

# **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.

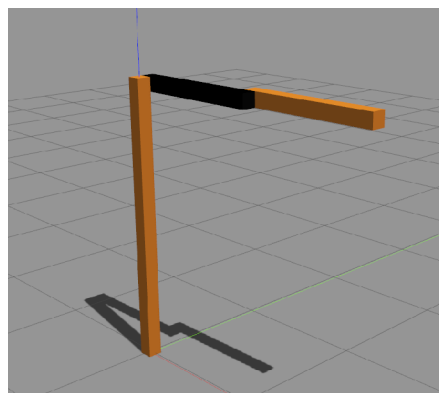


Figure 2: Initial RRBot in Gazebo

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.

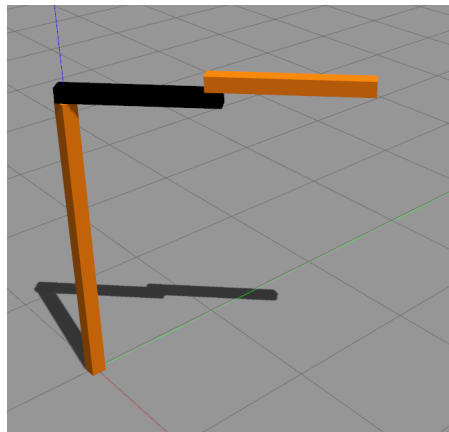


Figure 3: RRBot with top-joints

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.

```

1      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.

```

1      <!-- Tool Joint -->
2      <joint name="{prefix}tool_joint" type="prismatic">
3          <parent link="{prefix}link2"/>
4          <child link="{prefix}tool.link" />
5          <!-- Set limits of prismatic joint -->
6          <limit lower="{prismatic_offset * -1}" upper="1.1" effort="1000" velocity="1"/>
7          <origin xyz="{prismatic_offset - height4} 0 {height3 - axel_offset*2}" rpy="0 ...
          1.5708 0" />
8          <axis xyz="0 0 1"/>
9          <dynamics damping="0.7"/>
10     </joint>
11
12     <!-- Tool Link -->
13     <link name="{prefix}tool.link">
14         <collision>
15             <origin xyz="0 0 {height4/2 - axel_offset}" rpy="0 0 0"/>
16             <geometry>
17                 <box size="{prismatic_width} {prismatic_width} {height4}"/>
18             </geometry>
19         </collision>
20
21         <visual>
22             <origin xyz="0 0 {height4/2 - axel_offset}" rpy="0 0 0"/>
23             <geometry>
24                 <box size="{prismatic_width} {prismatic_width} {height4}"/>
25             </geometry>
26         </visual>
27
28         <inertial>
29             <origin xyz="0 0 {height4/2 - axel_offset}" rpy="0 0 0"/>
30             <mass value="{mass}"/>
31             <inertia
32                 ixx="{mass / 12.0 * (prismatic_width*prismatic_width + height4*height4)}" ...
33                 ixy="0.0" ixz="0.0"
34                 iyy="{mass / 12.0 * (height4*height4 + prismatic_width*prismatic_width)}" ...
35                 iyz="0.0"
36                 izz="{mass / 12.0 * (prismatic_width*prismatic_width + ...
37                     prismatic_width*prismatic_width)}"/>
38             </inertial>
39     </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 de-

fine 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.

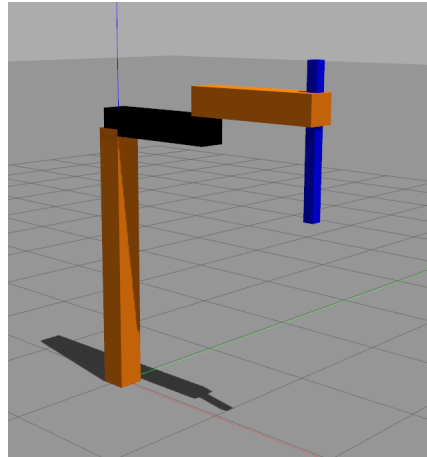


Figure 4: SCARA configuration without finishing touches

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.

#### Derivation

The image below shows the frame assignments for the robot.

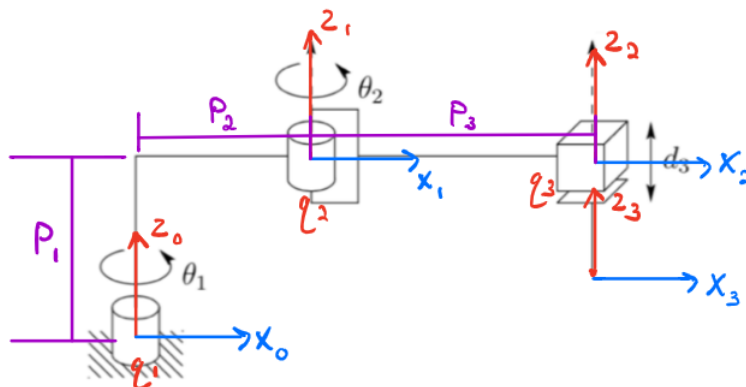


Figure 5: Frame assignments for forward kinematics

Based on this, we can form the DH table as shown below.

Link	$\alpha_i$	$a_i$	$\theta_i$	$d_i$
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
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

Below are screenshots for joint values  $q_1 = 30^\circ$ ,  $q_2 = 47^\circ$ ,  $q_3 = 0.7$

Pose 2

Below are screenshots for joint values  $q_1 = 72^\circ$ ,  $q_2 = 28^\circ$ ,  $q_3 = 1.05$

Pose 3

Below are screenshots for joint values  $q_1 = 72^\circ$ ,  $q_2 = 28^\circ$ ,  $q_3 = 1.05$



## 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.

#### Derivation

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

#### Top view

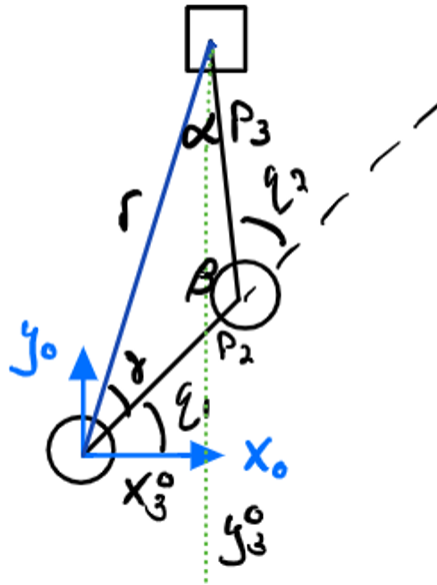


Figure 6: Top view for inverse kinematics

#### Side view

From the figure, we can see that

$$z_3^0 = p_1 - q_3$$

$$q_3 = -(p_1 - z_3^0)$$

From here, the law of cosines is used to obtain unknown lengths of formed triangles and necessary angles. After this, the equations for each joint value were formed and will be solved once the end effector location is known. In the Matlab script, these values are written out and when plugging in the values found in the forward kinematics, the original joint values are found therefore proving their accuracy. Both the forward and inverse kinematics are now solved and ready to be plugged into ROS.

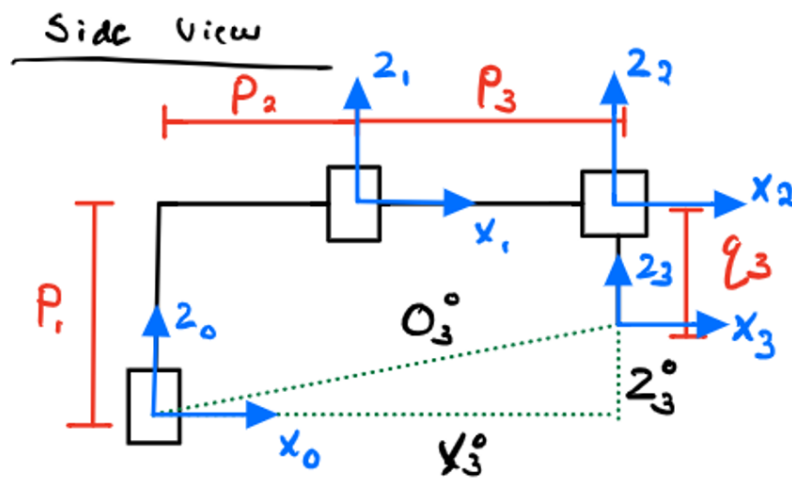


Figure 7: Side view for inverse kinematics