RBE 500 Group Assignment #1

Joshua Gross, Arjan Gupta, Melissa Kelly

Problem 1

Create SCARA Robot in Gazebo

The 3 DOF SCARA robot we have built is shown below.

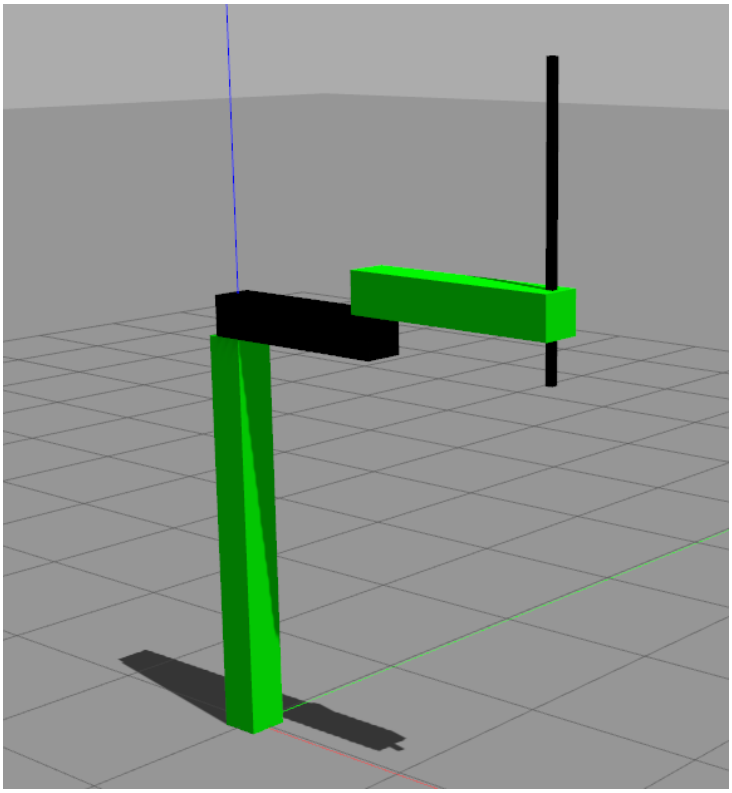


Figure 1: Final SCARA built by our team

We undertook the following steps to create our SCARA robot.

1 — Modify joint locations

In the downloaded package, the RRBot robot has its revolute joints on the 'sides' of its links, as shown in the following figure.

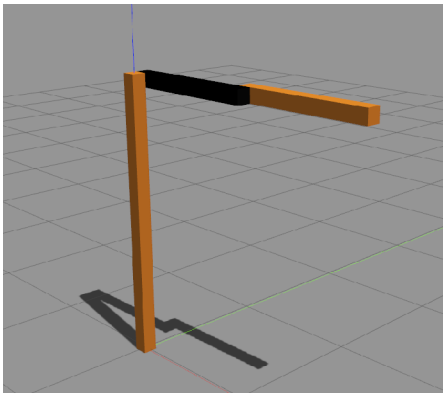


Figure 2: Initial RRBot in Gazebo

However, for a standard SCARA robot, we want the revolute joints to sweep angles in the XY plane of the world frame, not in the XZ plane.
Hence, we edited the `<joint>` element blocks in the URDF file rrbot_description.urdf.xacro. For the first joint, we made the following change.

```
1      <joint name="${prefix}joint1" type="revolute">
2        <parent link="${prefix}base_link"/>
3        <child link="${prefix}link1"/>
4        <!-- Set limits of revolute joint to -90deg to +90deg, 1000 N effort limit, velocity ...
             of 180 rad/s  -->
5        <limit lower="-1.5708" upper="1.5708" effort="1000" velocity="3.14159"/>
6        <origin xyz="0 0 ${height1 + axel_offset*2}" rpy="0 1.5708 0"/>
7        <axis xyz="-1 0 0"/>
8        <dynamics damping="0.7"/>
9      </joint>
```

In the above code snippet, we changed the type attribute of the joint element from continuous to revolute. We also added the limit sub-element, and modified the origin and axis sub-elements. We made similar changes for the second joint, for which the code snippet is shown below.

```
1        <joint name="${prefix}joint2" type="revolute">
2          <parent link="${prefix}link1"/>
3          <child link="${prefix}link2"/>
4          <!-- Set limits of revolute joint to -90deg to +90deg, 1000 N effort limit, ...
               velocity of 180 rad/s  -->
5          <limit lower="-1.5708" upper="1.5708" effort="1000" velocity="3.14159"/>
6          <origin xyz="${width * -1} 0 ${height2 - axel_offset*2}" rpy="0 0 0"/>
7          <axis xyz="-1 0 0"/>
8          <dynamics damping="0.7"/>
9        </joint>
```

As a result, our robot now looked like the following image.

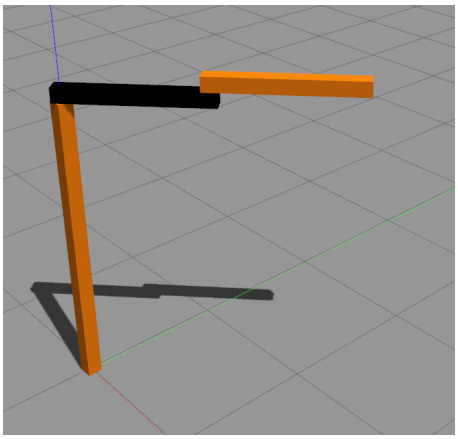


Figure 3: RRBot with top-joints

In order to test our changes, we moved our robot by publishing joint values in the format of the following ROS command in our terminal.

```
ros2 topic pub --once /forward_position_controller/commands ...
    std_msgs/msg/Float64MultiArray "{data: [0.75 0.82]}"
```

2 — Add prismatic joint

Now our revolute joints resemble those of a SCARA robot, but we still need a prismatic joint. In order to do this, we first made the following changes to the rrbot_description.urdf.xacro file.

```
<!-- Tool Joint -->
<joint name="${prefix}tool_joint" type="prismatic">
  <parent link="${prefix}link2"/>
  <child link="${prefix}tool_link" />
  <!-- Set limits of prismatic joint -->
  <limit lower="${prismatic_offset * -1}" upper="1.1" effort="1000" velocity="1"/>
  <origin xyz="${prismatic_offset - height4} 0 ${height3 - axel_offset*2}" rpy="0 ...
      1.5708 0" />
  <axis xyz="0 0 1"/>
  <dynamics damping="0.7"/>
</joint>

<!-- Tool Link -->
<link name="${prefix}tool_link">
  <collision>
    <origin xyz="0 0 ${height4/2 - axel_offset}" rpy="0 0 0"/>
    <geometry>
      <box size="${prismatic_width} ${prismatic_width} ${height4}"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 ${height4/2 - axel_offset}" rpy="0 0 0"/>
    <geometry>
      <box size="${prismatic_width} ${prismatic_width} ${height4}"/>
    </geometry>
  </visual>

  <inertial>
    <origin xyz="0 0 ${height4/2 - axel_offset}" rpy="0 0 0"/>
    <mass value="${mass}"/>
    <inertia
      ixx="${mass / 12.0 * (prismatic_width*prismatic_width + height4*height4)}" ...
          ixy="0.0" ixz="0.0"
      iyy="${mass / 12.0 * (height4*height4 + prismatic_width*prismatic_width)}" ...
          iyz="0.0"
      izz="${mass / 12.0 * (prismatic_width*prismatic_width + ...
          prismatic_width*prismatic_width)}"/>
  </inertial>
</link>
```

In the above code snippet, we have changed the tool joint from fixed to prismatic, defined its translation axis, set its limits, and defined its origin. We defined a special `prismatic_width` so that we could make the tool link thinner than the regular links. We also defined a `prismatic_offset` so that the tool link is slightly 'lowered'. This makes it so that the zero position of the tool link is at the same height as our first link, which makes our calculations consistent with our forward-kinematics derivation.

Next, we defined the tool joint in the rrbot.ros2_control.xacro file, so that the command and state interfaces

for ROS2 could be made available for that joint. Finally, we also edited the gazebo_controllers.yaml file to define the tool joint as part of the three controllers — `forward_position_controller`, `forward_velocity_controller`, and `forward_effort_controller`. All these modified files have been included as part of our submission inside the rrbot_description.zip compressed directory.
At this point, our robot looked as follows.

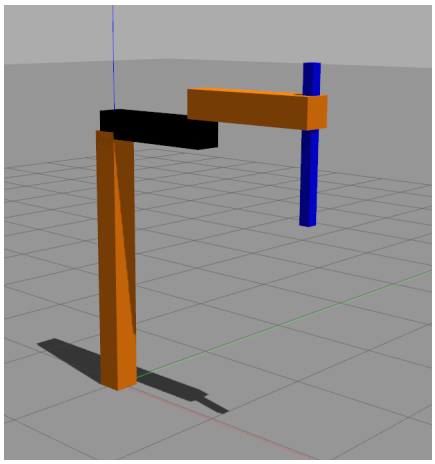


Figure 4: SCARA configuration without finishing touches

After this, we tweaked some offsets, adjusted some link lengths, changed the colors of the robot, and decreased the thickness of the tool link in order to arrive at our finished implementation of the SCARA robot. We also used the same ROS command as earlier (except with 3 data-points this time) to ensure that we were able to move all the joints of our robot.

Question 2

Forward Kinematics for SCARA

Before implementing forward kinematics in ROS, we worked out a derivation for the forward kinematics of the SCARA robot model.

Derviation

The image below shows the frame assignments for the robot.
Based on this, we can form the DH table as shown below.

Link	α_i	a_i	θ_i	d_i
1	0	p_2	q_1	p_1
2	0	p_3	q_2	0
3	0	0	0	q_3

Next, we create a MATLAB script as shown below. We use the matrix obtained from equation 3.10 of the textbook to write A_1, A_2, A_3. By multiplying these matrices, we obtain the T_3^0.

```
1 %% Forward Kinematics
2 clc
3 clear
```

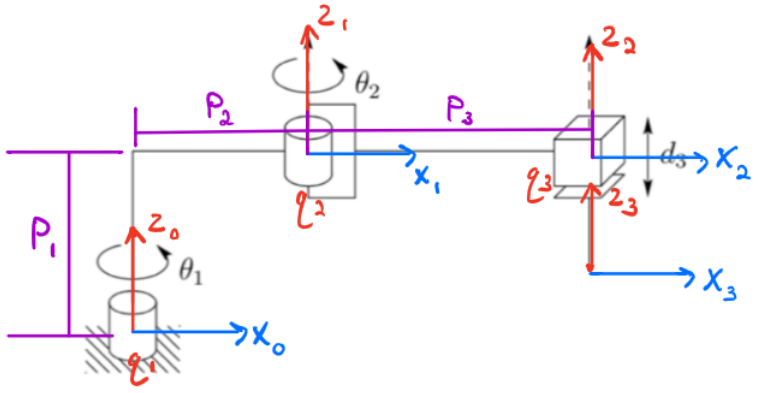


Figure 5: Frame assignments for forward kinematics

```
4
5  syms P1 P2 P3 q1 q2 q3
6  a = [P2 P3 0];
7  theta = [ q1 q2 0];
8  d = [P1 0 q3];
9  alpha = [ 0 0 0];
10
11 % NOTE: Please enter theta in degrees
12
13 i = 1;
14 A1 = [cosd(theta(i)) (-sind(theta(i))*cosd(alpha(i))) (sind(theta(i))*sind(alpha(i))) ...
       (a(i)*cosd(theta(i)));
15     sind(theta(i)) (cosd(theta(i))*cosd(alpha(i))) (-cosd(theta(i))*sind(alpha(i))) ...
           (a(i)*sind(theta(i)));
16     0 sind(alpha(i)) cosd(alpha(i)) d(i);
17     0 0 0 1];
18
19 i = 2;
20 A2 = [cosd(theta(i)) (-sind(theta(i))*cosd(alpha(i))) (sind(theta(i))*sind(alpha(i))) ...
       (a(i)*cosd(theta(i)));
21     sind(theta(i)) (cosd(theta(i))*cosd(alpha(i))) (-cosd(theta(i))*sind(alpha(i))) ...
           (a(i)*sind(theta(i)));
22     0 sind(alpha(i)) cosd(alpha(i)) d(i);
23     0 0 0 1];
24
25 i = 3;
26 A3 = [cosd(theta(i)) (-sind(theta(i))*cosd(alpha(i))) (sind(theta(i))*sind(alpha(i))) ...
       (a(i)*cosd(theta(i)));
27     sind(theta(i)) (cosd(theta(i))*cosd(alpha(i))) (-cosd(theta(i))*sind(alpha(i))) ...
           (a(i)*sind(theta(i)));
28     0 sind(alpha(i)) cosd(alpha(i)) d(i);
29     0 0 0 1];
30
31 T30 = A1*A2*A3;
32
33 % Export to latex
34 latex(T30)
```

This MATLAB script gives us the following T_3^0 matrix, which is the homogeneous transformation for the end-effector with respect to the base frame.

$$\begin{bmatrix} \cos q_1 \cos q_2 - \sin q_1 \sin q_2 & -\cos q_1 \sin q_2 - \cos q_2 \sin q_1 & 0 & P_2 \cos q_1 + P_3 \cos q_1 \cos q_2 - P_3 \sin q_1 \sin q_2 \\ \cos q_1 \sin q_2 + \cos q_2 \sin q_1 & \cos q_1 \cos q_2 - \sin q_1 \sin q_2 & 0 & P_2 \sin q_1 + P_3 \cos q_1 \sin q_2 + P_3 \cos q_2 \sin q_1 \\ 0 & 0 & 1 & P_1 + q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ROS Implementation of Forward Kinematics

For implementing the ROS portion of forward kinematics, we lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Purus semper eget duis at tellus at. Ac turpis egestas integer eget aliquet nibh. Aliquam etiam erat velit scelerisque in. Ac turpis egestas sed tempus urna et pharetra pharetra. Faucibus in ornare quam viverra. Libero id faucibus nisl tincidunt eget.

Aenean pharetra magna ac placerat vestibulum lectus mauris. Consequat nisl vel pretium lectus quam id leo. Lorem sed risus ultricies tristique nulla aliquet enim tortor at. Ligula ullamcorper malesuada proin libero nunc.

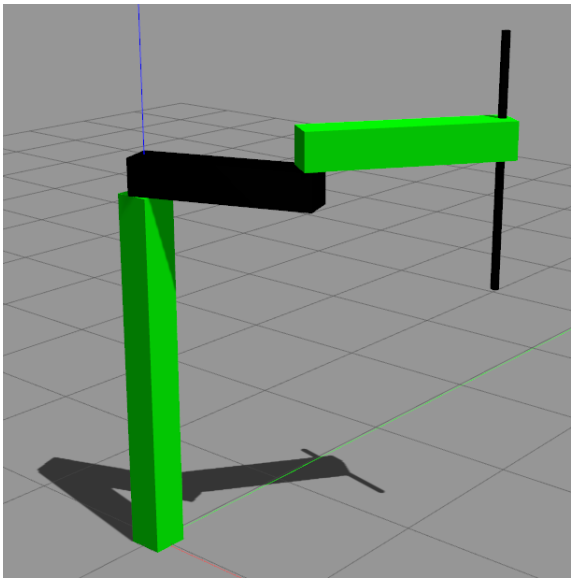
Gazebo Screenshots with ROS Output

Pose 1

First we moved our SCARA robot by changing the joint values to $q_1 = 20°$, $q_2 = 35°$, $q_3 = 0.5$. We did this by executing the following command

```
1          ros2 topic pub --once /forward_position_controller/commands ...
               std_msgs/msg/Float64MultiArray "{data: [0.349066 0.610865 0.5]}"
```

Next, we launced our forward kinematics calculation ROS node. The screenshots obtained are shown below.



(a) Gazebo robot screenshot

```
geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=1.51
32718606327764, y=1.1611691647983067, z=2.5139999999824482), or
ientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.45275
95280132489, w=0.8916326652793851))
geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=1.51
32718606327766, y=1.1611691647983071, z=2.5139999999824822), o
rientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.4527
5952801324854, w=0.8916326652793852))
geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=1.51
3271860632775, y=1.1611691647983309, z=2.5139999999824482), orie
ntation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.4527595
280132493, w=0.8916326652793851))
geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=1.51
32718606327735, y=1.1611691647983098, z=2.5139999999824827), o
rientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.4527
595280132499, w=0.8916326652793846))
geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=1.51
32718606327742, y=1.1611691647983093, z=2.5139999999824482), or
ientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.45275
95280132497, w=0.8916326652793847))
geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=1.51
32718606327766, y=1.1611691647983071, z=2.5139999999824822), o
rientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.4527
5952801324854, w=0.8916326652793852))
geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=1.51
3271860632776, y=1.1611691647983076, z=2.5139999999824827), or
ientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.45275
9528013249, w=0.8916326652793851))
geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=1.51
3271860632777, y=1.1611691647983062, z=2.5139999999824822), or
ientation=geometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.45275
952801324854, w=0.8916326652793852))
```

(b) Terminal output screenshot

Figure 6: Pose 1 Screenshots

Question 2 continued on next page...

Pose 2

We moved our SCARA robot by setting the joint values to $q_1 = 72°$, $q_2 = 28°$, $q_3 = 1.05$. We did this by executing the following command

Pose 3

Below are screenshots for joint values $q_1 = 72°$, $q_2 = 28°$, $q_3 = 1.05$

Question 3

Inverse Kinematics for SCARA

Before implementing forward kinematics in ROS, we worked out a derivation for the inverse kinematics of the SCARA robot model.

Derviation

For the inverse kinematics, a side view and a top view were used to geometrically solve for the unknown joint values.

Top view

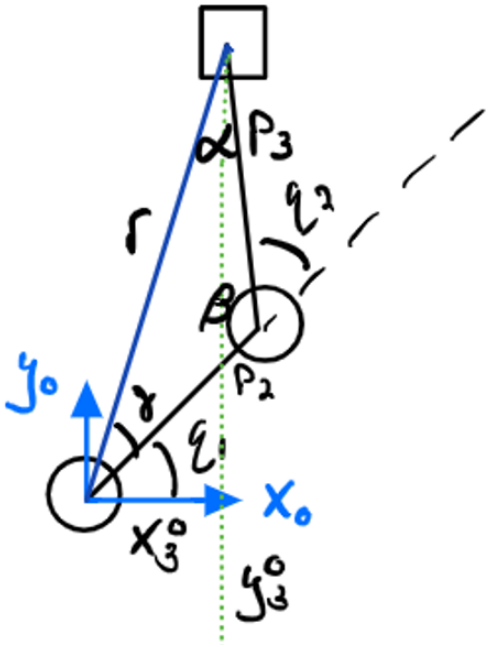


Figure 7: Top view for inverse kinematics

From the figure above, q_1 can be expressed as

$$\boxed{q_1 = \arctan\left(\frac{y_3^0}{x_3^0}\right) - \gamma}$$

Similarly, q_2 can be expressed as,

$$\boxed{q_2 = 180^\circ - \beta}$$

In order to find α and γ, we can use the law of cosines. For instance, α can be found as follows,

$$\begin{aligned} {P_2}^2 &= r^2 + {P_3}^2 - 2rP_3\cos\alpha \\ 2rP_3\cos\alpha &= r^2 + {P_3}^2 - {P_2}^2 \\ \alpha &= \arccos\left(\frac{r^2 + {P_3}^2 - {P_2}^2}{2rP_3}\right) \end{aligned}$$

Similarly, γ can be expressed as,

$$\gamma = \arccos\left(\frac{r^2 + {P_2}^2 - {P_3}^2}{2rP_3}\right)$$

Hence, β can be written as,

$$\beta = 180° - \alpha - \gamma$$

Also, we can express r in its norm,

$$r = \sqrt{(y_3^0)^2 + (x_3^0)^2}$$

Side view

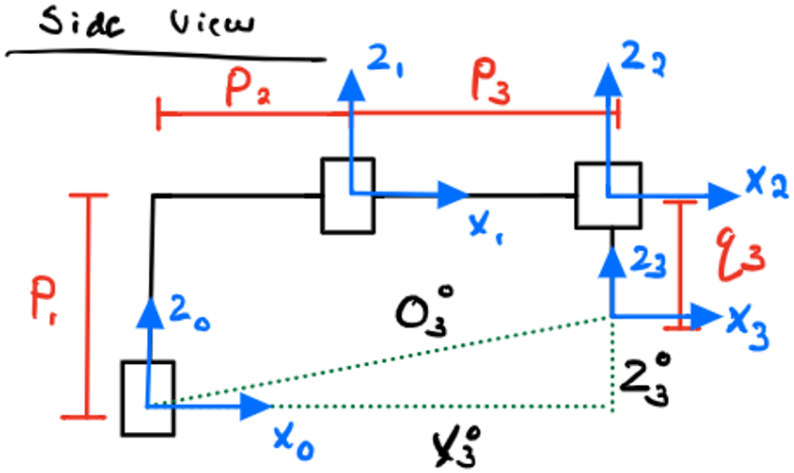


Figure 8: Side view for inverse kinematics

From the figure, we can find q_3 as follows

$$z_3^0 = P_1 - q_3$$

$$\boxed{q_3 = -(P_1 - z_3^0)}$$

Now we have expressed all the joint variables in the form of constants of the system and end-effector position coordinates. These equations for each joint value can now be solved once the end effector location is known. In the following MATLAB script, these equations are written out, and when plugging in the values found in the forward kinematics, the original joint values are found, therefore proving their accuracy.

```
% Group Project Part 1 Forward and Inverse Kinematics

clc; clear

% Forward Kinematics

% Link lengths from Gazebo
P1 = 2;
P2 = 1;
P3 = 1;

% This will come from ROS, current values are arbitrary for testing
q1 = 20;
q2 = 35;
q3 = 0.5;
```

```
16
17 % DH parameters from hand-written work
18 a = [P2 P3 0];
19 theta = [ q1 q2 0];
20 d = [P1 0 q3];
21 alpha = [ 0 0 0];
22
23 % A10, A21, and A32 transformation matrices
24 i = 1;
25 A1 = [cosd(theta(i)) (-sind(theta(i))*cosd(alpha(i))) (sind(theta(i))*sind(alpha(i))) ...
       (a(i)*cosd(theta(i)));
26     sind(theta(i)) (cosd(theta(i))*cosd(alpha(i))) (-cosd(theta(i))*sind(alpha(i))) ...
           (a(i)*sind(theta(i)));
27     0 sind(alpha(i)) cosd(alpha(i)) d(i);
28     0 0 0 1];
29
30 i = 2;
31 A2 = [cosd(theta(i)) (-sind(theta(i))*cosd(alpha(i))) (sind(theta(i))*sind(alpha(i))) ...
       (a(i)*cosd(theta(i)));
32     sind(theta(i)) (cosd(theta(i))*cosd(alpha(i))) (-cosd(theta(i))*sind(alpha(i))) ...
           (a(i)*sind(theta(i)));
33     0 sind(alpha(i)) cosd(alpha(i)) d(i);
34     0 0 0 1];
35
36 i = 3;
37 A3 = [cosd(theta(i)) (-sind(theta(i))*cosd(alpha(i))) (sind(theta(i))*sind(alpha(i))) ...
       (a(i)*cosd(theta(i)));
38     sind(theta(i)) (cosd(theta(i))*cosd(alpha(i))) (-cosd(theta(i))*sind(alpha(i))) ...
           (a(i)*sind(theta(i)));
39     0 sind(alpha(i)) cosd(alpha(i)) d(i);
40     0 0 0 1];
41
42 % End effector pose
43 T30 = A1*A2*A3;
44 end_effector_pose = T30 % display whole end-effector T matrix
45 end_effector = T30(1:3,4); % use only position part of T
46
47 % Inverse Kinematics
48
49 %These will come from the forward kinematics
50 x30 = end_effector(1);
51 y30 = end_effector(2);
52 z30 = end_effector(3);
53
54 q3 = z30 - P1;
55
56 r = sqrt((y30^2) + (x30^2));
57
58 % Needed angles from law of cosines
59 alpha = acos(((r^2) + (P3^2) - (P2^2))/(2*r*P3));
60 gamma = acos(((P2^2)+(r^2)-(P3^2))/(2*P2*r));
61 beta = 180 - alpha - gamma;
62
63 q2 = 180 - beta;
64 q1 = atan(y30/x30)-gamma;
65
66 % Calculated joint values
67 joint_angles = [(q1/(pi/180)); (q2/(pi/180)); q3]
```

Both the forward and inverse kinematics are now solved and ready to be plugged into ROS.

ROS Implementation of Inverse Kinematics

For implementing the ROS portion of forward kinematics, we lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Purus semper eget duis at tellus at. Ac turpis egestas integer eget aliquet nibh. Aliquam etiam erat velit scelerisque in. Ac turpis egestas sed tempus urna et pharetra pharetra. Faucibus in ornare quam viverra. Libero id faucibus nisl tincidunt eget.

Aenean pharetra magna ac placerat vestibulum lectus mauris. Consequat nisl vel pretium lectus quam id leo. Lorem sed risus ultricies tristique nulla aliquet enim tortor at. Ligula ullamcorper malesuada proin libero nunc.

Terminal Screenshots of ROS Output

Screenshot 1

Screenshot 2

Screenshot 3