ROS code

Function - 1

```python
import rclpy
from rclpy.node import Node
import math
from std_msgs.msg import Float32MultiArray
from geometry_msgs.msg import Pose

##globar variables; link lengths of a given robotic manipulator
l1=1
l2=1
l3=1

def angle_to_config(q_angles):

        ## initialize angle variables with converion of q angles to radians from degree
        q1=math.radians(q_angles[0])
        q2=math.radians(q_angles[1])
        q3=math.radians(q_angles[2])



        ## calculate robot's position vector using forward kinematics
        x = (l3 * math.cos(q1)* math.cos(q2) * math.cos(q3)) - (l3 * math.cos(q1) * math.sin(q2) * math.sin(q3)) + (l2 * math.cos(q2)*math.cos(q1))
        y = (l3 * math.sin(q1)* math.cos(q2)*math.cos(q3)) - (l3* math.sin(q1)* math.sin(q2) * math.sin(q3)) + (l2 * math.cos(q2)*math.sin(q1))
        z = -(l3 * math.sin(q2)* math.cos(q3)) - (l3* math.cos(q2) * math.sin(q3)) - (l2* math.sin(q2)) + l1

        ##x = (l3 * math.cos(q1)* math.cos(q2) * math.cos(q3)) - (l3 * math.cos(q1) * math.sin(q2) * math.sin(q3))
        ##y = (l3 * math.sin(q1)* math.cos(q2)* math.cos(q3)) - (l3* math.sin(q1)* math.sin(q2) * math.sin(q3))
        ##z = l1 -(l2* math.sin(q2)) - (l3 * math.sin(q2)* math.cos(q3)) - (l3* math.cos(q2) * math.sin(q3))

        ## calculate rotation matrix components using forward kinematics
        r11 = (math.cos(q1)* math.cos(q2) * math.cos(q3) )- (math.cos(q1)* math.sin(q2) * math.sin(q3))
        r12 = -(math.cos(q1)* math.cos(q2) * math.sin(q3) ) - (math.cos(q1)* math.sin(q2) * math.cos(q3))
        r13 = -math.sin(q1)

        r21 = (math.cos(q3) * math.sin(q1)) - (math.sin(q1) * math.sin(q2) * math.sin(q3))
        r22 = -(math.sin(q1) * math.sin(q3)) - (math.sin(q1) * math.sin(q2) * math.cos(q3))
        r23 = math.cos(q1)

        r31 = -(math.sin(q2) * math.cos(q3)) - (math.cos(q2) * math.sin(q3))
        r32 = (math.sin(q2) * math.sin(q3)) - (math.cos(q2) * math.cos(q3))
        r33 = 0


        return [r11,r12,r13, r21,r22,r23, r31,r32,r33, x, y,z]

```

Function2

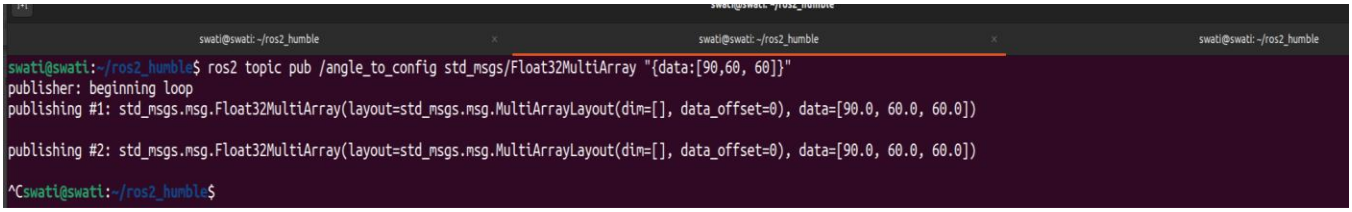
```
45
46
47 def config_to_angle(config):
48         ##extract postion info from the geometry message recived
49         ##position: holds info of [x,y,z] position of an end effector
50
51         position = config
52
53
54         ##extract x,y,x values
55         x_value = position.x
56         y_value = position.y
57         z_value = position.z
58
59         ##Calculate joint angles theta1, theta2 and theta3 for the given robot manipulator
60         ##seeing from top view we can calculate theta_1
61         theta_1 =  math.atan2(y_value,x_value)
62
63         ## calculate theta2 using law of cosine; r_value and s_value are temporary variables used for calculations
64         ## seeing from top view, we can calculate r_value
65         r_value = x_value / math.cos(theta_1)
66         s_value = z_value - l1
67         ##using law of cosine of traingle
68         cos_alpha = -(((r_value ** 2 + s_value ** 2)- l2**2 - l3**2)/(2*l2*l3))
69         ##print (cos_alpha)
70         theta_3 = math.acos(cos_alpha)
71         theta_3 = math.pi - theta_3
72
73
74         ## calculate theta2, seeing from side view, using subtraction of angles
75         theta_2 =  math.atan2(s_value, r_value) - math.atan2(l3 * math.sin(theta_3),( l2 + l3 * math.cos(theta_3)) )
76
77
78
79         ##now convert radians to degree
80         theta_1 = math.degrees(theta_1)
81         theta_2 = math.degrees(theta_2)
82         theta_3 = math.degrees(theta_3)
83
84
85         return [theta_1, theta_2, theta_3]
86
87
88
89
```

```python


class my_node(Node):

    def __init__(self):
        super().__init__('my_node')
        ##this subscriber recives joint angle values and calculates robot end effector configurations using forward kinematics
        self.subscription_1 = self.create_subscription(
            Float32MultiArray,
            'angle_to_config',
            self.listener_callback_1,
            10)
        ## creating a subscriber to recieve configuration values-pose of the end effector
        ## it calculates joint angles from end effector configurations using Inverse Kinematics
        self.subscription_2 = self.create_subscription(
            Pose,
            'config_to_angle',
            self.listener_callback_2,
            10)


        self.subscription_1  # prevent unused variable warning
        self.subscription_2

    def listener_callback_1(self, msg):

        configurations = angle_to_config(msg.data)

        self.get_logger().info('Robot position vector is: "%s"' % configurations[9:12])
        self.get_logger().info('Robot rotation matrix is: "%s"'% configurations[0:9])

    def listener_callback_2(self, msg):
        joint_angles = config_to_angle(msg.position)
        self.get_logger().info('Robot joint angles are [theta_1, theta_2, theta_3]: "%s"' % joint_angles)







def main(args=None):

    rclpy.init(args=args)
    my_node_subscriber = my_node()
    rclpy.spin(my_node_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    my_node_subscriber.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

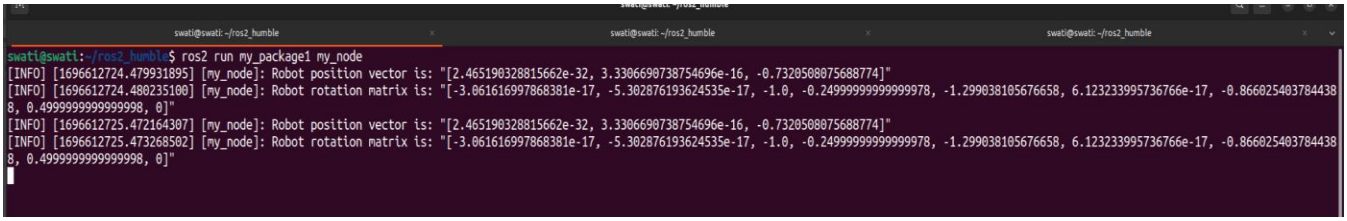
Results

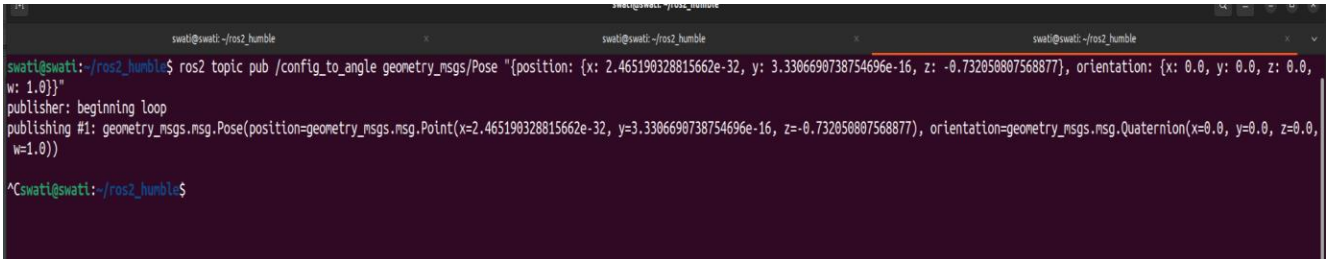
Case 1

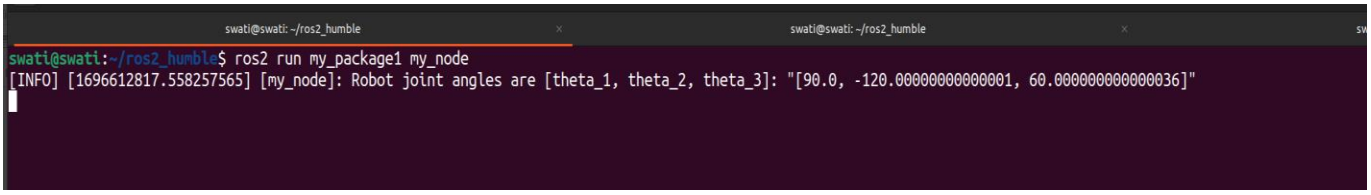
Input to forward kinematics



Output configs - input to inverse kinematics

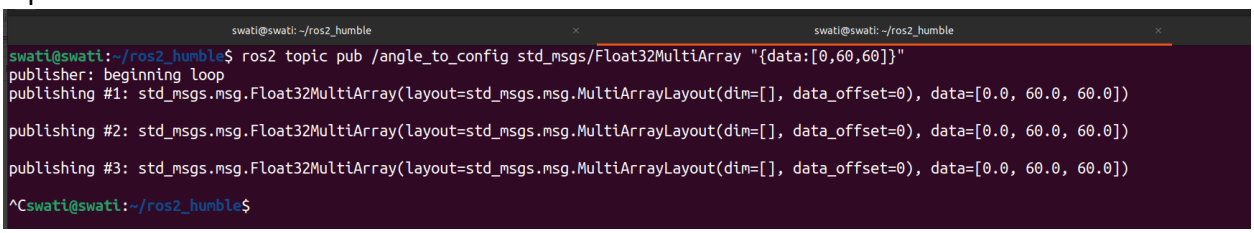




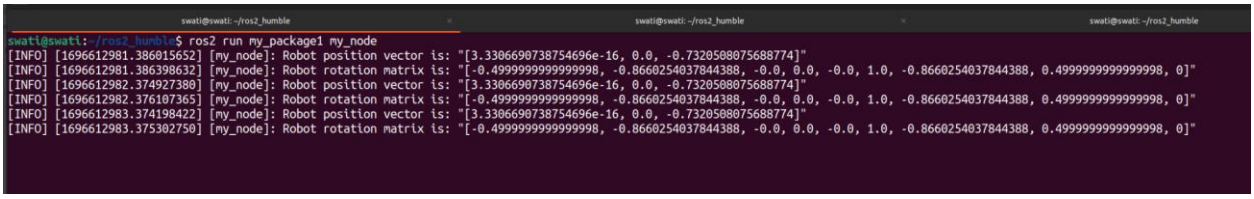


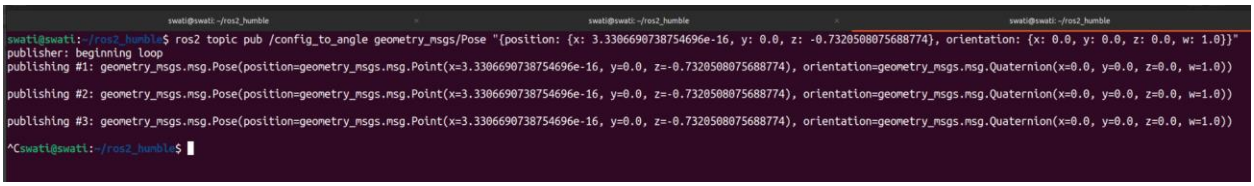
Case 2:

Input to forward kinematics



Config output





Output of inverse linematic