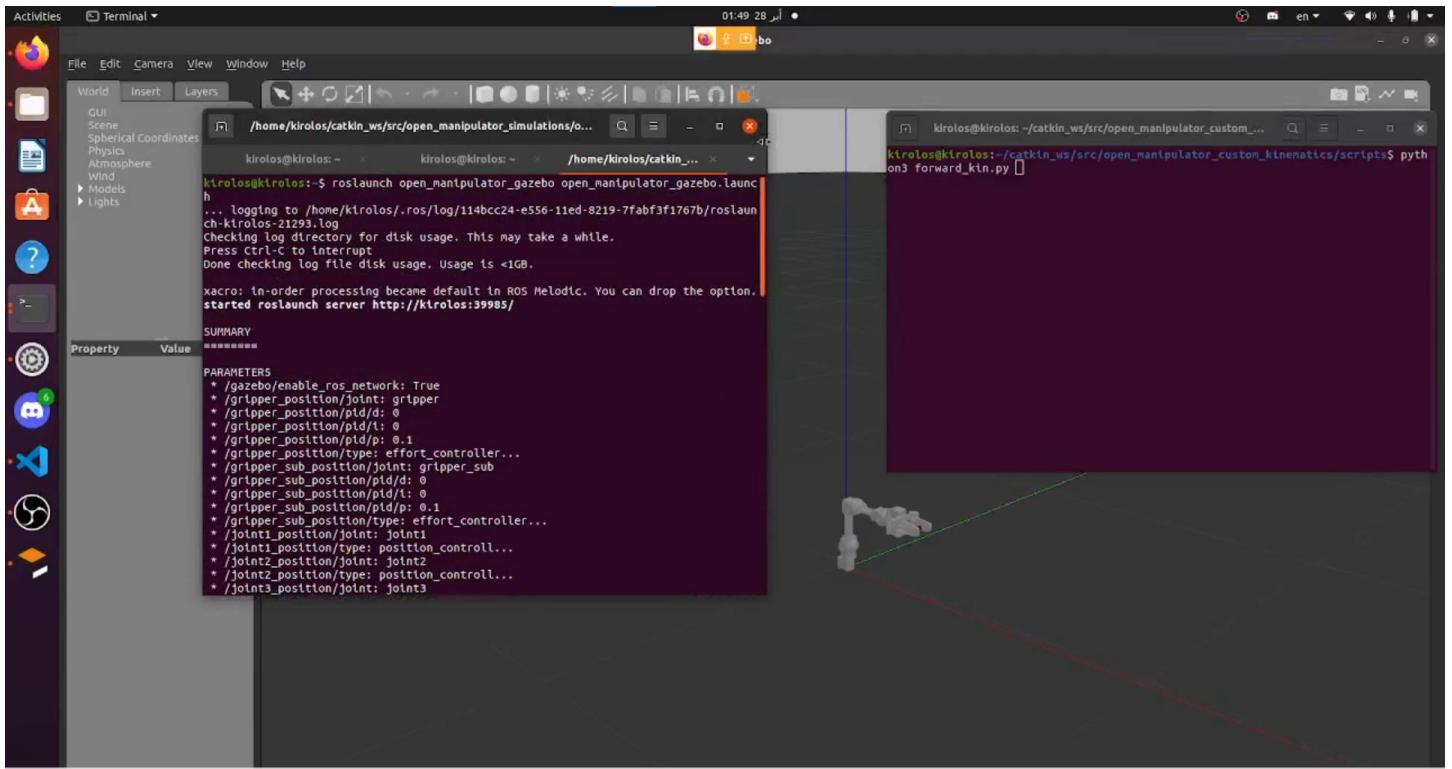


Figure 5: the Initial Position of the robot

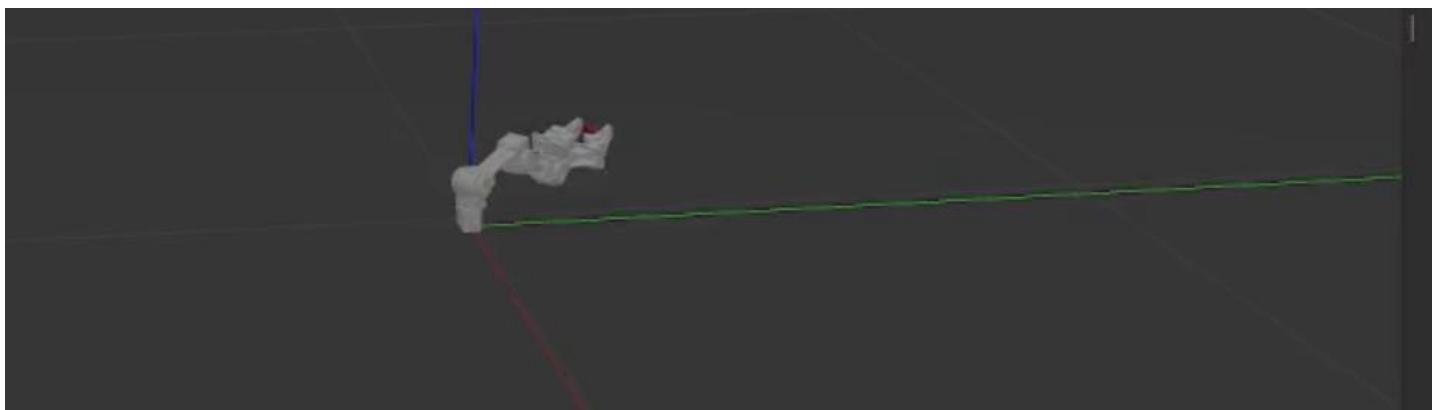


Figure 6: moved position of the robot.

Screenshots for the outputs:

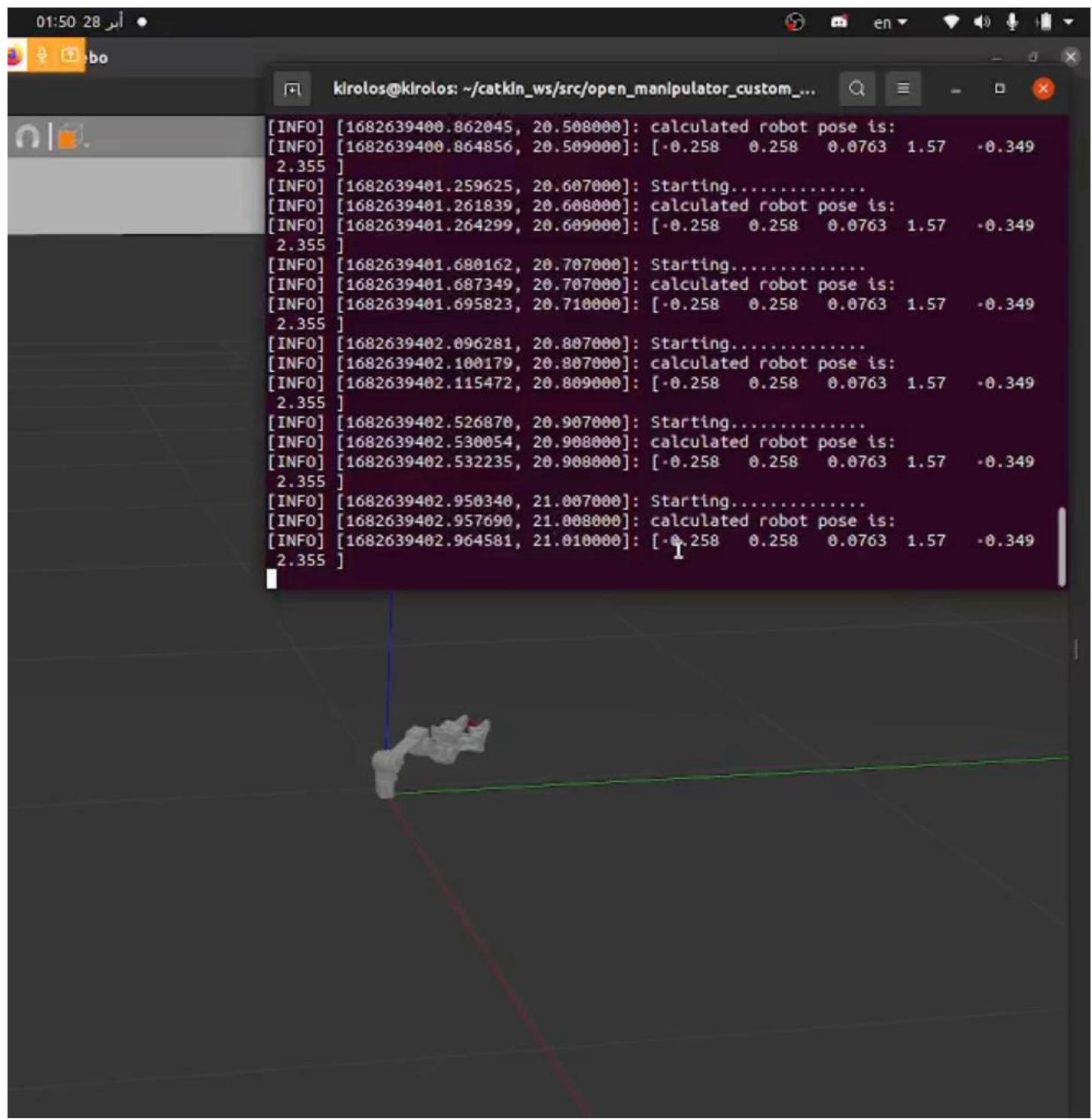
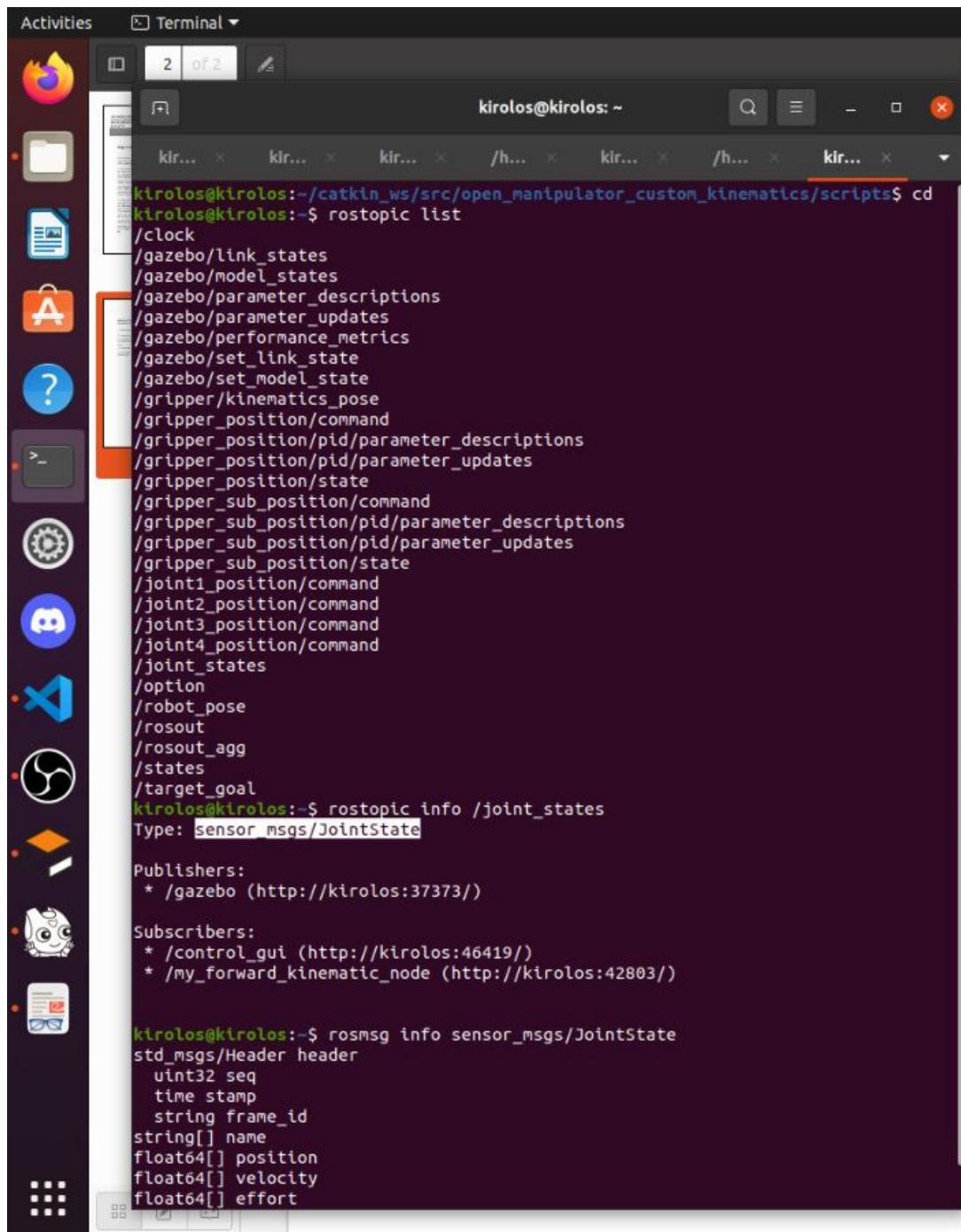


Figure 7: outputs with terminal

Screenshot for the published topic:



The screenshot shows a terminal window titled "Terminal" in the Activities overview. The terminal is running on a system named "kirolos". The user has run several commands to list and inspect ROS topics:

```
kirolos@kirolos:~/catkin_ws/src/open_manipulator_custom_kinematics/scripts$ cd
kirolos@kirolos:~$ rostopic list
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/gripper/kinematics_pose
/gripper_position/command
/gripper_position/pid/parameter_descriptions
/gripper_position/pid/parameter_updates
/gripper_position/state
/gripper_sub_position/command
/gripper_sub_position/pid/parameter_descriptions
/gripper_sub_position/pid/parameter_updates
/gripper_sub_position/state
/joint1_position/command
/joint2_position/command
/joint3_position/command
/joint4_position/command
/joint_states
/option
/robot_pose
/rosout
/rosout_agg
/states
/target_goal
kirolos@kirolos:~$ rostopic info /joint_states
Type: sensor_msgs/JointState

Publishers:
* /gazebo (http://kirolos:37373/)

Subscribers:
* /control_gui (http://kirolos:46419/)
* /my_forward_kinematic_node (http://kirolos:42803/)

kirolos@kirolos:~$ rosmsg info sensor_msgs/JointState
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string[] name
float64[] position
float64[] velocity
float64[] effort
```

Figure 8: Topic lists

## Video link:

In this [link](#), you can watch our video.

And in this [link](#) the drive folder containing all data.

## References:

[OpenMANIPULATOR-X \(robotis.com\)](#)

# MCT344- Industrial Robotics



TEAM 15

Supervised by:

Dr. Shady Ahmed Maged

Dr Diaa Emad

Eng. Mina Yousry

**Submitted by:**

Kirolos Thabet Fouad                    19P6754

Omar Mahrous Eltoutongy                19P1060

## Table of Contents

Introduction: .....	3
Screenshots of code: .....	3
1. Position node: .....	3
2. fkine node: .....	4
3. ikine node:.....	5
Screenshots of the outputs:.....	9
RQT Graph:.....	11
Links: In this link, you will find our video for this milestone.....	11
References:.....	11

## Table of Figures:

Figure 1: position node .....	3
Figure 2: fkine node part2.....	4
Figure 3: fkine node part1.....	4
Figure 4: fkine node part3.....	5
Figure 5: Joint_states details.....	5
Figure 6: ikine node part 2 .....	6
Figure 7: ikine node part 1 .....	6
Figure 9: geometrical explain p1.....	7
Figure 8: geometrical explain p2 .....	7
Figure 10: geometrical explain p3 .....	8
Figure 11: Rostopic list .....	9
Figure 12: topic echoing robot pose .....	9
Figure 13: robot after published the required position .....	10
Figure 14:reading the position to compare with the target position node. ....	10
Figure 15: sending the calculated angles from the ikine node. ....	10
Figure 16: RQT graph.....	11

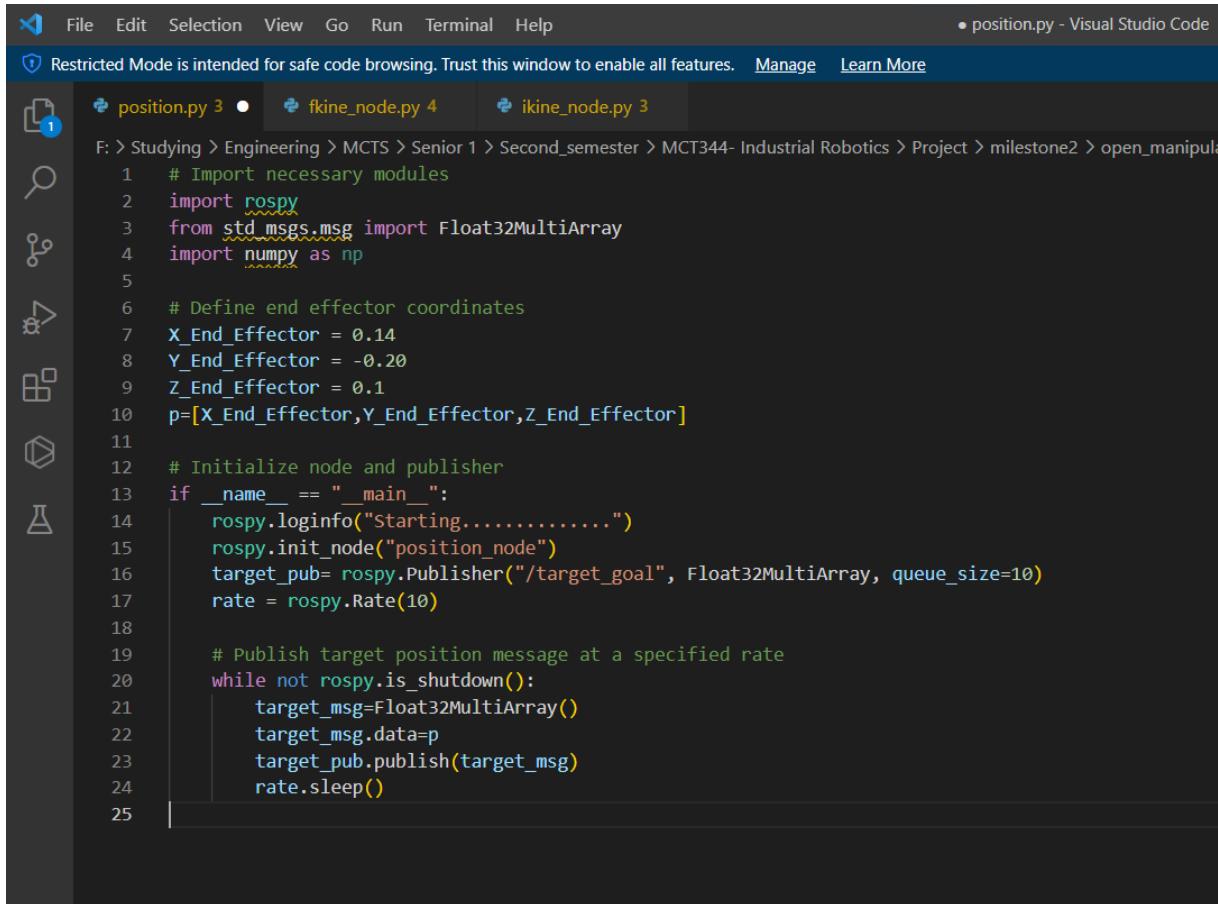
## Introduction:

Project Milestone 2 is a robotics project where each team is required to submit a working package named **open\_manipulator\_custom\_kinematics**. The package consists of two nodes, **fkine\_node** and **ikine\_node**, that perform forward kinematics and inverse kinematics calculations, respectively, for a robot. The project involves using DH parameters and geometric methods to calculate joint angles and transform matrices and publishing them on specific topics in Gazebo. The robot can be controlled using a GUI controller to verify the accuracy of the calculations.

## Screenshots of code:

### 1. Position node:

This node is responsible for holding the target goal values and publish it to a topic named `target_goal` which the ikine node subscribes to it.



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** position.py - Visual Studio Code
- Status Bar:** Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
- Editor Area:** The code for `position.py` is displayed. The code initializes a ROS node to publish target positions at a rate of 10 Hz. It defines end-effector coordinates and publishes them to a topic named `/target_goal`.

```
1 # Import necessary modules
2 import rospy
3 from std_msgs.msg import Float32MultiArray
4 import numpy as np
5
6 # Define end effector coordinates
7 X_End_Effector = 0.14
8 Y_End_Effector = -0.20
9 Z_End_Effector = 0.1
10 p=[X_End_Effector,Y_End_Effector,Z_End_Effector]
11
12 # Initialize node and publisher
13 if __name__ == "__main__":
14     rospy.loginfo("starting.....")
15     rospy.init_node("position_node")
16     target_pub= rospy.Publisher("/target_goal", Float32MultiArray, queue_size=10)
17     rate = rospy.Rate(10)
18
19     # Publish target position message at a specified rate
20     while not rospy.is_shutdown():
21         target_msg=Float32MultiArray()
22         target_msg.data=p
23         target_pub.publish(target_msg)
24         rate.sleep()
25
```

Figure 1: position node

## 2. fkine node:

this node read the angle values from the gazebo topic Jointstates and do the regular algorithm to calculate the position of end effector in Euler form.

```

File Edit Selection View Go Run Terminal Help
fkine_node.py - Visual Studio Code

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

fkine_node.py 4 ✘ ikins_node.py 3

F: > Studying > Engineering > MCTS > Senior 1 > Second_semester > MCT344- Industrial Robotics > Project > milestone2 > open_manipulator_custom_kinematics > scripts > fkine_node.py > ...

1 import rospy
2 from std_msgs.msg import Float64, Float32MultiArray
3 from sensor_msgs.msg import JointState
4 import numpy as np
5
6 # Initialize the joint angles as 0
7 joint1_angle=0
8 joint2_angle=0
9 joint3_angle=0
10 joint4_angle=0
11
12 # Function to compute the DH parameters for a given joint
13 def DH_parameters(theta,d,a,alpha):
14     # Create the DH parameter matrix
15     DH_table=np.array([[np.cos(theta), -np.sin(theta)*np.cos(alpha), np.sin(theta)*np.sin(alpha), a*np.cos(theta)],
16                         [np.sin(theta), np.cos(theta)*np.cos(alpha), -np.cos(theta)*np.sin(alpha), a*np.sin(theta)],
17                         [0, np.sin(alpha), np.cos(alpha), d],
18                         [0, 0, 0, 1]])
19     return DH_table
20
21 # Function to compute the forward kinematics
22 def forward_kin(a,b,c,d):
23     # Compute the constant angle (angle between two links in the robot arm)
24     const_angle=np.arctan2(0.024,0.128)
25
26     # Compute the DH parameter matrices for each joint
27     _T1 = DH_parameters(a, 0.077, 0, np.pi/2)
28     _T2 = DH_parameters(-b,const_angle+np.pi/2, 0, 0.130, 0)
29     _T3 = DH_parameters(-c,const_angle-np.pi/2, 0, 0.135, 0)
30     _T4 = DH_parameters(-d, 0, 0.126, 0)
31
32     # Compute the transformation matrix from the base to the end effector
33     _Tf4 = _T1 @ _T2 @ _T3 @ _T4
34
35
36     return _Tf4

```

Figure 3: fkine node part1

```

File Edit Selection View Go Run Terminal Help
fkine_node.py - Visual Studio Code

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

fkine_node.py 4 ✘ ikins_node.py 3

F: > Studying > Engineering > MCTS > Senior 1 > Second_semester > MCT344- Industrial Robotics > Project > milestone2 > open_manipulator_custom_kinematics > scripts > fkine_node.py > ...

34     return _Tf4
35
36 # Callback function to update the joint angles
37 def callback(msg):
38     # Declare the joint angles as global so that they can be accessed outside the function
39     global joint1_angle
40     global joint2_angle
41     global joint3_angle
42     global joint4_angle
43
44     # Update the joint angles based on the message from the joint_states topic
45     joint1_angle = msg.position[2]
46     joint2_angle = msg.position[3]
47     joint3_angle = msg.position[4]
48     joint4_angle = msg.position[5]
49
50 if __name__ == "__main__":
51
52     rospy.init_node("my_forward_kinematic_node")
53     rospy.Subscriber("joint_states", JointState, callback)
54     robot_pose_pub= rospy.Publisher("/robot_pose", Float32MultiArray, queue_size=10)
55
56     # Setup ROS rate
57     rate = rospy.Rate(10)
58
59     while not rospy.is_shutdown():
60
61         rospy.loginfo("Starting.....")
62
63         # Call forward kinematics function and calculate pose
64         T = forward_kin(joint1_angle, joint2_angle, joint3_angle, joint4_angle)
65         x = T[0,3]
66         y = T[1,3]
67         z = T[2,3]
68         roll = np.arctan2(T[2,1], T[2,2])
69         pitch = np.arctan2(-T[2,0], np.sqrt(T[2,1]**2 + T[2,2]**2))
70         yaw = np.arctan2(T[1,0], T[0,0])

    Restricted Mode 0 ▲ 7 tabnine starter Ln 82, Col 1

```

Figure 2: fkine node part2

```

61     rospy.loginfo("Starting.....")
62
63     # Call forward kinematics function and calculate pose
64     T = forward_kin(joint1_angle, joint2_angle, joint3_angle, joint4_angle)
65     x = T[0,3]
66     y = T[1,3]
67     z = T[2,3]
68     roll = np.arctan2(T[2,1], T[2,2])
69     pitch = np.arctan2(-T[2,0], np.sqrt(T[2,1]**2 + T[2,2]**2))
70     yaw = np.arctan2(T[1,0], T[0,0])
71
72     pose = [x, y, z, roll, pitch, yaw]
73
74     rospy.loginfo("calculated robot pose is:  ")
75     rospy.loginfo(np.array(pose).astype(np.float16))
76
77     # Publish calculated pose
78     pose_msg = Float32MultiArray()
79     pose_msg.data = pose
80     robot_pose_pub.publish(pose_msg)
81     rate.sleep()
82

```

Figure 4: fkine node part3

To work with the joint states we had to know its details as following:

```

kirolos@kirolos:~$ rostopic info /joint_states
Type: sensor_msgs/JointState

Publishers:
* /gazebo (http://kirolos:37373/)

Subscribers:
* /control_gui (http://kirolos:46419/)
* /my_forward_kinematic_node (http://kirolos:42803/)

kirolos@kirolos:~$ rosmsg info sensor_msgs/JointState
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string[] name
float64[] position
float64[] velocity

```

Figure 5: Joint\_states details

### 3. ikine node:

this nodes subscribes to only one topic which is the target\_goal and publishes to the 4 jointstate/command topic which are connected to gazebo to move the robot, its accuracy may be checked using the gui and we found that the maximum error in our calculations is about 0.02 , we can guess the problem which is the shifting in the axis in gazebo (small but considerable).

```

File Edit Selection View Go Run Terminal Help
ikine_node.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
ikine_node.py 3 ×
F: > Studying > Engineering > MCTS > Senior 1 > Second_semester > MCT344- Industrial Robotics > Project > milestone2 > open_manipulator_custom_kinematics >
1 import rospy
2 from std_msgs.msg import Float64,Float32MultiArray
3 import numpy as np
4
5 # Initialize the end effector coordinates
6 X_End_Effector=0
7 Y_End_Effector=0
8 Z_End_Effector=0
9
10 # Callback function to update the end effector coordinates
11 def callback(msg):
12     global X_End_Effector, Y_End_Effector, Z_End_Effector
13     # Extract the end effector coordinates from the message
14     X_End_Effector = msg.data[0]
15     Y_End_Effector = msg.data[1]
16     Z_End_Effector = msg.data[2]
17
18     # Define the link lengths
19     d1 = 0.077
20     d2 = 0.13
21     d3 = 0.25 #we will treat the link three and four as one rigid link
22
23     # Define a constant angle to simplify the inverse kinematics calculation
24     const_angle=np.arctan2(0.024,0.128)#10.619
25
26     # Calculate the joint angles using inverse kinematics
27     joint1_angle=np.arctan2(Y_End_Effector,X_End_Effector)
28
29     r = np.sqrt(X_End_Effector**2 + Y_End_Effector**2)
30
31     R = np.sqrt(r**2 + (Z_End_Effector-d1)**2)
32     theta3 = np.arccos((R**2 - d2**2 - d3**2)/(2*d2*d3))
33     #in quadrant 1,4 it will be positive else it will be negative and i need to remove the sign
34     #to get theta 2
35     alfa = np.arctan2((Z_End_Effector-d1),r)
36     gamma = np.arcsin(d3*np.sin(theta3)/R)
37
38     if(0<theta3<np.pi/2) or (3/2*np.pi<theta3<2*np.pi):
39         theta2 = alfa - gamma
40     else:
41         theta2 = alfa + gamma
42         theta3=-theta3
43
44     joint2_angle = -theta2 -const_angle +np.pi/2
45     joint3_angle = -theta3 + const_angle -np.pi/2
46
47     # Publish the joint angles to the robot
48     joint1_pub.publish(Float64(joint1_angle))
49     joint2_pub.publish(Float64(joint2_angle))
50     joint3_pub.publish(Float64(joint3_angle))
51
52     # Log the joint angles for debugging purposes
53     rospy.loginfo(joint1_angle)
54     rospy.loginfo(joint2_angle)
55     rospy.loginfo(joint3_angle)
56
57 if __name__ == "__main__":
58     # Initialize the node
59     rospy.init_node("my_inverse_kinematics_node")
60     # Subscribe to the /target_goal topic to receive end effector coordinates
61     rospy.Subscriber("/target_goal", Float32MultiArray, callback)
62     # Initialize publishers to send joint angles to the robot
63     joint1_pub= rospy.Publisher("//joint1_position/command", Float64, queue_size=10)
64     joint2_pub= rospy.Publisher("//joint2_position/command", Float64, queue_size=10)
65     joint3_pub= rospy.Publisher("//joint3_position/command", Float64, queue_size=10)
66
67     # Set the publishing rate
68     rate = rospy.Rate(10)
69     # Start the node
70     rospy.spin()

```

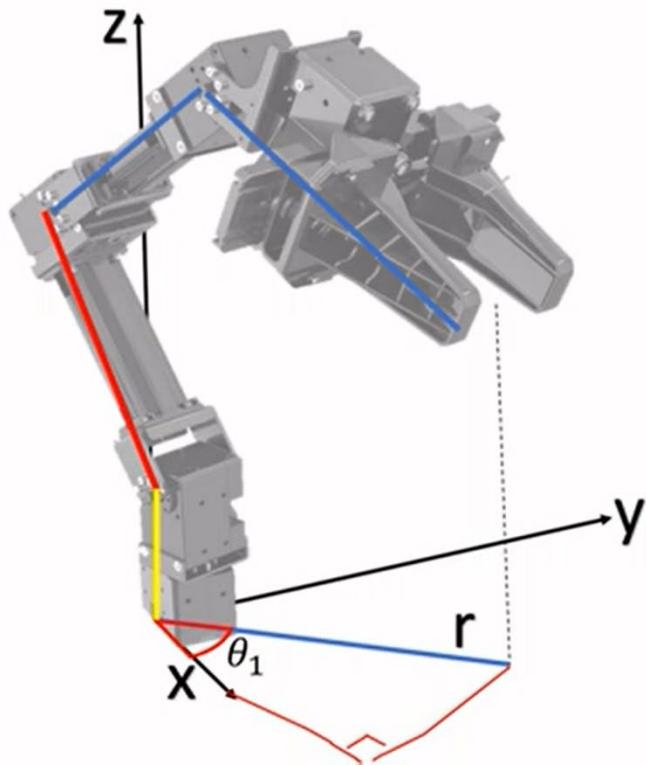
Figure 7: ikine node part 1

```

File Edit Selection View Go Run Terminal Help
ikine_node.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
ikine_node.py 3 ×
F: > Studying > Engineering > MCTS > Senior 1 > Second_semester > MCT344- Industrial Robotics > Project > milestone2 > open_manipulator_custom_kinematics >
36 gamma = np.arcsin(d3*np.sin(theta3)/R)
37 if(0<theta3<np.pi/2) or (3/2*np.pi<theta3<2*np.pi):
38     theta2 = alfa - gamma
39 else:
40     theta2 = alfa + gamma
41     theta3=-theta3
42
43 joint2_angle = -theta2 -const_angle +np.pi/2
44 joint3_angle = -theta3 + const_angle -np.pi/2
45
46 # Publish the joint angles to the robot
47 joint1_pub.publish(Float64(joint1_angle))
48 joint2_pub.publish(Float64(joint2_angle))
49 joint3_pub.publish(Float64(joint3_angle))
50
51 # Log the joint angles for debugging purposes
52 rospy.loginfo(joint1_angle)
53 rospy.loginfo(joint2_angle)
54 rospy.loginfo(joint3_angle)
55
56
57 if __name__ == "__main__":
58     # Initialize the node
59     rospy.init_node("my_inverse_kinematics_node")
60     # Subscribe to the /target_goal topic to receive end effector coordinates
61     rospy.Subscriber("/target_goal", Float32MultiArray, callback)
62     # Initialize publishers to send joint angles to the robot
63     joint1_pub= rospy.Publisher("//joint1_position/command", Float64, queue_size=10)
64     joint2_pub= rospy.Publisher("//joint2_position/command", Float64, queue_size=10)
65     joint3_pub= rospy.Publisher("//joint3_position/command", Float64, queue_size=10)
66
67     # Set the publishing rate
68     rate = rospy.Rate(10)
69     # Start the node
70     rospy.spin()

```

Figure 6: ikine node part 2



$$\theta_1 = \tan^{-1} \frac{y_{end\ effector}}{x_{end\ effector}}$$

Figure 9: geometrical explain p1

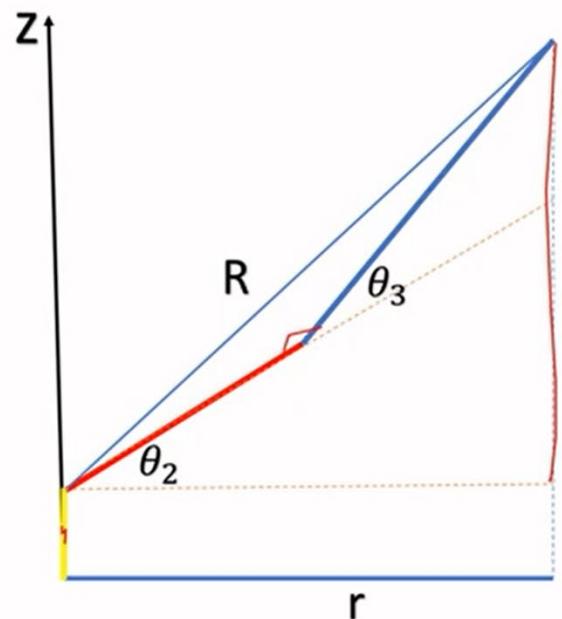
Applying cos rule to get  $\theta_3$ :

$$R^2 = r^2 + (Z_{end\ effector} - d_1)^2$$

$$r^2 = X_{end\ effector}^2 + Y_{end\ effector}^2$$

$$R^2 = d_2^2 + d_3^2 - 2d_2d_3 * \cos(180 - \theta_3)$$

$$\theta_3 = \cos^{-1} \frac{R^2 - d_2^2 - d_3^2}{2 * d_2 * d_3}$$



Note: d represent the link length

Figure 8: geometrical explain p2

Applying sin rule to get  $\theta_2$ :

$$\alpha = \tan^{-1} \frac{(Z_{end\ effector} - d_1)}{r}$$

$$\gamma = \sin^{-1} \frac{d_3 \sin(\theta_3)}{R}$$

$$\therefore \theta_3 = \alpha - \gamma$$

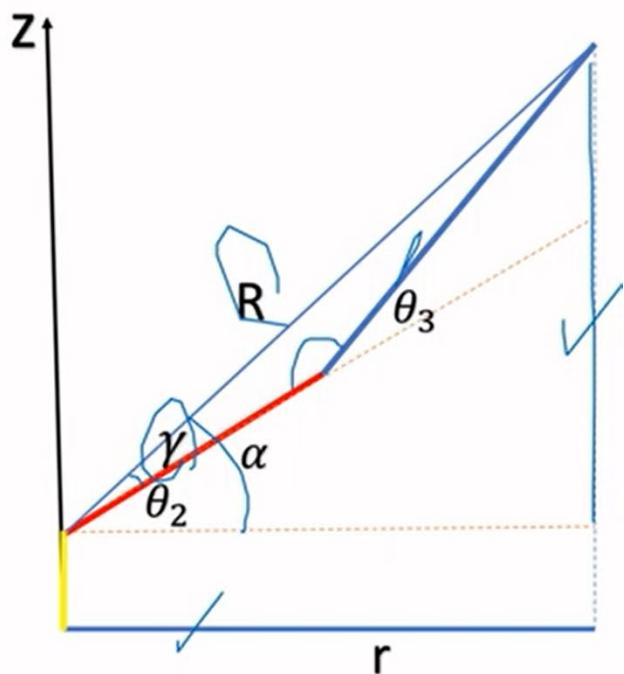


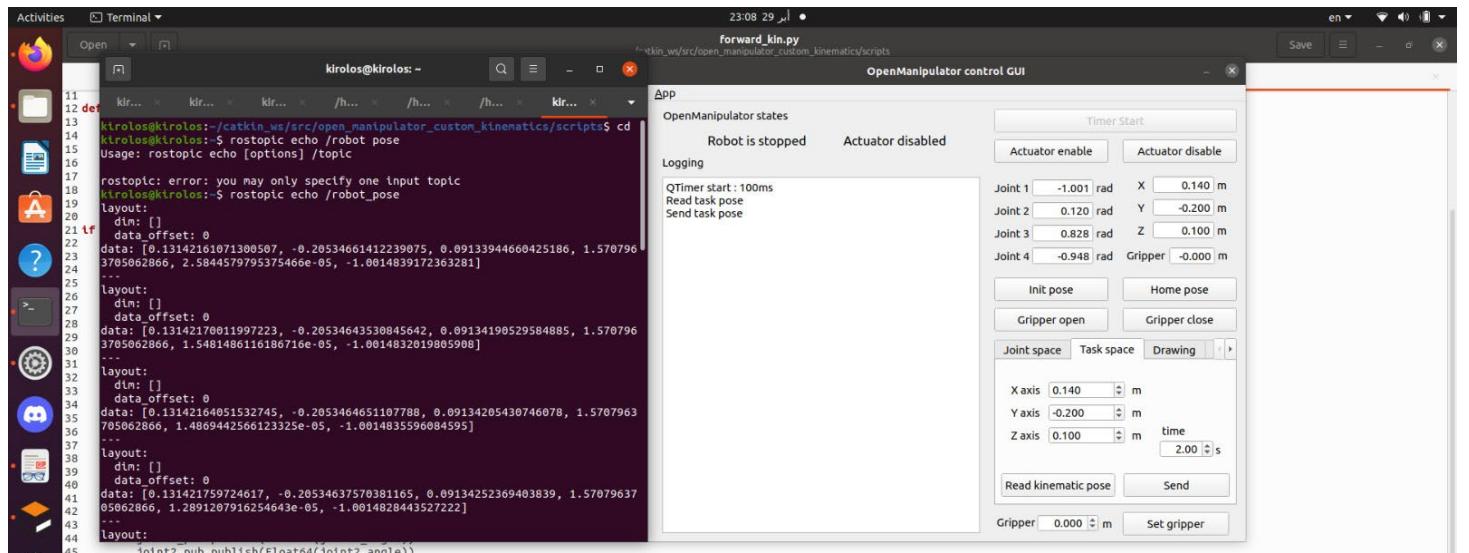
Figure 10: geometrical explain p3

## Screenshots of the outputs:

```
Terminal ▾ 2 of 2 Terminal ▾ kirolos@kirolos: ~ Q ⌂ - x kirolos@kirolos:~/catkin_ws/src/open_manipulator_custom_kinematics/scripts$ cd kirolos@kirolos:~$ rostopic list /clock /gazebo/link_states /gazebo/model_states /gazebo/parameter_descriptions /gazebo/parameter_updates /gazebo/performance_metrics /gazebo/set_link_state /gazebo/set_model_state /gripper/kinematics_pose /gripper_position/command /gripper_position/pid/parameter_descriptions /gripper_position/pid/parameter_updates /gripper_position/state /gripper_sub_position/command /gripper_sub_position/pid/parameter_descriptions /gripper_sub_position/pid/parameter_updates /gripper_sub_position/state /joint1_position/command /joint2_position/command /joint3_position/command /joint4_position/command /joint_states /option /robot_pose /rosout /rosout_agg /states /target_goal
```

*Figure 11: Rostopic list*

a. screenshots of topic echoing of calculated pose after running forward kinematics nodes



*Figure 12: topic echoing robot pose*

b. screenshots for the robot moved inside gazebo after running your inverse kinematics node.

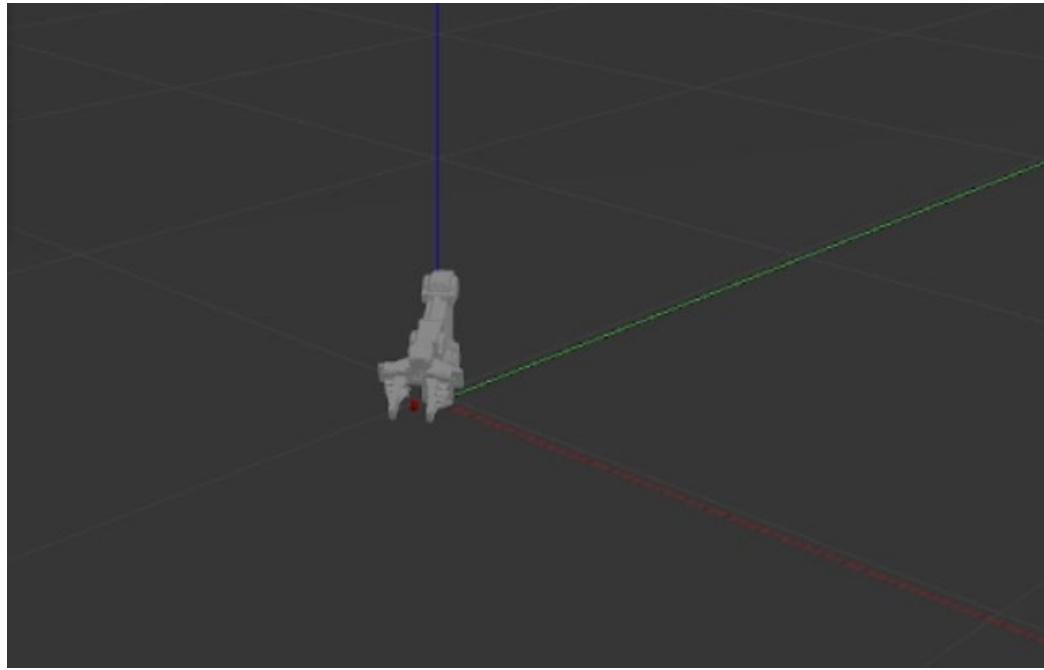


Figure 13: robot after published the required position

c. screenshots of moving the robot using GUI controller and verifying your calculations

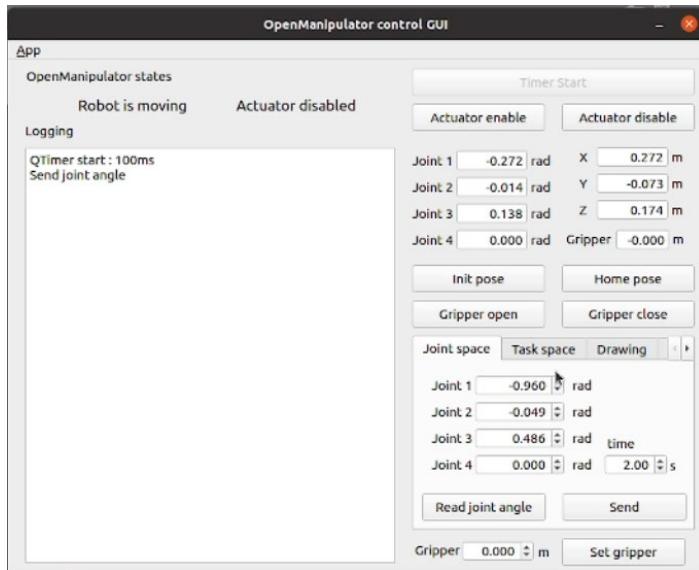


Figure 15: sending the calculated angles from the ikine node.

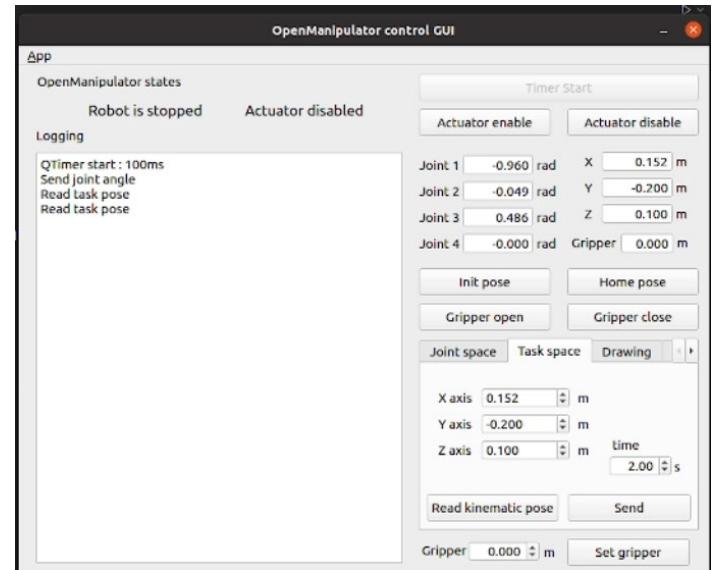


Figure 14: reading the position to compare with the target position node.

## RQT Graph:

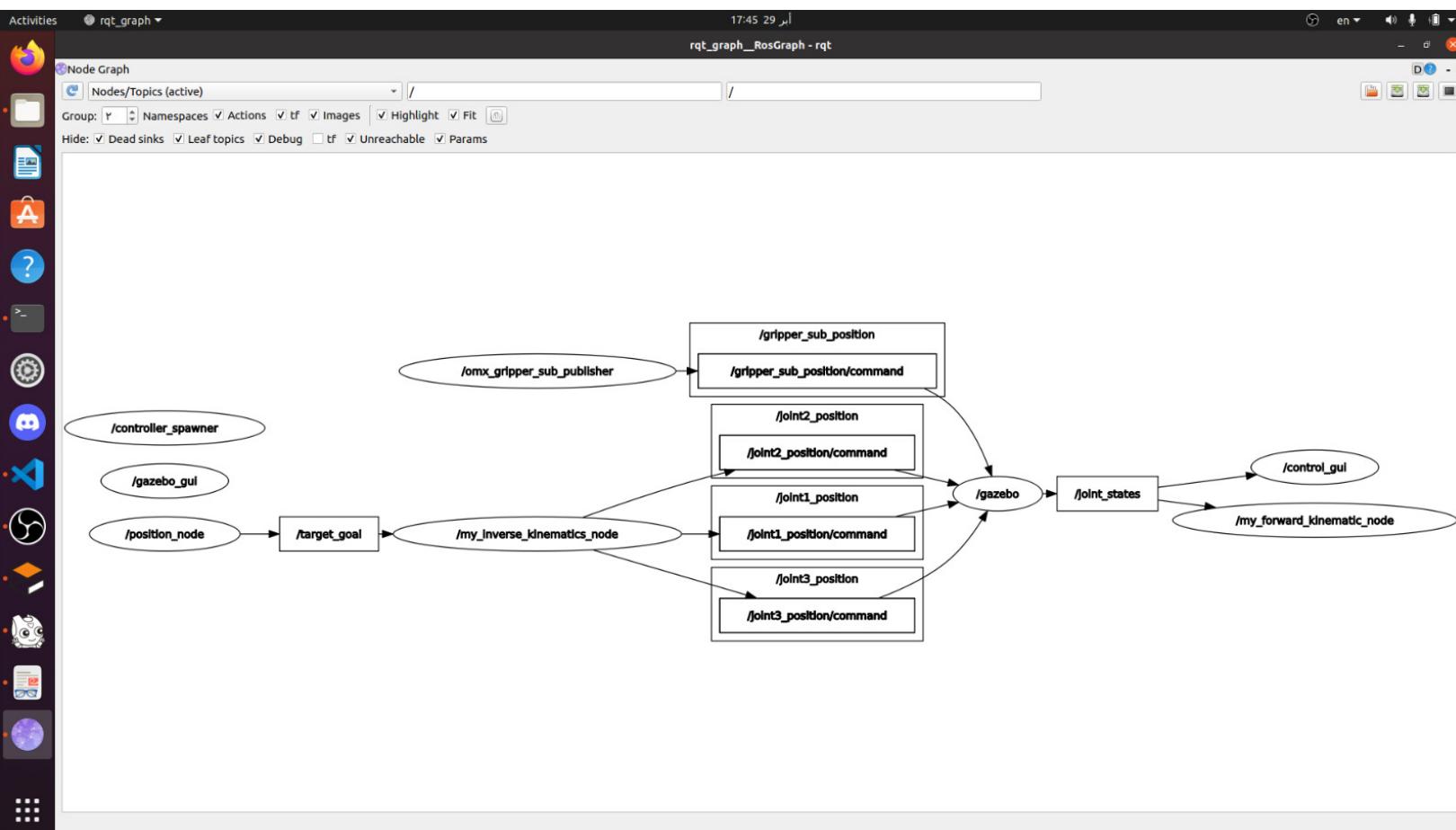


Figure 16: RQT graph

## Links:

In this [link](#), you will find our video for this milestone.

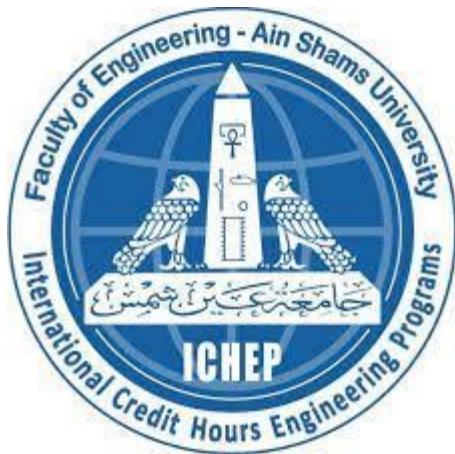
In this [link](#), you will find our data for this milestone.

## References:

[OpenMANIPULATOR-X \(robotis.com\)](#)



University of  
East London



## MILESTONE 3

### Team 5

Kirolos Thabet Fouad  
Omar Eltoutongy

19P6754  
19P1060

## Table of Contents:

1.	Introduction: .....	3
2.	Trajectory Planning Node:.....	4
3.	RQT Graph: .....	5
4.	Environment From Gazebo:.....	6
5.	Simulation Video Link:.....	8

## Table of Figures:

Figure 1:Open manipulator robot.....	3
Figure 2: Code Part2.....	4
Figure 3: Code Part1.....	4
Figure 4: Code Part3.....	5
Figure 5: RQT Graph (all) .....	5
Figure 6: Gazebo Environment.....	6
Figure 7: RQT Graph (active) .....	6
Figure 8: Robot reached in Gazebo.....	7
Figure 9:Robot is moving in Gazebo.....	7

## 1. Introduction:

In this report, a significant stage in our project that will showcase the team's progress and achievements. In this milestone, we will be focusing on developing an environment using GAZEBO, a powerful simulation tool, where our robot, tables, and a task object will coexist. The primary objective of this milestone is to demonstrate the robot's ability to execute a pick-and-place task, successfully moving the task object from one location to another, while ensuring collision-free operation.

By reaching this milestone, our team aims to showcase our proficiency in designing and implementing a functional simulation environment, as well as our ability to program the robot's movements effectively. This milestone is crucial as it tests the core capabilities of the robot in terms of navigation, perception, and manipulation, emphasizing the importance of seamless and safe interaction with its surroundings.

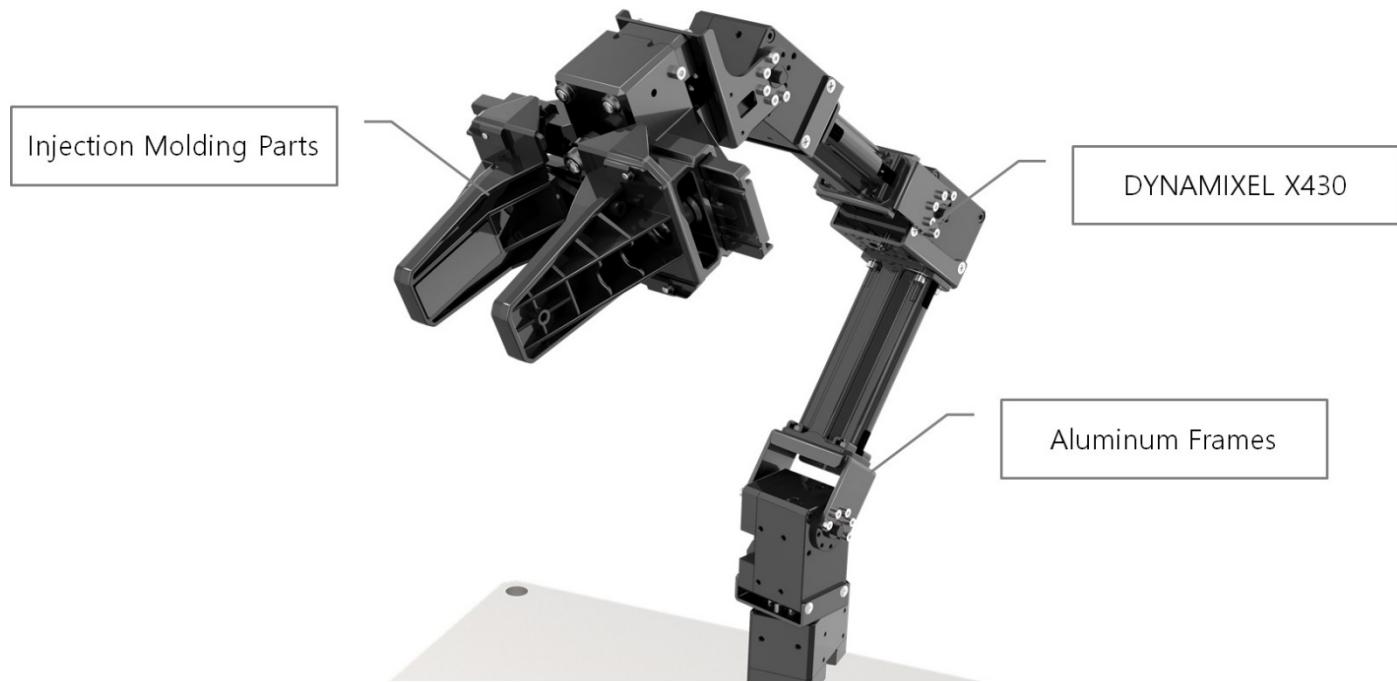


Figure 1:Open manipulator robot

## 2. Trajectory Planning Node:

This node moves the robot end effector to the required position at which the object needed to be moved is. The motion is performed by two joints (joint1 & joint 2& joint3). Then, the gripper closes to hold the object. After that, the robot moves the end effector to the desired release position. Finally, the gripper opens to release the object.

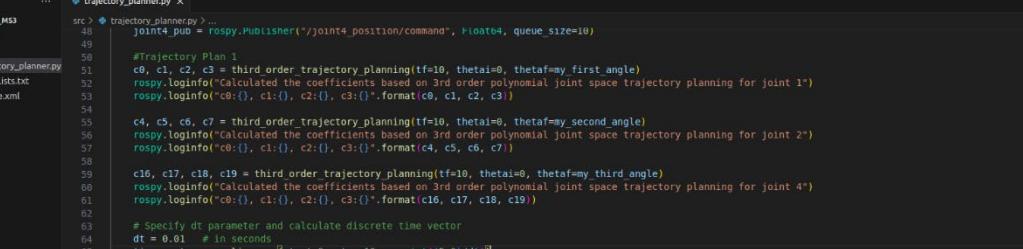
The screenshot shows the Visual Studio Code interface with the following details:

- Activity Bar:** Activities, Visual Studio Code, en.
- Header:** 17 May 10:21 AM, trajectory\_planner.py - robotics\_ms3 - Visual Studio Code
- Sidebar:** File Explorer showing the project structure: ROBOTICS\_MS3, vscode, src, trajectory\_planner.py, CMakeLists.txt, package.xml.
- Code Editor:** The trajectory\_planner.py file is open, displaying Python code for trajectory planning. The code includes imports for numpy, std\_msgs.msg, and rospy, as well as definitions for grip(), release(), and third\_order\_trajectory\_planning() functions. It also initializes ROS nodes and publishers for gripper control.
- Right Panel:** Shows the Python file's content in a large preview pane.

```
File Edit Selection View Go Run Terminal Help

trajectory_planner.py x
src > trajectory_planner.py > ...
1 import rospy
2 from std_msgs.msg import Float64, Bool
3 import numpy as np
4
5 def grip():
6     gripper_pos_pub.publish(Float64(-0.1))
7     gripper_grip_pub.publish(Bool(True))
8
9 def release():
10    gripper_pos_pub.publish(Float64(0.1))
11    gripper_grip_pub.publish(Bool(False))
12
13 def third_order_trajectory_planning(tf, thetaf, theta0):
14     # Calculate third order polynomial coefficients
15     a = np.array([[1, 0, 0, 0],
16                  [1, tf, tf**2, tf**3],
17                  [0, 1, 0, 0],
18                  [0, 1, 2*tf, 3*(tf**2)]])
19     b = np.array([thetaf,
20                  thetaf,
21                  0,
22                  0])
23
24     poly_coeffs = np.linalg.solve(a, b).reshape(-1).tolist()
25     c0, c1, c2, c3 = poly_coeffs[0], poly_coeffs[1], poly_coeffs[2], poly_coeffs[3]
26
27     return c0, c1, c2, c3
28
29 if __name__ == "__main__":
30
31     #Used angles
32     my_first_angle = -0.2
33     my_second_angle = 0.185
34     my_third_angle = 0.15
35
36     # Initialize ROS node
37     rospy.init_node("trajectory_planner_node")
38
39     # Gripper publishers
40
41     #The 'latch=True' parameter in the 'rospy.Publisher' function ensures that the gazebo receives the last message
42     #and if not ready, the keep it alive until it is ready.
43     gripper_pos_pub = rospy.Publisher("/gripper position/command", Float64, queue_size=10, latch=True)
44     gripper_grip_pub = rospy.Publisher("/gripper attach cmd", Bool, queue_size=10, latch=True)
45
46     # Create the publishers that'll move the robot's joints
47     joint1_pub = rospy.Publisher('/joint1 position/command', Float64, queue_size=10)
```

*Figure 3: Code Part 1*



The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Activities > Visual Studio Code - trajectory\_planner.py - robotics\_ms3 - Visual Studio Code
- File Explorer:** Shows the project structure with files like trajectory\_planner.py, CMakeLists.txt, and package.xml.
- Code Editor:** Displays the Python script trajectory\_planner.py. The code performs trajectory planning for a robotic arm using ROS and Pybullet. It includes comments explaining the planning of third-order polynomials for joints 1 through 5, with specific angles for the first and second joints. It also handles gripper movement and terminates the process.

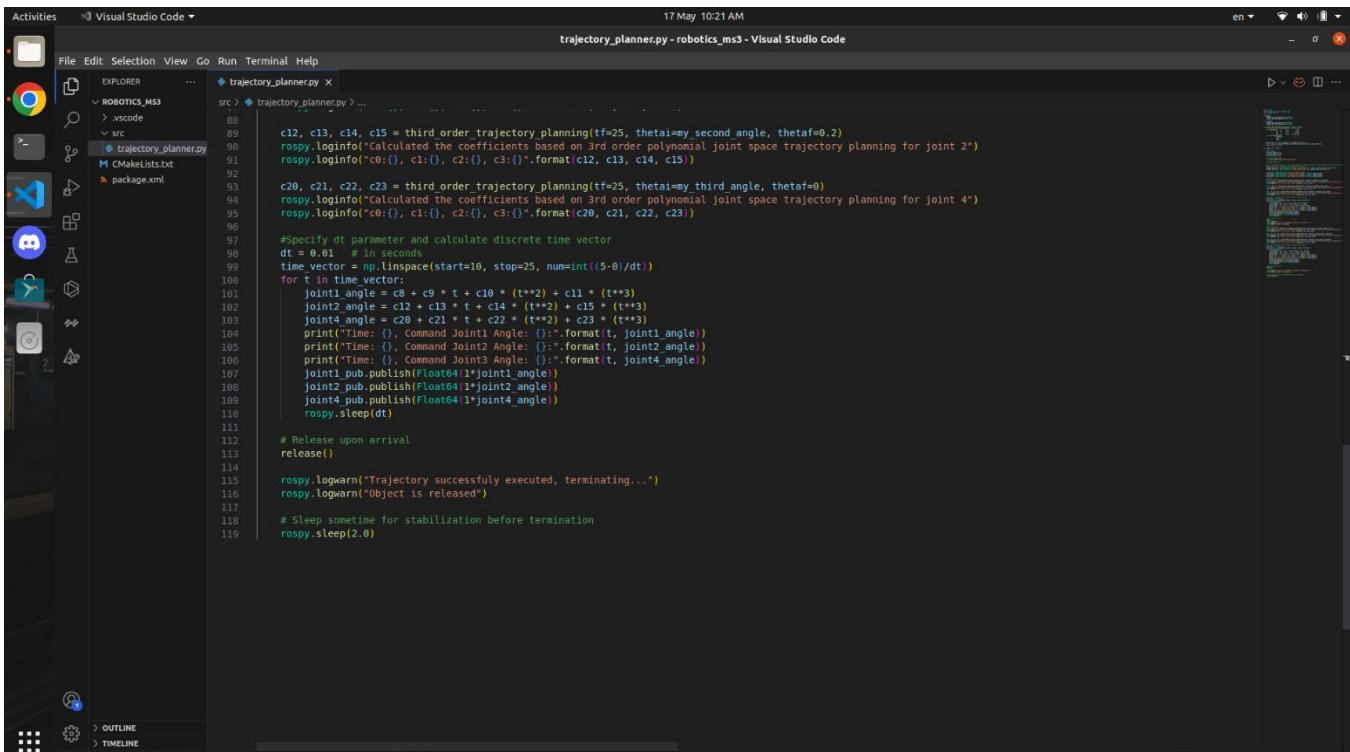
```
17 May 10:21 AM
```

```
trajectory_planner.py - robotics_ms3 - Visual Studio Code
```

```
File Edit Selection View Go Run Terminal Help
```

```
trajectory_planner.py x
src > trajectory_planner.py ...
48     joint4_pub = rospy.Publisher('/joint4_position/command', Float64, queue_size=10)
49
50     #Trajectory Plan 1
51     c0, c1, c2, c3 = third_order_trajectory_planning(tf=10, theta1=0, thetaf=my_first_angle)
52     rospy.loginfo("Calculated the coefficients based on 3rd order polynomial joint space trajectory planning for joint 1")
53     rospy.loginfo("c0:{}, c1:{}, c2:{}, c3:{}".format(c0, c1, c2, c3))
54
55     c4, c5, c6, c7 = third_order_trajectory_planning(tf=10, theta1=0, thetaf=my_second_angle)
56     rospy.loginfo("Calculated the coefficients based on 3rd order polynomial joint space trajectory planning for joint 2")
57     rospy.loginfo("c4:{}, c5:{}, c6:{}, c7:{}".format(c4, c5, c6, c7))
58
59     c16, c17, c18, c19 = third_order_trajectory_planning(tf=10, theta1=0, thetaf=my_third_angle)
60     rospy.loginfo("Calculated the coefficients based on 3rd order polynomial joint space trajectory planning for joint 4")
61     rospy.loginfo("c16:{}, c17:{}, c18:{}, c19:{}".format(c16, c17, c18, c19))
62
63     # Specify dt parameter and calculate discrete time vector
64     dt = 0.01 # in seconds
65     t = np.linspace(start=0, stop=10, num=int((5-0)/dt))
66
67     for i in range(len(t)):
68         joint1_angle = c0 + c1 * t[i] + c2 * (t[i]**2) + c3 * (t[i]**3)
69         joint2_angle = c4 + c5 * t[i] + c6 * (t[i]**2) + c7 * (t[i]**3)
70         joint4_angle = c16 + c17 * t[i] + c18 * (t[i]**2) + c19 * (t[i]**3)
71         print("Time: {}, Command Joint1 Angle: {}".format(t[i], joint1_angle))
72         print("Time: {}, Command Joint2 Angle: {}".format(t[i], joint2_angle))
73         print("Time: {}, Command Joint4 Angle: {}".format(t[i], joint4_angle))
74         joint1_pub.publish(Float64(1*joint1_angle))
75         joint2_pub.publish(Float64(1*joint2_angle))
76         joint4_pub.publish(Float64(1*joint4_angle))
77         rospy.sleep(dt)
78
79     # Grip before moving
80     grip()
81     rospy.sleep(2.0)
82     rospy.logwarn("Trajectory successfully executed, terminating...")
83     rospy.logwarn("Object is gripped")
84
85     #Trajectory Plan 2
86     c8, c9, c10, c11 = third_order_trajectory_planning(tf=25, theta1=my_first_angle, thetaf=-np.pi)
87     rospy.loginfo("Calculated the coefficients based on 3rd order polynomial joint space trajectory planning for joint 1")
88     rospy.loginfo("c8:{}, c9:{}, c10:{}, c11:{}".format(c8, c9, c10, c11))
89
90     c12, c13, c14, c15 = third_order_trajectory_planning(tf=25, theta1=my_second_angle, thetaf=0.2)
91     rospy.loginfo("Calculated the coefficients based on 3rd order polynomial joint space trajectory planning for joint 2")
92     rospy.loginfo("c12:{}, c13:{}, c14:{}, c15:{}".format(c12, c13, c14, c15))
```

*Figure 2: Code Part2*



```

Activities 17 May 10:21 AM
File Edit Selection View Go Run Terminal Help
 trajecty_planner.py - robotics_ms3 - Visual Studio Code
src > trajecty_planner.py ...
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119

#Specify dt parameter and calculate discrete time vector
dt = 0.01 # in seconds
time_vector = np.linspace(start=10, stop=25, num=int((5-0)/dt))
for t in time_vector:
    joint1_angle = c0 + c9 * t + c10 * (***) + c11 * (***)
    joint2_angle = c12 + c13 * t + c14 * (***) + c15 * (***)
    joint3_angle = c20 + c21 * t + c22 * (***) + c23 * (***)
    print("Time: {} Command Joint1 Angle: {}".format(t, joint1_angle))
    print("Time: {} Command Joint2 Angle: {}".format(t, joint2_angle))
    print("Time: {} Command Joint3 Angle: {}".format(t, joint3_angle))
    joint1_pub.publish(Float64(1*joint1_angle))
    joint2_pub.publish(Float64(1*joint2_angle))
    joint3_pub.publish(Float64(1*joint3_angle))
    rospy.sleep(dt)

# Release upon arrival
release()

rospy.logwarn("Trajectory successfully executed, terminating...")
rospy.logwarn("Object is released")

# Sleep sometime for stabilization before termination
rospy.sleep(2.0)

```

Figure 4: Code Part3

### 3. RQT Graph:

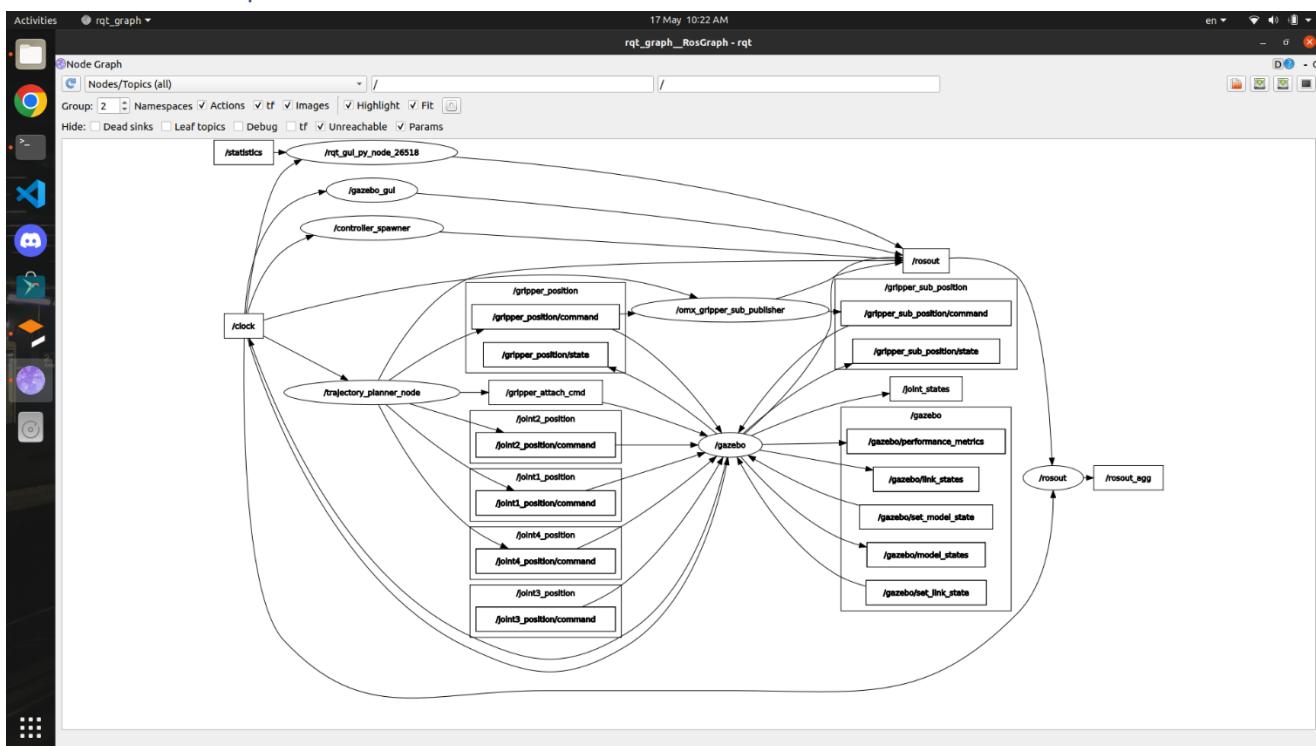


Figure 5: RQT Graph (all)

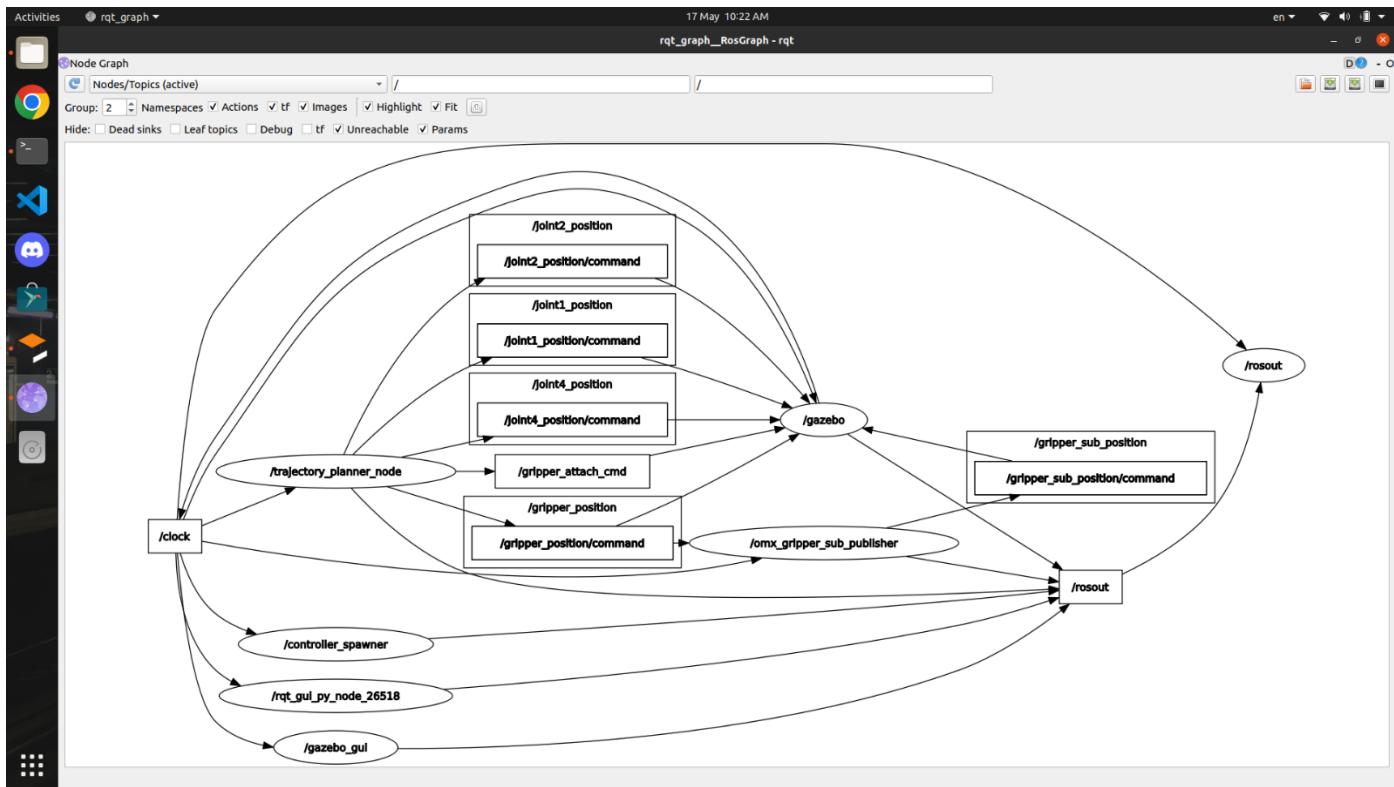


Figure 7: RQT Graph (active)

#### 4. Environment From Gazebo:

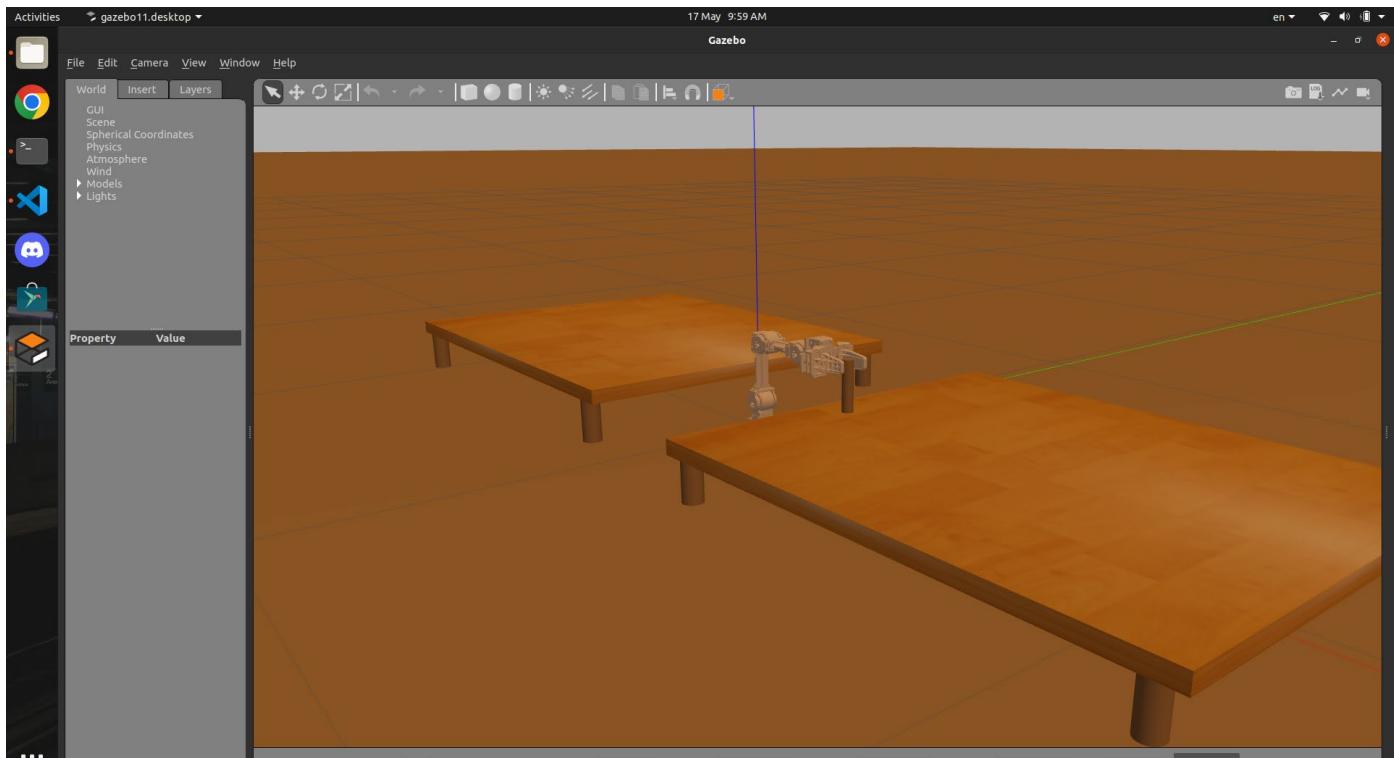


Figure 6: Gazebo Environment

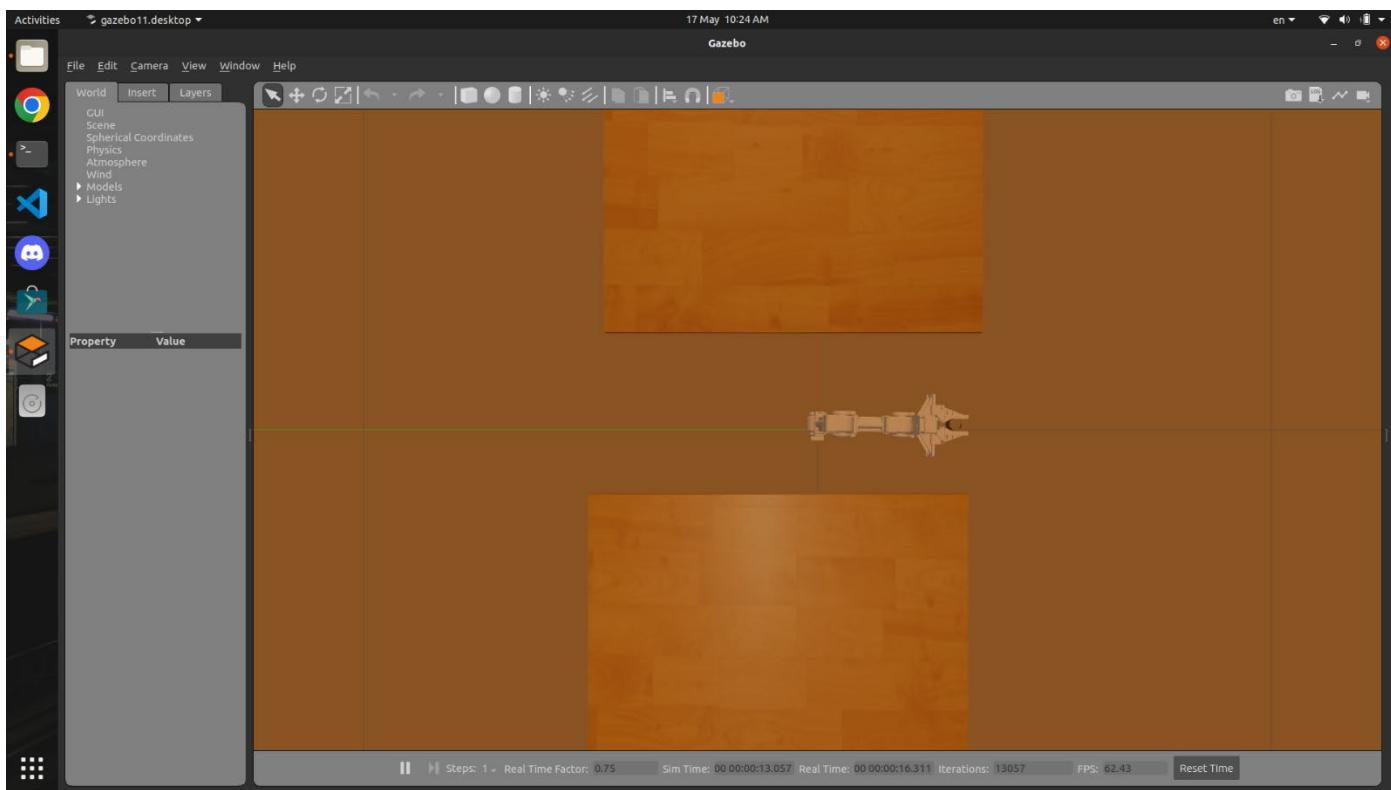


Figure 9: Robot is moving in Gazebo.

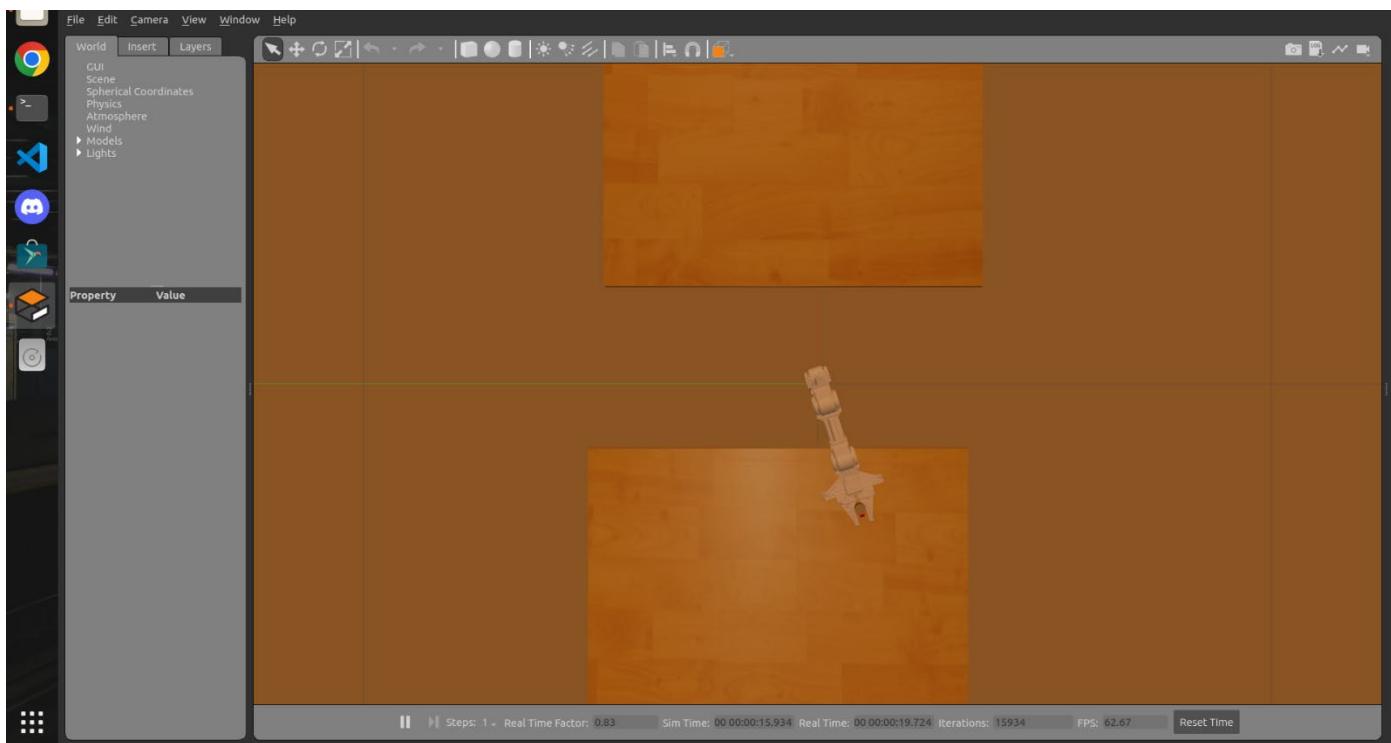


Figure 8: Robot reached in Gazebo.



University of  
East London

## 5. Simulation Video Link:

You can watch the video of our project [here](#).