Final Project Writeup

ME 555: Intro to Robotics

Shuxiang (Mike) Cui

Jake Mann

Kent Yamamoto

Changxin Yu

Friday, December 9, 2022

Project Description

This project focused on motion planning and code implementation on the Universal Robotics 5e (UR5e) robot. It intended to simulate the process of mixing liquids in a beaker, with three cups filled with small beads of the primary colors and a fourth empty cup designated for collective pouring and mixing. The program architecture involved selecting two colors to mix, picking up the respective cups, pouring them into the fourth beaker, and lightly swirling the final beaker to simulate liquid mixing.

To prepare code for implementation on a physical system, a variety of virtual models were developed. An important first step was taking measurements of the physical system and designing a cup compatible with the Robotiq gripper. Importing the .stl file into Gazebo allowed for an accurate model of the real-world environment, including the height of both the mounted robot and the table containing the cups. Note that all four cups were rotated 90 degrees clockwise around their z-axis during real-world testing. This initial environment, coupled with preliminary path planning in Rviz, allowed the team to simulate the task before jumping straight into the physical implementation. A picture of the virtual environment is shown below.

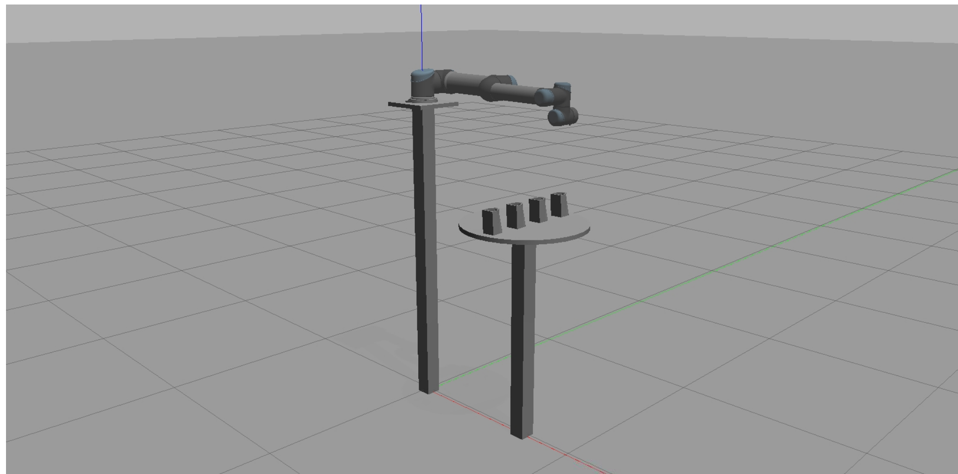

Figure 1: Simulation Developed for Project

To prepare the cups for 'mixing' in the real world, the first three cups were partially filled with small beads to simulate liquids. The fourth cup was left empty to serve as the mixing container. The four cups were then evenly distributed across the table. During the mixing task, the UR5e robot moved to the front of each cup to grasp and move the cup without collision. The

still images of this process are shown below (Figures 2-5). In addition, the four cups are properly aligned with small, 3D-printed lips that do not interfere with the gripper hand.

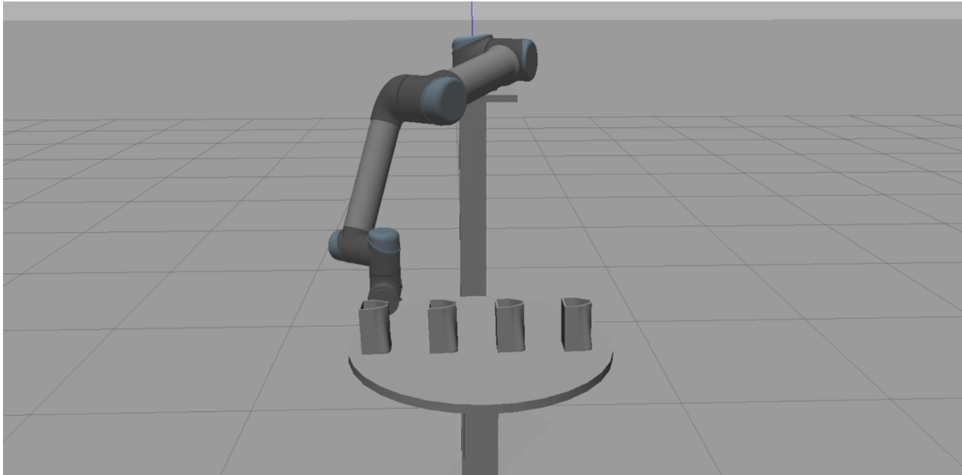

Figure 2: UR5e with end-effector (EE) in the front of the 1st cup

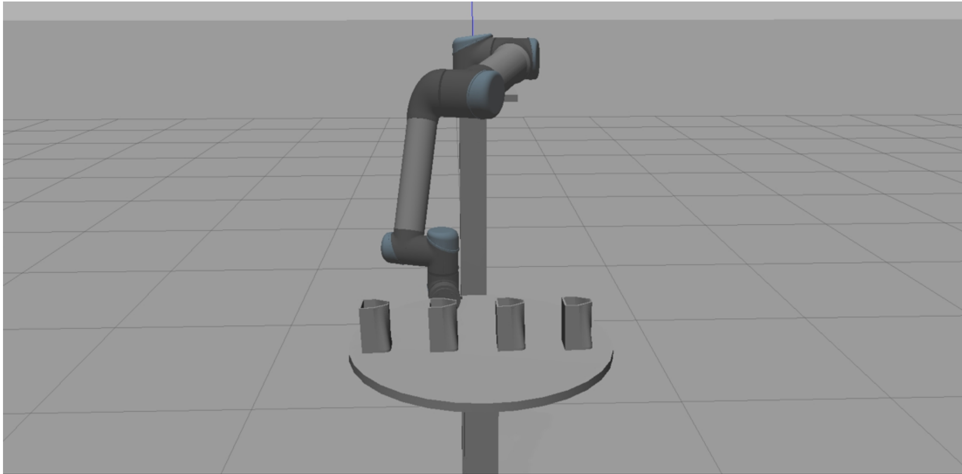

Figure 3: UR5e with EE in the front of the 2nd cup

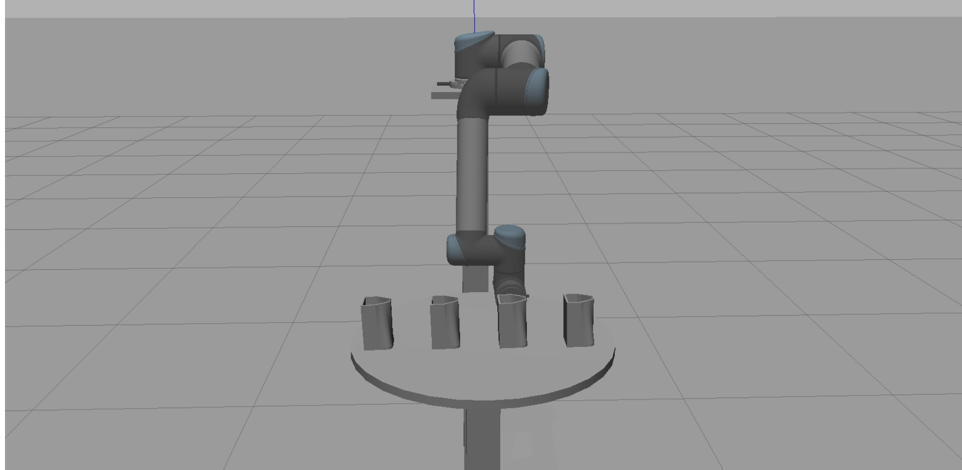

Figure 4: UR5e with EE in the front of the 3rd cup

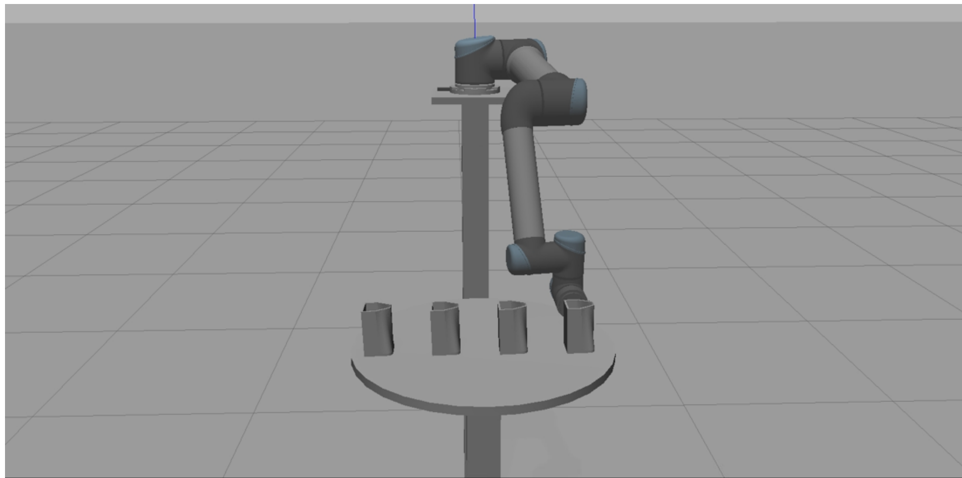

Figure 5: UR5e with EE in the front of the 4th cup

Individual Efforts

Our team met every Monday from 6 – 7 pm to discuss the progress of the project, and we tackled the final project by completing tasks in our field of interest and specialty while also learning new skills. All members of the team contributed to write both the project write-up and the research paper and will contribute to the physical implementation next week.

Mike led the development of the simulated environment, integrated the UR5 model into the new environment, and helped with the overall simulation. Jake designed and 3D-printed the cups tailored for easy grasping and the alignment rings for the physical setup. Kent led the research project component and assisted with the Robotiq gripper code. Changxin headed the motion planning and coding and assisted in the simulated environment setup.

Component Description and Code Walkthrough

The Gazebo simulation environment is mostly set up under SDF and URDF files. The motion planning part, on the other hand, is done in Python. The system diagram of the communication channels is created to understand how the components of the project interact with each other:

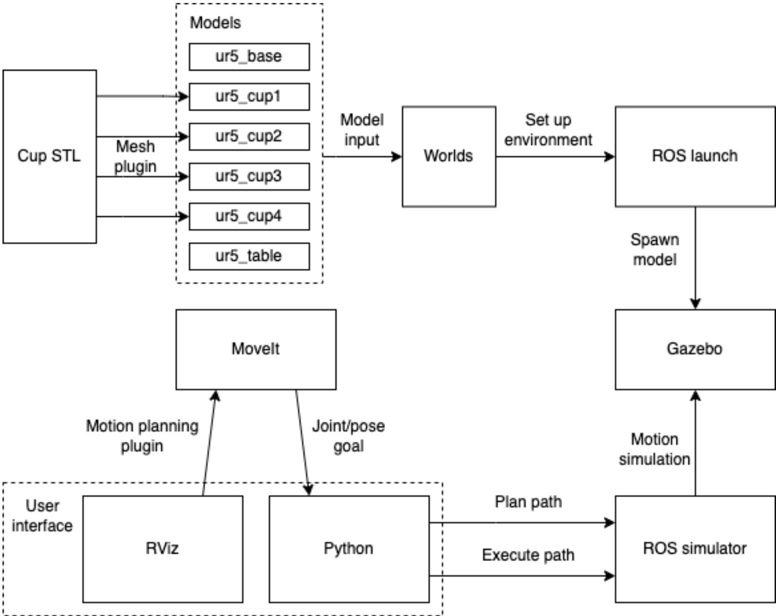

Figure 6: System Diagram of Proposed System

To reiterate, Figure 1 above simulates the workspace and objects involved in our experiment. As the picture implies, the UR5e robot was located on a basis with a height of 44 inches; there were

also four cups specifically designed for this project sitting on top of a table which is composed of a circular top plate with a 1-foot radius and a leg 2 feet in height. This environment setup aimed to imitate the experimental environment in-lab. The gripper, in the simulation, was not attached to robot's end-effector but was already implemented on the physical arm. Notable sections of simulation coding are shown below.

Initial UR5e Pose

To mimic the actual UR5e robot position in the lab, the simulated robot is set to spawn a little higher than the original position in Gazebo. This is achieved by setting the value in the z-axis under the ur5.launch file as shown below.

```
  <!-- push robot_description to factory and spawn robot in gazebo -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model" args="-urdf -param robot_description -model
robot -z 1.594" respawn="false" output="screen" />
```

Figure 7: Robot Position Code Snippet

Design of Models

To establish a well-crafted virtual experimental environment, a robot basis, a circular table, and four cups were designed. A *model.sdf* file was created for each model separately. The examples below demonstrate a general idea of the code on how to create the circular table as well as the cup. Note that the cup model is created with a separate mesh file due to the complexity of its shape.

```
<sdf version="1.6">
  <model name="ur5_table">
    <static>true</static>

    <link name="top_plate">
      <pose>0 0 1.08 0 0 0</pose>
      <collision name="top_plate_collision">
        <geometry>
          <cylinder>
            <radius>0.3048</radius>
            <length>0.02</length>
          </cylinder>
        </geometry>
        <surface>
          <contact>
            <collide_bitmask>0x01</collide_bitmask>
          </contact>
        </surface>
      </collision>
      <visual name="top_plate_visual">
        <geometry>
          <cylinder>
            <radius>0.3048</radius>
            <length>0.02</length>
          </cylinder>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
          </script>
        </material>
      </visual>
    </link>
```

Figure 8: Table Properties Code Snippet

```
    <link name="link">
      <pose>0 0 1.1 0 0 0</pose>
      <collision name="collision">
        <geometry>
          <mesh>
            <uri>model://ur5_cup1/meshes/Cup.stl</uri>
            <scale>0.001 0.001 0.001</scale>
          </mesh>
        </geometry>
        <surface>
          <contact>
            <collide_bitmask>0x01</collide_bitmask>
          </contact>
        </surface>
      </collision>
      <visual name="visual">
        <geometry>
          <mesh>
            <uri>model://ur5_cup1/meshes/Cup.stl</uri>
            <scale>0.001 0.001 0.001</scale>
          </mesh>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
          </script>
        </material>
      </visual>
    </link>
```

Figure 9: Cup Properties Code Snippet

Position of Models

Apart from the UR5e robot in this simulation, there other objects required a positional setup as well, including the basis under the robot, the circular table, and the cups. This was done

by adjusting parameters under the pose attribute accordingly. The file associated with this modification is *ur_setup.world.* Note that the numbers in the picture below are flexible depending on the team's simulation progress.

```
<!-- The robot base -->
<model name="ur5_base">
  <include>
    <uri>model://ur5_base</uri>
    <pose>0 0 0 0 0 0</pose>
  </include>
</model>

<model name="ur5_table">
  <include>
    <uri>model://ur5_table</uri>
    <pose>0.5 0 0 0 0 0</pose>
  </include>
</model>

<model name="ur5_cup1">
  <include>
    <uri>model://ur5_cup1</uri>
    <pose>0.5 -0.2686 0 0 0 0</pose>
  </include>
</model>
```

Figure 10: Model Position Code Snippet

Robot Motion

The robot motion planning was divided into two parts: UR5e motion planning and gripper control. Since the control of the gripper was difficult to integrate with the robot motion planning in the same script, these two parts were separated in different scripts and will be executed at the same time in the real-world to accomplish solution mixing.

The robot is expected to complete the following steps: move to one of the cups, pick up the cup, move to another cup diagonally above (to pour the liquid into the new cup), move back to put the previous cup down. A Python script is coded to achieve this process, which can be found in Appendix B. In the script, the major code is integrated into the class "Solution_mixing" containing eleven methods. They are "__init__", "init_orientation_constraints", "move_to_home", "move_to_key_positions_from_home", "move_to_key_positions_with_cup", "move_up_or_down", "move_before_pouring", "pour", "mix", "pouring_task" and "mixing_task". The function of each method is elaborated below.

The "__init__" method performs system initialization, including initializing moveit_commander and rospy node, instantiating a MoveGroupCommander object, enabling motion replanning, setting error tolerance, getting end effector link, setting the positions of four

cups, setting the orientation parallel to the ground (for holding a cup), and setting the useful joint indexes for “pour” and “mix” methods. The “init_orientation_constraints” method (Figure 7) creates a constraint object with the orientation tolerances of the x and y axes equaling 23 degrees. When the gripper holds a cup, the constraint object will be called to avoid spilling liquid.

```
# define orientation constraints with MoveIt
def init_orientation_constraints(self, pose):

    self.constraints = Constraints()
    self.constraints.name = "orientationConstraints"
    orientation_constraint = OrientationConstraint()
    orientation_constraint.link_name = self.EE_link
    orientation_constraint.orientation = pose.orientation
    orientation_constraint.absolute_x_axis_tolerance = 0.4
    orientation_constraint.absolute_y_axis_tolerance = 0.4
    orientation_constraint.absolute_z_axis_tolerance = pi
    orientation_constraint.weight = 1

    self.constraints.orientation_constraints.append(orientation_constraint)
```

Figure 11. Code of “init_orientation_constraints” method

The "move_to_home" method sets a home pose, then sets it as the target and executes the plan. The method "move_to_key_positions_from_home" plans and moves the end effector to the location of one of four cups and the index of the cup will be assigned. The method "move_to_key_positions_with_cup" is similar to "move_to_key_positions_from_home" but has orientation constraints. The "move_up_or_down" method sets the target pose that only changes the z-value from the current pose and the command “up” or “down” will be assigned. The "move_before_pouring" method moves the current pose diagonally upwards to reach an appropriate position for pouring liquid. The “pour” method rotates the fourth joint 100 degrees to pour liquid and then return. Finally, the “mix” method rotates the sixth joint to shake the cup.

The "pouring_task" method combines the above methods to perform a complete pouring task, including moving to cup A, picking cup A up, moving to cup B with cup A, pouring liquid from cup A to cup B, and putting cup A back. The index of cup A and cup B can be assigned by the programmer. The "mixing_task" method includes moving to a cup, picking up the cup for shaking, and putting down the cup.

The main function instantiates a "Solution_mixing" instance and then calls the "pouring_task" and "mixing_task" methods several times to carry out the autonomous solution mixing.

Human Integration

As the project involved assessing the feasibility of robotic mixing of hazardous chemical solutions to protect experimenters from casualties, the primary user that would interface with this system are the experimenters. This means that our system would be semi-autonomous, with the experimenters physically placing the solutions where the robot arm can reach, then leaving the environment. They will then utilize a graphical user interface (GUI) to remotely control and input what solutions need mixing in which beaker.

Challenges

The team encountered many challenges over the course of the project. One of them was setting up the environment for the simulation. This was a significant part of the project since the simulation and workspace analysis are heavily based on the environmental setup. To tackle the problem, the team researched similar open-source environment setups. By interpolating from many examples, a basic understanding of the environment code was established. Further online investigations helped the team polish the setup to its final setup.

Another challenge we faced was the singularity of the UR5e during robot motion planning. To achieve the smooth movement of the robot, the horizontal and vertical distances between the table and the robot base were tweaked. To perform both liquid-pouring and mixing, at the beginning, the sixth joint of the UR5e was chosen to tilt the cup by rotation. However, the simulation results showed that when the robot holds a cup, the sixth joint can only rotate at small degrees, which meant that only the mixing task could be carried out (since the pouring task requires a large rotation about the wrist). To solve the issue, the team tried other approaches and finally completed the pouring task by changing the fourth joint value.

Future Work

As of 12/09, the team could not start the physical implementation. The first component of future work the team proposes is to implement the simulation in real-life with the Duke Robotics UR5e robot arm using small dice or popcorn as a surrogate for liquids. After showing that the

surrogate material works robustly, the team would like to integrate actual liquids into the system. Furthermore, if used in an industrial context, increasing the mixing speed would be beneficial as it would optimize the time needed to complete the task in large batches.

Takeaways

- Simulation is integral before physical implementation
    - Through simulating the proposed task, the team was able to understand the small details that would prevent the physical implementation from working. This includes initializing the robot at a singularity, causing the simulation to crash. The team believes that the simulation made for this project is synonymous with creating CAD models of mechanical assemblies before buying the components to ensure the components work in simulation first.
- Good team cooperation results in strong results
    - This is not an easy project due to the breadth of the topics it covered. To complete it in such a limited time, great cooperation skills are integral for a thorough final product. Identifying each other's strengths, setting a reasonable objective for each stage, and communicating regularly are always helpful for a challenge like this project.

Appendix A. URDF and SDF code

ur5.launch

```
<?xml version="1.0"?>
<launch>
  <!-- Export env variable so that gazebo finds our models -->
  <env name="GAZEBO_MODEL_PATH"
       value="$(find ur_gazebo)/models:$(optenv GAZEBO_MODEL_PATH)" />
  <arg name="limited" default="false"  doc="If true, limits joint range [-
PI, PI] on all joints." />
  <arg name="paused" default="false" doc="Starts gazebo in paused mode" />
  <arg name="gui" default="true" doc="Starts gazebo gui" />
  <arg name="world_name" default="$(find ur_gazebo)/worlds/ur_setup.world"
/>

  <!-- startup simulated world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(arg world_name)"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg gui)"/>
  </include>

  <!-- send robot urdf to param server -->
  <include file="$(find ur_description)/launch/ur5_upload.launch">
    <arg name="limited" value="$(arg limited)"/>
  </include>

  <!-- push robot_description to factory and spawn robot in gazebo -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model"
args="-urdf -param robot_description -model robot -z 1.594"
respawn="false" output="screen" />

  <include file="$(find ur_gazebo)/launch/controller_utils.launch"/>

  <!-- start this controller -->
  <rosparam file="$(find ur_gazebo)/controller/arm_controller_ur5.yaml"
command="load"/>
  <node name="arm_controller_spawner" pkg="controller_manager"
type="controller_manager" args="spawn arm_controller" respawn="false"
output="screen"/>

  <!-- load other controllers -->
  <node name="ros_control_controller_manager" pkg="controller_manager"
type="controller_manager" respawn="false" output="screen" args="load
joint_group_position_controller" />

</launch>
```

model.sdf for robot base

```
<?xml version="1.0"?>

<sdf version="1.6">
  <model name="ur5_base">
    <static>true</static>

    <link name="top_plate">
      <pose>0 0 1.58 0 0 0</pose>
      <collision name="top_plate_collision">
        <geometry>
          <box>
            <size>0.25 0.25 0.02</size>
          </box>
        </geometry>
        <surface>
          <contact>
            <collide_bitmask>0x01</collide_bitmask>
          </contact>
        </surface>
      </collision>
      <visual name="top_plate_visual">
        <geometry>
          <box>
            <size>0.25 0.25 0.02</size>
          </box>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
          </script>
        </material>
      </visual>
    </link>
```

```
    <link name="leg1">
      <pose>0 0 0.785 0 0 0</pose>
      <collision name="leg1_collision">
        <geometry>
          <box>
            <size>0.08 0.08 1.57</size>
          </box>
        </geometry>
      </collision>
      <visual name="leg1_visual">
        <geometry>
          <box>
            <size>0.08 0.08 1.57</size>
          </box>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
          </script>
        </material>
      </visual>
    </link>

  </model>
</sdf>
```

model.sdf for cup1-4

```
<?xml version="1.0"?>

<sdf version="1.6">
  <model name="ur5_cup1">
    <static>false</static>

    <link name="link">
      <pose>0 0 1.1 0 0 0</pose>
      <collision name="collision">
        <geometry>
          <mesh>
            <uri>model://ur5_cup1/meshes/Cup.stl</uri>
            <scale>0.001 0.001 0.001</scale>
          </mesh>
        </geometry>
        <surface>
          <contact>
            <collide_bitmask>0x01</collide_bitmask>
          </contact>
        </surface>
      </collision>
      <visual name="visual">
        <geometry>
          <mesh>
            <uri>model://ur5_cup1/meshes/Cup.stl</uri>
            <scale>0.001 0.001 0.001</scale>
          </mesh>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
```

```
          </script>
        </material>
      </visual>
    </link>

  </model>
</sdf>
```

model.sdf for table

```
<?xml version="1.0"?>

<sdf version="1.6">
  <model name="ur5_table">
    <static>true</static>

    <link name="top_plate">
      <pose>0 0 1.08 0 0 0</pose>
      <collision name="top_plate_collision">
        <geometry>
          <cylinder>
            <radius>0.3048</radius>
            <length>0.02</length>
          </cylinder>
        </geometry>
        <surface>
          <contact>
            <collide_bitmask>0x01</collide_bitmask>
          </contact>
        </surface>
      </collision>
      <visual name="top_plate_visual">
        <geometry>
          <cylinder>
            <radius>0.3048</radius>
            <length>0.02</length>
          </cylinder>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
```

```
          </script>
        </material>
      </visual>
    </link>
    <link name="leg1">
      <pose>0 0 0.535 0 0 0</pose>
      <collision name="leg1_collision">
        <geometry>
          <box>
            <size>0.08 0.08 1.07</size>
          </box>
        </geometry>
      </collision>
      <visual name="leg1_visual">
        <geometry>
          <box>
            <size>0.08 0.08 1.07</size>
          </box>
        </geometry>
        <material>
          <script>
            <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
          </script>
        </material>
      </visual>
    </link>

  </model>
</sdf>
```

ur_setup.world

```
<?xml version="1.0" ?>

<sdf version="1.6">

  <world name="ur5_robot_base">

    <gravity>0 0 -9.81</gravity>
    <physics name="default_physics" default="0" type="ode">
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1</real_time_factor>
      <real_time_update_rate>1000</real_time_update_rate>
    </physics>

    <scene>
      <ambient>0.4 0.4 0.4 1</ambient>
      <background>0.7 0.7 0.7 1</background>
      <shadows>1</shadows>
    </scene>

    <!-- Light Source -->
    <include>
      <uri>model://sun</uri>
    </include>

    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
      <pose>0 0 0 0 0 0</pose>
    </include>

    <!-- The robot base -->
    <model name="ur5_base">
      <include>
        <uri>model://ur5_base</uri>
        <pose>0 0 0 0 0 0</pose>
      </include>
    </model>

    <model name="ur5_table">
      <include>
        <uri>model://ur5_table</uri>
        <pose>0.5 0 0 0 0 0</pose>
      </include>
    </model>

    <model name="ur5_cup1">
      <include>
```

```
        <uri>model://ur5_cup1</uri>
        <pose>0.5 -0.2686 0 0 0 0</pose>
      </include>
    </model>

    <model name="ur5_cup2">
      <include>
        <uri>model://ur5_cup2</uri>
        <pose>0.5 -0.1162 0 0 0 0</pose>
      </include>
    </model>

    <model name="ur5_cup3">
      <include>
        <uri>model://ur5_cup3</uri>
        <pose>0.5 0.0362 0 0 0 0</pose>
      </include>
    </model>

    <model name="ur5_cup4">
      <include>
        <uri>model://ur5_cup4</uri>
        <pose>0.5 0.1886 0 0 0 0</pose>
      </include>
    </model>



  </world>
</sdf>
```

Appendix B. Python code of UR5e motion planning

```
1.  import rospy, sys
2.  import moveit_commander
3.  import tf
4.  from geometry_msgs.msg import PoseStamped
5.  from math import pi
6.  from moveit_msgs.msg import Constraints, OrientationConstraint
7.  
8.  class Solution_mixing:
9.      # initialization
10.     def __init__(self):
11. 
12.         # initialize moveit_commander and a rospy node
13.         moveit_commander.roscpp_initialize(sys.argv)
14.         rospy.init_node('group_project', anonymous=True)
15. 
16.         # instantiate a MoveGroupCommander object
17.         self.group = moveit_commander.MoveGroupCommander('manipulator')
18. 
19.         # enable replanning
20.         self.group.allow_replanning(True)
21. 
22.         # set error tolerance
23.         self.group.set_goal_position_tolerance(0.01)
24.         self.group.set_goal_orientation_tolerance(0.05)
25. 
26.         # get EE link
27.         self.EE_link = self.group.get_end_effector_link()
28. 
29.         # set four key positions
30.         self.key_positions = [[0.65, -0.23685, -0.26],
31.                               [0.65, -0.08445, -0.26],
32.                               [0.65, 0.06795, -0.26],
33.                               [0.65, 0.22035, -0.26]]
34. 
35.         # set the orientation for holding a cup
36.         orientations = tf.transformations.quaternion_from_euler(-pi/2, 0.0, 0.0)
37.         self.key_orientations = orientations
38. 
39.         # whichever joint interfaces with the gripper
40.         self.wrist_pour = 3
41.         self.wrist_mix = 5
42. 
43. 
44.     # define orientation constraints with MoveIt
45.     def init_orientation_constraints(self, pose):
46. 
47.         self.constraints = Constraints()
48.         self.constraints.name = "orientationConstraints"
49.         orientation_constraint = OrientationConstraint()
```

```
50.         orientation_constraint.link_name = self.EE_link
51.         orientation_constraint.orientation = pose.orientation
52.         orientation_constraint.absolute_x_axis_tolerance = 0.4
53.         orientation_constraint.absolute_y_axis_tolerance = 0.4
54.         orientation_constraint.absolute_z_axis_tolerance = pi
55.         orientation_constraint.weight = 1
56.
57.         self.constraints.orientation_constraints.append(orientation_constraint)
58.
59.
60.     # move to starting joint positions
61.     def move_to_home(self):
62.         home_positions = [0, 0, 1.7, 0, 0, 0]
63.         self.group.set_joint_value_target(home_positions)
64.         self.group.go()
65.         rospy.sleep(1)
66.
67.
68.     # move to one of four key positions from home
69.     def move_to_key_positions_from_home(self, position):
70.
71.         target_pose = PoseStamped().pose
72.         target_pose.position.x = self.key_positions[position-1][0]
73.         target_pose.position.y = self.key_positions[position-1][1]
74.         target_pose.position.z = self.key_positions[position-1][2]
75.         target_pose.orientation.x = self.key_orientations[0]
76.         target_pose.orientation.y = self.key_orientations[1]
77.         target_pose.orientation.z = self.key_orientations[2]
78.         target_pose.orientation.w = self.key_orientations[3]
79.
80.         # set current condition as beginning
81.         self.group.set_start_state_to_current_state()
82.
83.         # set goal pose
84.         self.group.set_pose_target(target_pose, self.EE_link)
85.
86.         # plan and execute the motion
87.         plan_success, traj, planning_time, error_code = self.group.plan()
88.         self.group.execute(traj, wait=True)
89.         rospy.sleep(1)
90.
91.
92.     # move to one of four key positions with cup
93.     def move_to_key_positions_with_cup(self, position):
94.
95.         target_pose = PoseStamped().pose
96.         target_pose.position.x = self.key_positions[position-1][0]
97.         target_pose.position.y = self.key_positions[position-1][1]
98.         target_pose.position.z = self.key_positions[position-1][2]
99.         target_pose.orientation.x = self.key_orientations[0]
100.        target_pose.orientation.y = self.key_orientations[1]
101.        target_pose.orientation.z = self.key_orientations[2]
102.        target_pose.orientation.w = self.key_orientations[3]
103.
104.        # set current condition as beginning
105.        self.group.set_start_state_to_current_state()
106.
```

```
107.         # set goal pose
108.         self.group.set_pose_target(target_pose, self.EE_link)
109.
110.         # set orientation constraints
111.         self.init_orientation_constraints(target_pose)
112.         self.group.set_path_constraints(self.constraints)
113.
114.         # plan and execute the motion
115.         plan_success, traj, planning_time, error_code = self.group.plan()
116.         self.group.execute(traj, wait=True)
117.         rospy.sleep(1)
118.
119.
120.     # move on z axis, up or down
121.     def move_up_or_down(self, command):
122.         if command == 'up': z_value = -0.26
123.         elif command == 'down': z_value = -0.43
124.         else: raise ValueError('Invalid command, command must be up or down')
125.
126.         current_pose = self.group.get_current_pose().pose
127.         target_pose = current_pose
128.         target_pose.position.z = z_value
129.
130.         # set current condition as beginning
131.         self.group.set_start_state_to_current_state()
132.
133.         # set goal pose
134.         self.group.set_pose_target(target_pose, self.EE_link)
135.
136.         # set orientation constraints
137.         self.init_orientation_constraints(target_pose)
138.         self.group.set_path_constraints(self.constraints)
139.
140.         # plan and execute the motion
141.         plan_success, traj, planning_time, error_code = self.group.plan()
142.         self.group.execute(traj, wait=True)
143.         rospy.sleep(1)
144.
145.
146.     def move_before_pouring(self):
147.
148.         current_pose = self.group.get_current_pose().pose
149.         target_pose = current_pose
150.         target_pose.position.x -= 0.05
151.         target_pose.position.z = -0.3
152.
153.         # set current condition as beginning
154.         self.group.set_start_state_to_current_state()
155.
156.         # set goal pose
157.         self.group.set_pose_target(target_pose, self.EE_link)
158.
159.         # set orientation constraints
160.         self.init_orientation_constraints(target_pose)
161.         self.group.set_path_constraints(self.constraints)
162.
163.         # plan and execute the motion
```

```
164.         plan_success, traj, planning_time, error_code = self.group.plan()
165.         self.group.execute(traj, wait=True)
166.         rospy.sleep(1)
167.
168.
169.     # pour the liquid/beads into cup
170.     def pour(self):
171.
172.         # initialize joint_goal variable by collecting starting position
173.         joint_goal = self.group.get_current_joint_values()
174.         pre_pour = joint_goal[self.wrist_pour]
175.
176.         # a little more than 90 deg (100 deg) for slow pouring
177.         joint_goal[self.wrist_pour] = pre_pour + pi/2 + pi/18
178.         self.group.set_joint_value_target(joint_goal)
179.         self.group.go(joint_goal, wait =True)
180.         rospy.sleep(3)
181.
182.         # go back to pre-pour wrist configuration
183.         joint_goal[self.wrist_pour] = pre_pour
184.         self.group.go(joint_goal, wait =True)
185.
186.
187.     # mix the mixing cup
188.     def mix(self):
189.
190.         joint_goal = self.group.get_current_joint_values()
191.         pre_mix = joint_goal[self.wrist_mix]
192.         joint_goal[self.wrist_mix] = pre_mix + pi/9 # 20 deg rotate left
193.         self.group.go(joint_goal, wait =True)
194.         rospy.sleep(1)
195.         joint_goal[self.wrist_mix] = pre_mix - pi/9 # 20 deg rotate to the right
196.         self.group.go(joint_goal, wait =True)
197.         rospy.sleep(1)
198.         joint_goal[self.wrist_mix] = pre_mix + pi/9 # 20 deg rotate to the left
199.         self.group.go(joint_goal, wait =True)
200.         rospy.sleep(1)
201.         joint_goal[self.wrist_mix] = pre_mix # go back to pre_mix joint configuration
202.         self.group.go(joint_goal, wait =True)
203.
204.
205.     # a complete pouring task
206.     # start from home, pick up cup_a, pour the water from cup_a to cup_b, put cup_a
    back
207.     def pouring_task(self, cup_a, cup_b):
208.
209.         self.move_to_home()
210.         self.move_to_key_positions_from_home(cup_a)
211.         self.move_up_or_down('down')
212.         rospy.sleep(1)
213.         self.move_up_or_down('up')
214.         self.move_to_key_positions_with_cup(cup_b)
215.         self.move_before_pouring()
216.         self.pour()
217.         self.move_to_key_positions_with_cup(cup_a)
218.         self.move_up_or_down('down')
219.         rospy.sleep(1)
```

```
220.         self.move_up_or_down('up')
221.
222.
223.     # a complete mixing task
224.     def mixing_task(self, cup_a):
225.
226.         self.move_to_home()
227.         self.move_to_key_positions_from_home(cup_a)
228.         self.move_up_or_down('down')
229.         rospy.sleep(1)
230.         self.move_up_or_down('up')
231.         self.mix()
232.         self.move_up_or_down('down')
233.         rospy.sleep(1)
234.         self.move_up_or_down('up')
235.
236.
237. if __name__ == "__main__":
238.     project = Solution_mixing()
239.     try:
240.         project.pouring_task(1, 4)
241.         project.pouring_task(2, 4)
242.         project.mixing_task(4)
243.         project.pouring_task(2, 4)
244.         project.pouring_task(3, 4)
245.         project.mixing_task(4)
246.     except rospy.ROSInterruptException:
247.         pass
```

Appendix C. MATLAB code for research project

```
%% ME 555 - Final Project
% UR5 Workspace/Singularity Analysis

close all
clear all
ur5_RBT = loadrobot("universalUR5");

%% MC for S = 0.01
figure
sVals = []; % empty vector for singularity points
wVals = []; % empty vector for workspace points
sVec = []; % empty vector for singulary percentage
tic % timer starts
num = 1000; % number of MC simulations
sCount = 0; % counter initialized for number of singularities

for i = 1:num
    config = randomConfiguration(ur5_RBT); % create random configuration

    focusT = getTransform(ur5_RBT,config,'tool0'); % get transform of random
configuration
    pVec = focusT(1:3,4); % index out position of end effector

    J = geometricJacobian(ur5_RBT,config,'tool0'); % get jacobian of same
configuration

    if abs(det(J)) >= 0 && abs(det(J)) <= 0.01 % logic if statement for
absolute value of Jacobian
        sVals = [sVals;pVec']; % concatenate position of end-effector to
singularity vector
        sCount = sCount + 1; % increase counter by 1
    else
        wVals = [wVals;pVec']; % concatenate position of end-effector if
workspace
    end
end
toc
sVec = [sVec sCount]; % concatenate number of singularities
subplot(2,2,1) % subplot of all values, both workspace and singularities
scatter3(wVals(:,1),wVals(:,2),wVals(:,3),'k')
hold on
scatter3(sVals(:,1),sVals(:,2),sVals(:,3),'r')
xlabel('[X]')
ylabel('[Y]')
ylabel('[Z]')
lgd = legend('Workspace','Singularity');
lgd.FontSize = 18;
```

```
title('Monte Carlo Simulation of UR5 Workspace and Singularities','(n =
10,000; k = 0.01)','FontSize',25)
hold off

%% MC for S = 0.005 (Same comments as above, just S is different)
sCount = 0;
tic
for i = 1:num
    config = randomConfiguration(ur5_RBT);

    focusT = getTransform(ur5_RBT,config,'tool0');
    pVec = focusT(1:3,4);

    J = geometricJacobian(ur5_RBT,config,'tool0');

    if abs(det(J)) >= 0 && abs(det(J)) <= 0.005
        sVals = [sVals;pVec'];
        sCount = sCount + 1;
    else
        wVals = [wVals;pVec'];
    end
end
toc
sVec = [sVec sCount];
subplot(2,2,2)
scatter3(wVals(:,1),wVals(:,2),wVals(:,3),'k')
hold on
scatter3(sVals(:,1),sVals(:,2),sVals(:,3),'r')
xlabel('[X]')
ylabel('[Y]')
ylabel('[Z]')
lgd = legend('Workspace','Singularity');
lgd.FontSize = 18;
title('Monte Carlo Simulation of UR5 Workspace and Singularities','(n =
10,000; k = 0.005)','FontSize',25)
hold off
%%  MC for S = 0.001 (Same comments as above, just S is different)
sCount = 0;
tic
for i = 1:num
    config = randomConfiguration(ur5_RBT);

    focusT = getTransform(ur5_RBT,config,'tool0');
    pVec = focusT(1:3,4);

    J = geometricJacobian(ur5_RBT,config,'tool0');

    if abs(det(J)) >= 0 && abs(det(J)) <= 0.001
        sVals = [sVals;pVec'];
        sCount = sCount + 1;
    else
        wVals = [wVals;pVec'];
    end
end
toc
```

```
sVec = [sVec sCount];
subplot(2,2,3)
scatter3(wVals(:,1),wVals(:,2),wVals(:,3),'k')
hold on
scatter3(sVals(:,1),sVals(:,2),sVals(:,3),'r')
xlabel('[X]')
ylabel('[Y]')
ylabel('[Z]')
lgd = legend('Workspace','Singularity');
lgd.FontSize = 18;
title('Monte Carlo Simulation of UR5 Workspace and Singularities','(n =
10,000; k = 0.001)','FontSize',25)
hold off
%% Last Subplot

subplot(2,2,4) % subplot for just singularity values at when S = 0.001
scatter3(sVals(:,1),sVals(:,2),sVals(:,3),'r')
xlabel('[X]')
ylabel('[Y]')
ylabel('[Z]')
title('3D Visualization of UR5 Singularities','(n = 10,000; k =
0.001)','FontSize',25)

%% Values recorded using tictoc function for time it takes to complete each
MC simulation
t1000 = [1.1299 0.9657 0.914461];
avgT1000 = mean(t1000); % average time for 1000 configs
t5000 = [4.8325 4.6426 4.5907];
avgT5000 = mean(t5000); % average time for 5000 configs
t10000 = [9.3437 9.1429 9.1716];
avgT10000 = mean(t10000); % average time for 10000 configs

%% Calculate percentage of singularities in all simulations
sPercent = sVec/num*100;
```