

# 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

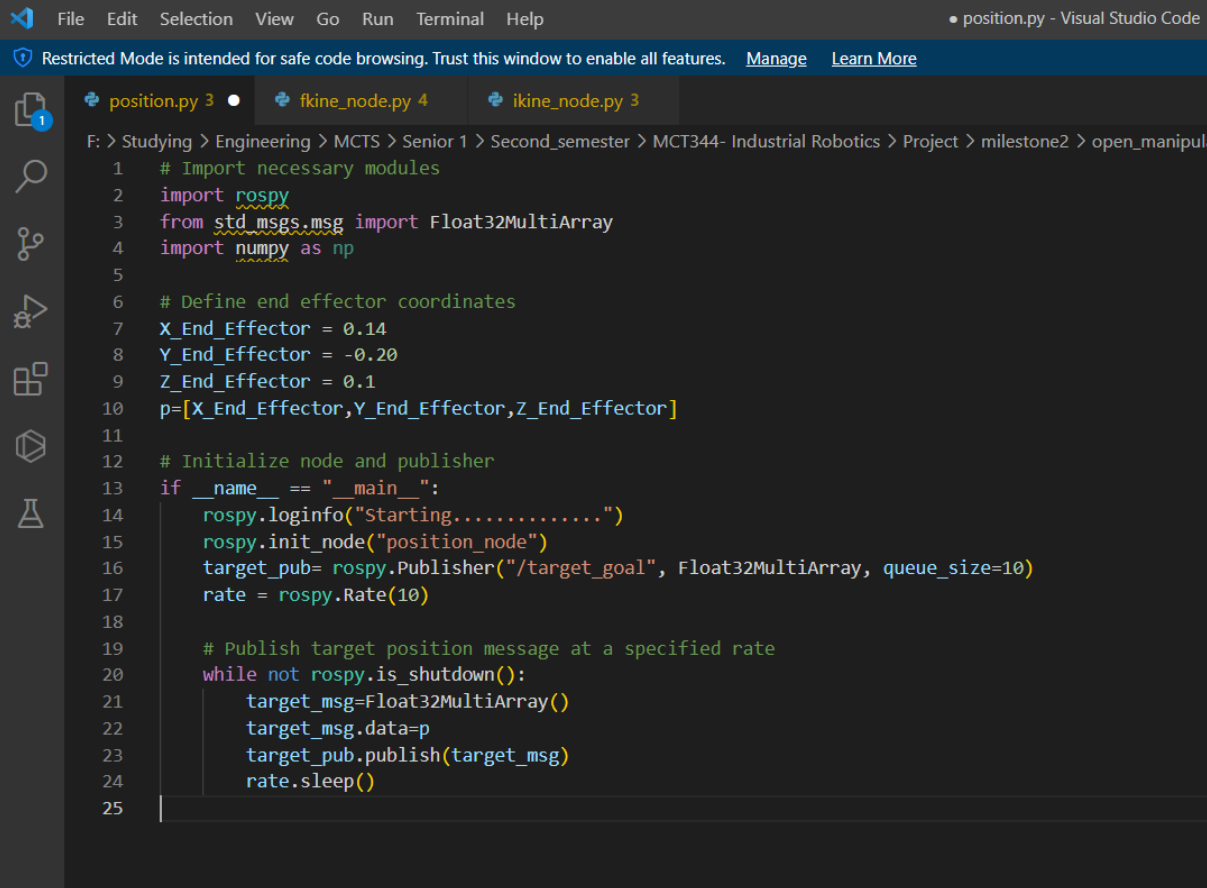
## Introduction:

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.



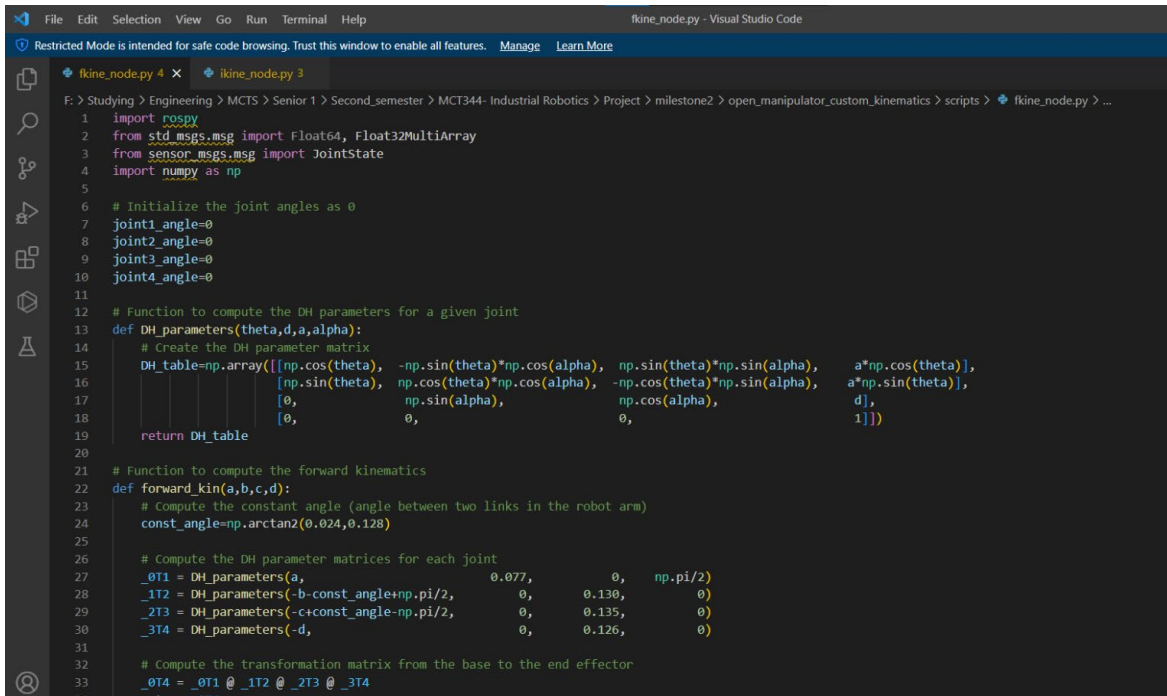
```
F:\> Studying > Engineering > MCTS > Senior 1 > Second_semester > MCT344- Industrial Robotics > Project > milestone2 > open_manipula

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

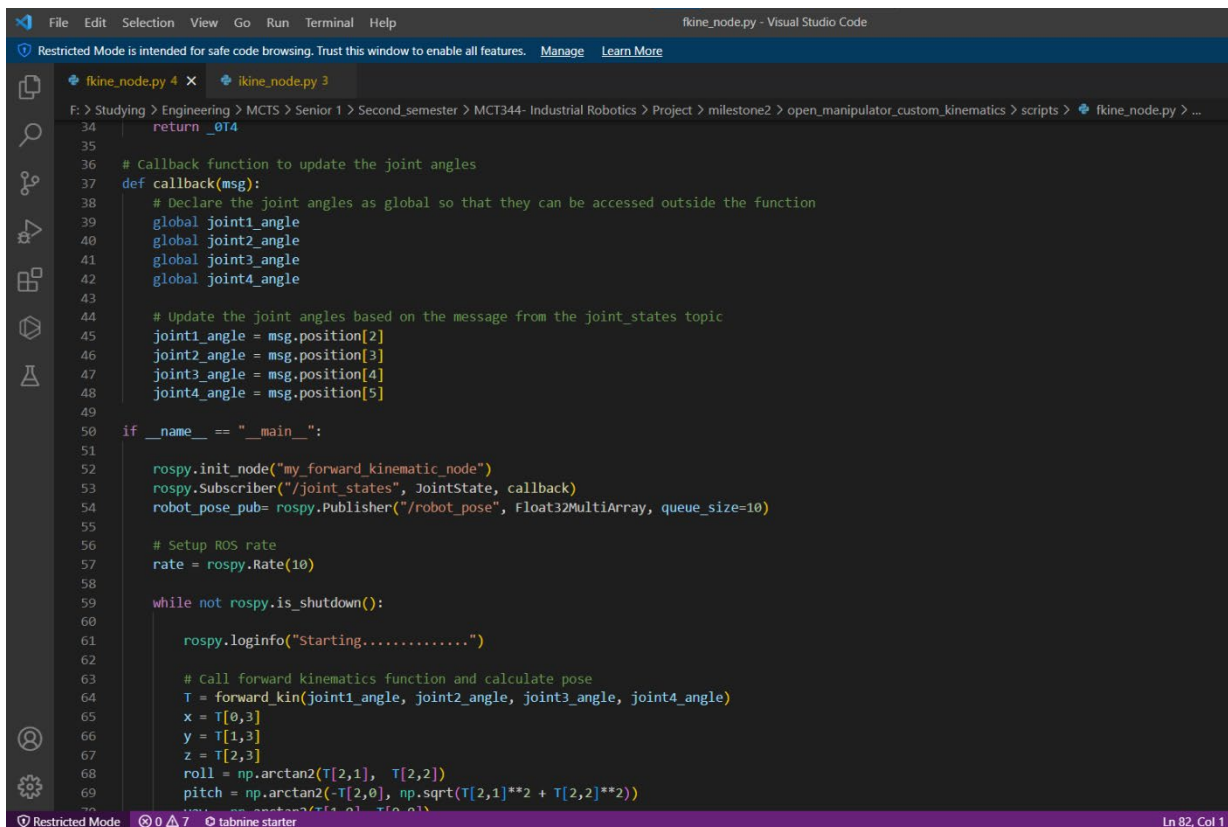
## 2. fkine node:

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.



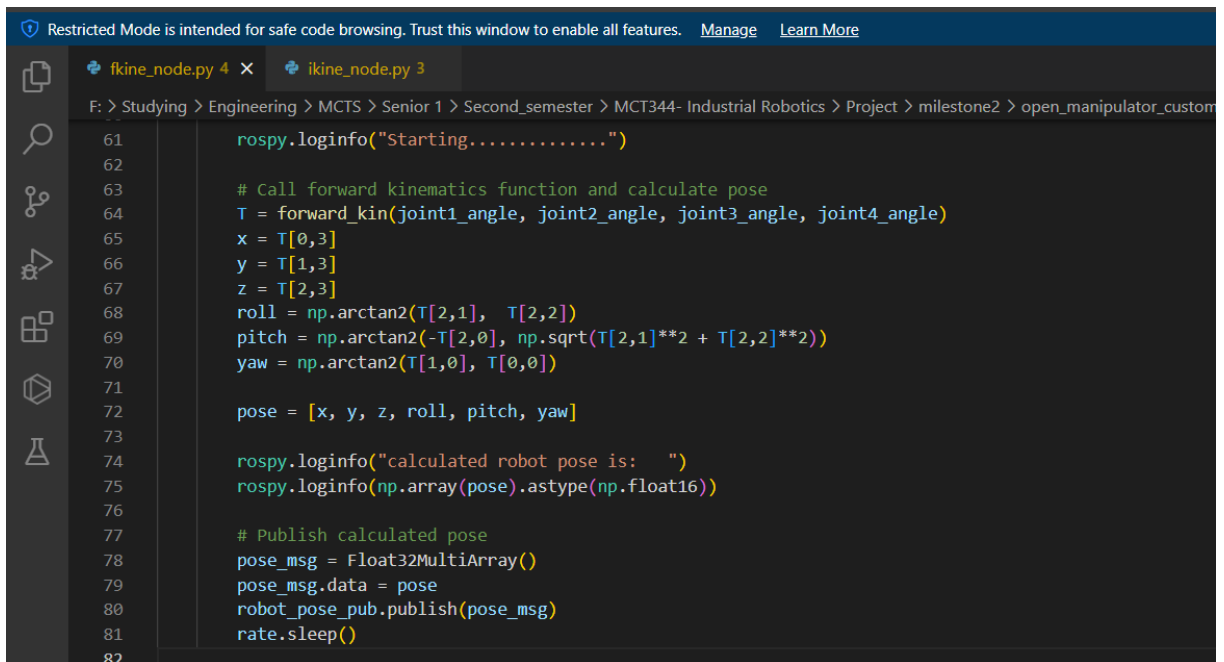
```
1 import rospy
2 from std_msgs.msg import Float64, Float32MultiArray
3 from sensor_msgs.msg import JointState
4 import numpy as np
5
6 # Initialize the joint angles as 0
7 joint1_angle=0
8 joint2_angle=0
9 joint3_angle=0
10 joint4_angle=0
11
12 # Function to compute the DH parameters for a given 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     _0T1 = DH_parameters(a, 0.077, 0, np.pi/2)
28     _1T2 = DH_parameters(-b-const_angle+np.pi/2, 0, 0.130, 0)
29     _2T3 = DH_parameters(-c+const_angle-np.pi/2, 0, 0.135, 0)
30     _3T4 = DH_parameters(-d, 0, 0.126, 0)
31
32     # Compute the transformation matrix from the base to the end effector
33     _0T4 = _0T1 @ _1T2 @ _2T3 @ _3T4
34     return _0T4
```

Figure 3: fkine node part1



```
34     return _0T4
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     rospy.init_node("my_forward_kinematic_node")
52     rospy.Subscriber("/joint_states", JointState, callback)
53     robot_pose_pub= rospy.Publisher("/robot_pose", Float32MultiArray, queue_size=10)
54
55     # Setup ROS rate
56     rate = rospy.Rate(10)
57
58     while not rospy.is_shutdown():
59         rospy.loginfo("Starting.....")
60
61         # Call forward kinematics function and calculate pose
62         T = forward_kin(joint1_angle, joint2_angle, joint3_angle, joint4_angle)
63         x = T[0,3]
64         y = T[1,3]
65         z = T[2,3]
66         roll = np.arctan2(T[2,1], T[2,2])
67         pitch = np.arctan2(-T[2,0], np.sqrt(T[2,1]**2 + T[2,2]**2))
68         yaw = np.arctan2(T[1,0], T[0,0])
69
70     rospy.loginfo("Exiting.....")
```

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:~$ rosmmsg 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 X
F: > Studying > Engineering > MCTS > Senior 1 > Second_semester > MCT344- Industrial Robotics > Project > milestone2 > open_manipulator_custom_ki
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)

```

Figure 7: ikine node part 1

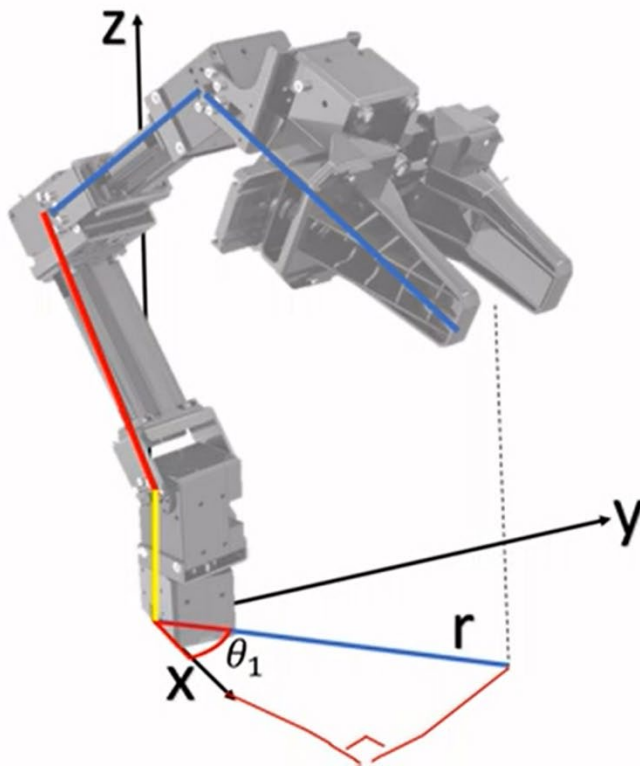
```

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 X
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

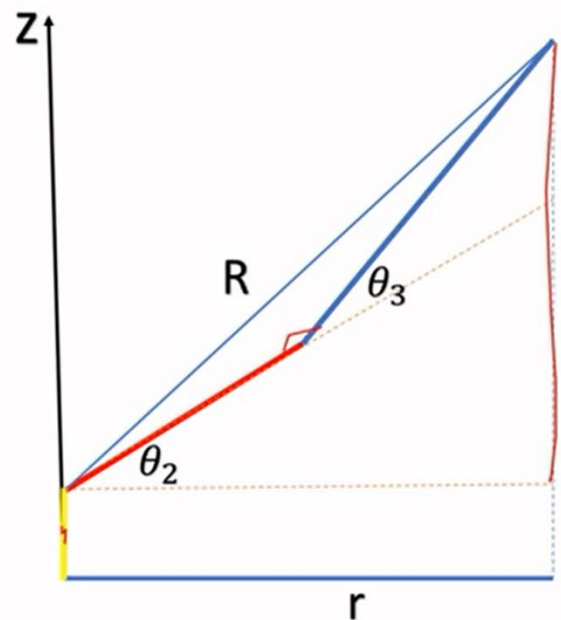
Applying cos rule to get  $\theta_3$ :

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

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

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

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



Note: d represent the link length

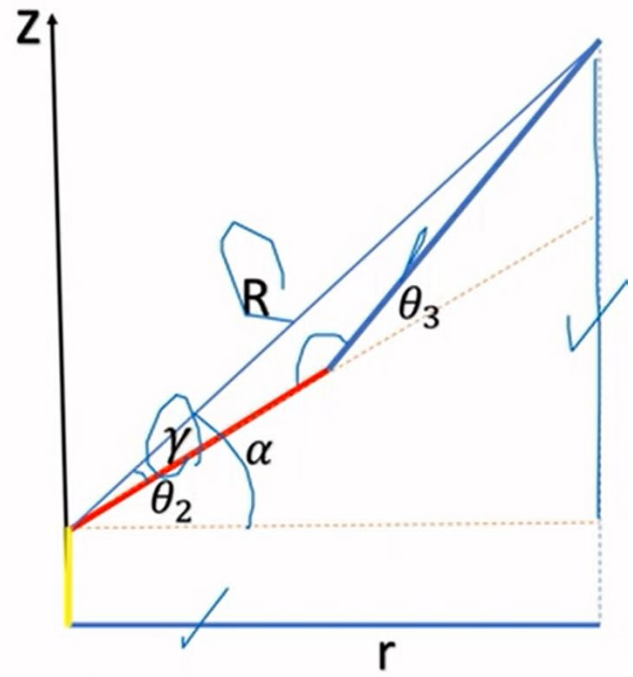
Figure 8: geometrical explain p2

Applying sin rule to get  $\theta_2$ :

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

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

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



Note: d represent the link length

Figure 10: geometrical explain p3



Screenshots of the outputs:

```

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

The terminal window shows the following output for the command `rostopic echo /robot_pose`:

```

data: [0.13142161071300507, -0.20534661412239075, 0.09133944660425186, 1.570796
3705062866, 2.5844579795375466e-05, -1.0014839172363281]
layout:
  dim: []
  data_offset: 0
data: [0.13142170011997223, -0.20534643530845642, 0.09134190529584885, 1.570796
3705062866, 1.5481486116186716e-05, -1.0014832019805908]
layout:
  dim: []
  data_offset: 0
data: [0.13142164051532745, -0.2053464651107780, 0.09134205430746078, 1.5707963
705062866, 1.4869442566123325e-05, -1.0014835596084595]
layout:
  dim: []
  data_offset: 0
data: [0.131421759724617, -0.20534637570381165, 0.09134252369403839, 1.57079637
05062866, 1.2891207916254643e-05, -1.0014828443527222]
layout:
  dim: []
  data_offset: 0
data: [0.131421759724617, -0.20534637570381165, 0.09134252369403839, 1.57079637
05062866, 1.2891207916254643e-05, -1.0014828443527222]

```

The OpenManipulator control GUI shows the following data:

Joint	Position (rad)	Position (m)
Joint 1	-1.001	X 0.140
Joint 2	0.120	Y -0.200
Joint 3	0.828	Z 0.100
Joint 4	-0.948	Gripper -0.000

Figure 12: topic echoing robot pose

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

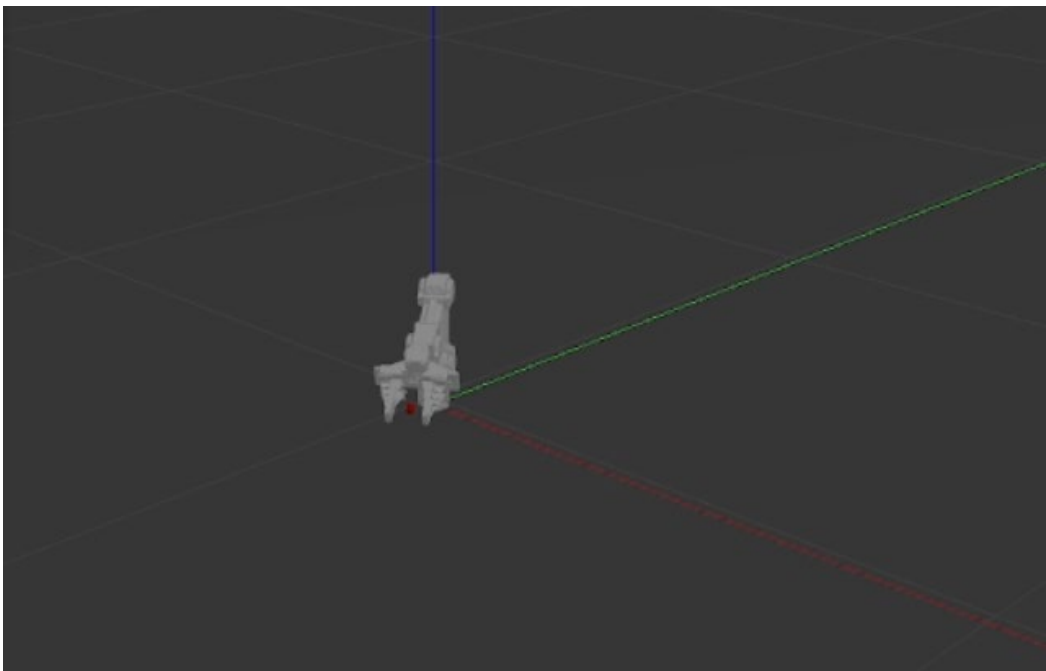


Figure 13: robot after published the required position

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

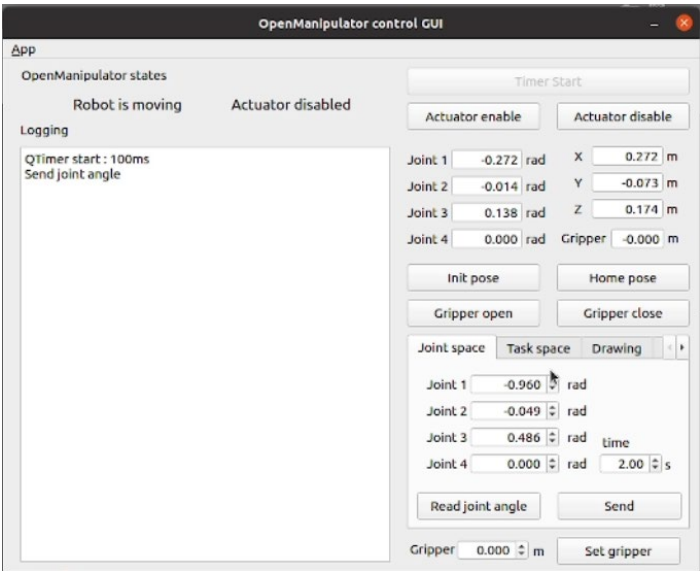


Figure 15: sending the calculated angles from the ikine node.

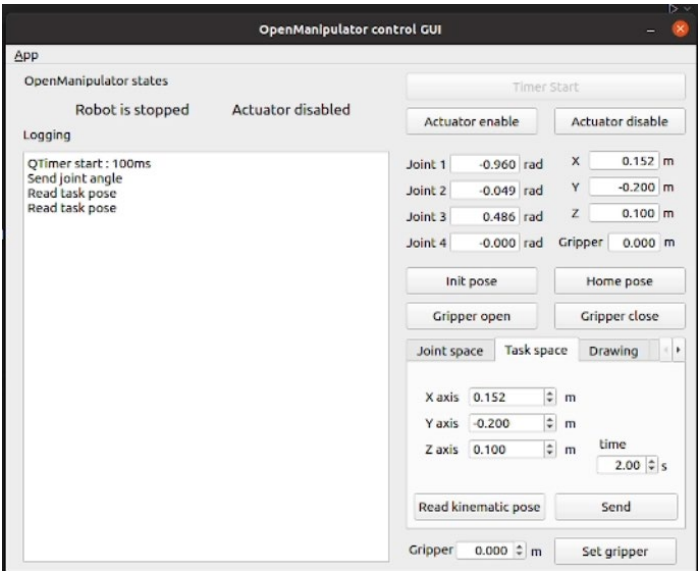


Figure 14:reading the position to compare with the target position node.

## RQT Graph:

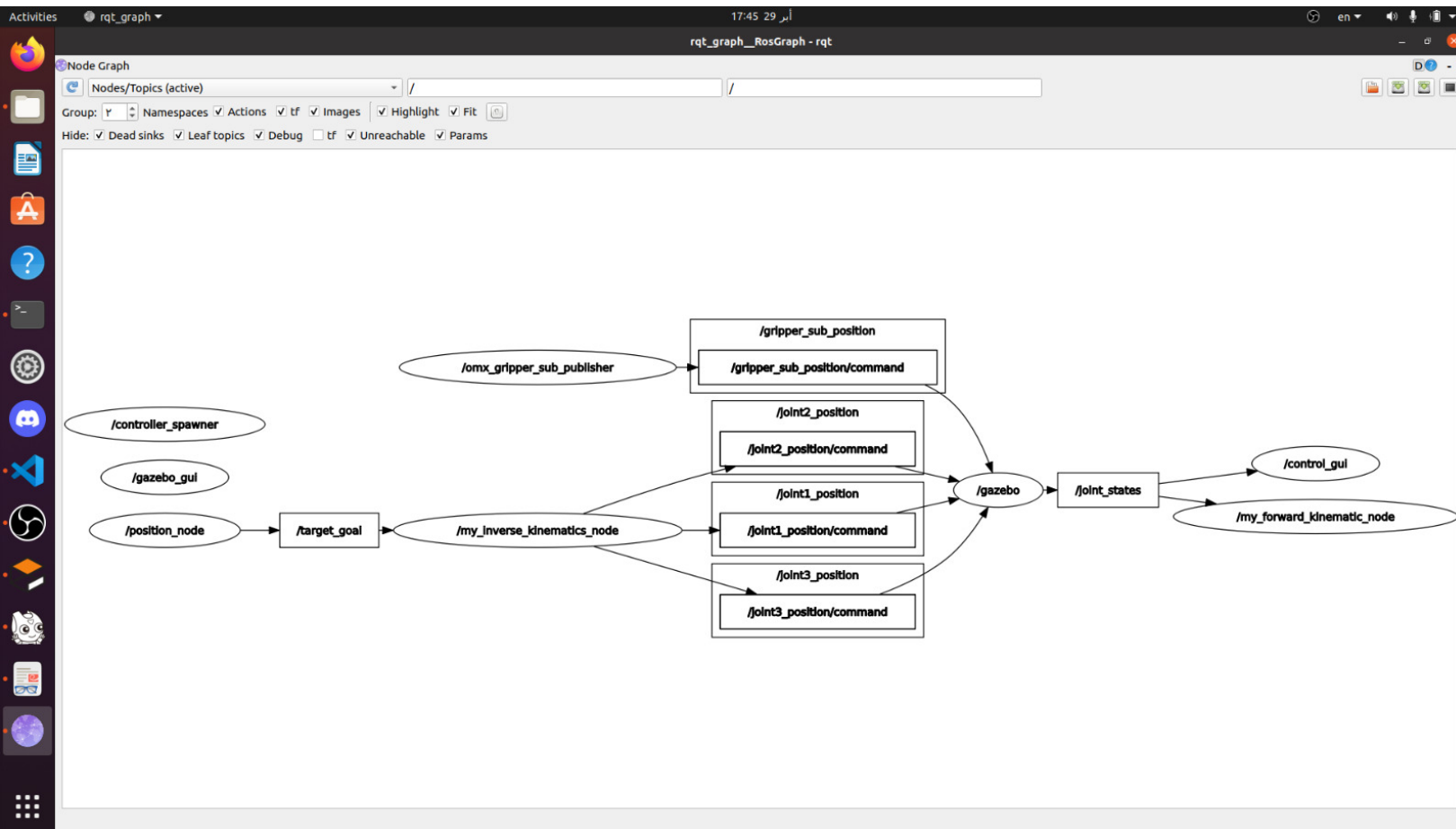


Figure 16: RQT graph

### 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 \(robotis.com\)](#)