Project -7

Design and implement a sorting robotic arm to sort the objects based on the colour. The manipulator should be built from scratch. The input of the system can be locations of the objects and colour-based destination points. The system should be tested with a minimum of three different colours. The type of sensor can be the team's choice but the reason to use the sensor should be clear in the report and the presentation.

A PROJECT REPORT

Submitted by

Team – SPD Power Rangers

A DHANUSH -BL.EN.U4AIE19002

NVS PRADYUMNA – BL.EN.U4AIE19043

SATWIK REDDY S– BL.EN.U4AIE19059

for the course

19AIE213- ROBOTICS OPERATING SYSTEM & ROBOT SIMULATION

Guided and Evaluated by

Nippun Kumar A.A

Asst. Prof(SG),

Dept. of CSE,

AMRITA SCHOOL OF ENGINEERING, BANGALORE

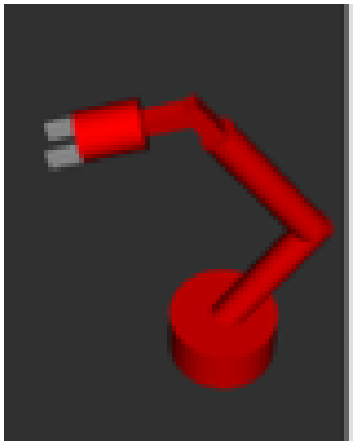
AMRITA VISHWA VIDHYAPEETHAM, BANGALORE-560 035

INTRODUCTION

In general, Mobile robots with agility, one or more robotic arms, and a gripper are known as robot manipulators. Manipulators can either be fixed or movable. Movable manipulators can walk about in an area on their own, identify objects to grab, and grasp certain objects to carry them to the correct position. They're commonly used to locate and transport items in warehouses, to clean malls and airports, and to gain access to unsafe zones in nuclear power plants and even in the underwater regions to reach into some very difficult locations. On the other hand, the fixed ones stay in their place and separate out the objects which either approach the manipulator or when the manipulator reaches the object. These are widely used when objects keep approaching the manipulator on a conveyor belt are to be separated based on color, shape etc

We create a basic program that would help us interact with the navigation system and then complete the navigation launch. After this comes the process of setting up the manipulation where we start by generating a moveit configuration package and define a self-collision matrix. Then we define all the virtual joints and the planning groups for the manipulator. We also work on defining the poses for the robot and the end effector and then set up the ROS controllers and the perception and do the basic motion planning. Next, we check if the robot is moving like a real robot and then go for adding the controllers to the gripper and with the help of perception, we move it. Then we go for the execution of the trajectory and move the robot arms by the joint positions and the end effector positions. Then we come for the part of identification of the objects for the sorting, we use sensors for the identification of those objects and with the help of this identification we work on creating a simple pick up and place task for the separation of these objects.

Robot manipulators will also be used even more in the near future as their skills improve. In our project we are building a 6 Degree of Freedom robotic manipulator which is capable of lifting objects and placing them in desired locations. This manipulator is enabled with sensors such that it can identify the objects based on the colour and thus be drop the objects at desired location in order to sort them.



IMPLEMENTATION

Step 1: URDF file to design manipulator

A urdf file is required to describe how a robot's design is. Every link, its name, its geometry, physical appearance etc are mentioned which is necessary to build a robot. Links are the rigid bodies. Joints are used to connect two rigid bodies together. How the links are positioned, how are they connected with joints is properly mentioned in this file.

Apart from design, the required plugins are also added. These plugins are required for support for simulation. Gazebo needs some extra plugins for it to perform a few tasks. Two of the major plugins we added are "camera_controller" for camera sensor and "gazebo_grasp_fix" to be able to grasp and grip the object collected.

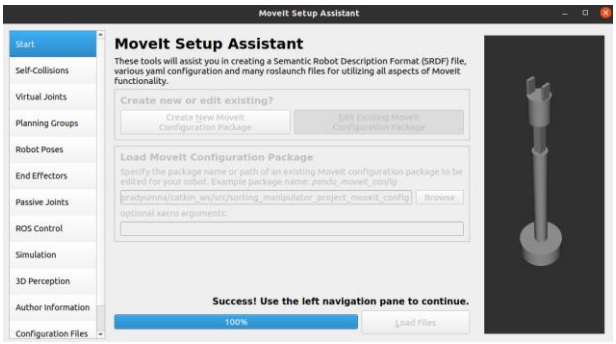
A sample of the same is shown.

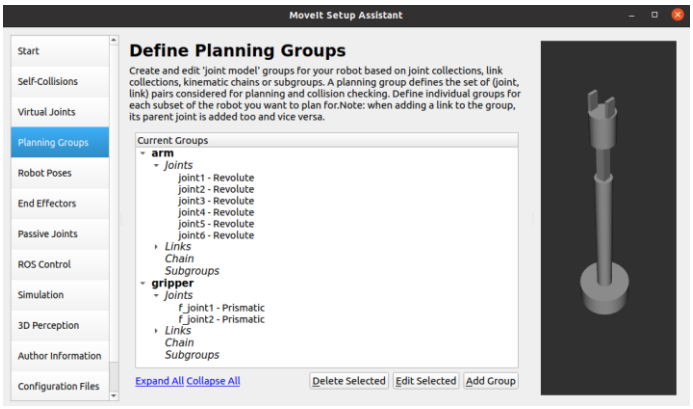
```
    <!-- Link 1-->
    <link name="link1">
        <visual>
            <origin ryp="0 0 0" xyz="0 0 0.05" />
            <geometry>
                <cylinder length="0.10" radius="0.10" />
            </geometry>
            <material name="silver">
                <color rgba="0.75 0.75 0.75 1" />
            </material>
        </visual>
        <collision>
            <origin ryp="0 0 0" xyz="0 0 0.05" />
            <geometry>
                <cylinder length="0.10" radius="0.10" />
            </geometry>
        </collision>

<inertial>
            <mass value="0.1" />
            <inertia ixx="0.03" iyy="0.03" izz="0.03" ixy="0.0" ixz="0.0" iyz="0.0" />
        </inertial>

    </link>
    <gazebo reference="link1">
        <material>Gazebo/Black</material>
        <turnGravityOff>false</turnGravityOff>
    </gazebo>
    <!-- Joint 1-->
    <joint name="joint1" type="continuous">
        <origin ryp="0 0 0" xyz="0 0 0.01" />
        <parent link="base_link" />
        <child link="link1" />
        <axis xyz="0 0 1" />
    </joint>
```

Step 2: Package creation using Moveit

Movit is a setup assistant which helps us quickly set up our robot. It takes in the urdf file which we previously built and provide with many features.

Here, two groups are created. Arm group and the gripper group. Arm group decided different poses which the manipulator can reach without considering the end effector

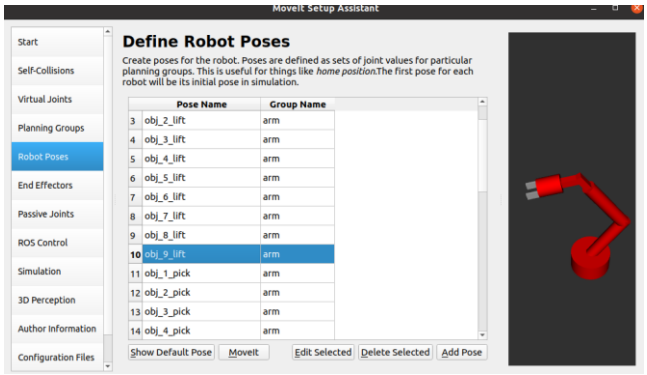


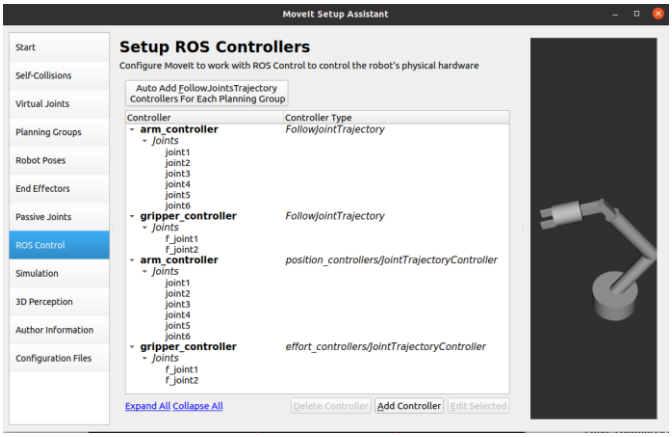


poses. Another group gripper is required to create different poses which the end effector can reach. In our case, the end effector has fingers and thus them for a gripper group.

After the groups are set, we can add multiple poses. Both arm and gripper groups can have multiple poses.

This will help us move the manipulator using rviz.
End effector group is created. Information of the parent link and the group which the grippers belong to is to be specified here.
ROS controllers are where the controller types for the groups created by us are defined. Arm group belongs to arm_cotrollers/JointTrajectoryController. Only then we will be able to publish poses for the manipulator to reach. Same way, as gripper belongs to another group, it falls under effort_controllers/JointTrajectoryController.





Finally, a package is generated with necessary files and it is ready for us to control the manipulator using move_group and rviz.
Apart from these, we can control the manipulator by publishing under the topic /arm_controller/command and /gripper_controller/command respectively.

Step 3: Python script to detect color

In order to detect the color of the object, we used a plugin named “camera_controller”. It publishes a 3D array of values which describe BGR pixel values. Python’s openCV is used to identify the color from the data published by the camera sensor. In order to do this, we subscribe to the topic /mybot/camera1/image_raw and do the necessary comparison in the call back function. From the 3D array, we extract each and every element and see if it fall under possible color combinations. If the blue pixel is common in all three layers, green and red pixel values remain the same, we can conclude that the object is blue in color.
Similarly for red and green objects.

```
def detect_callback(image_msg, color_publisher):
        possible_colors = [[127,25,25],[25,127,25],[25,25,127]]
        global onely_once
        try:
            BLUE = 'BLUE'
            GREEN = 'GREEN'
            RED = 'RED'
            cv_image = bridge.imgmsg_to_cv2(image_msg, 'bgr8')
            x = cv_image.tolist()
            color_found = []

            correct = False
            row = len(x)
            for i in range(row):
                col = len(x[i])
                for j in range(col):
                    blue, green, red = 0,0,0
                    for k in range(3):
                        if x[i][j][k] == possible_colors[0][k]:
                            blue = blue + 1
                        if x[i][j][k] == possible_colors[1][k]:
                            green = green + 1
                        if x[i][j][k] == possible_colors[2][k]:
                            red = red + 1

                    if blue == 3:
                        color_found.append(BLUE)
                    elif green == 3:
                        color_found.append(GREEN)
                    elif red == 3:
                        color_found.append(RED)
                    count_vaues = [color_found.count(BLUE), color_found.count(GREEN), color_found.count(RED)]
                    max_val = max(count_vaues)
                    if max_val > 3500:
                        #print(count_vaues)
                        colors = [BLUE, GREEN, RED]
                        val = max(range(len(colors)), key=lambda i: count_vaues[i])
                        correct = True
                        if onely_once == 0:
                            onely_once += 1
                        elif onely_once == 1:
                            onely_once += 1
                            color_publisher.publish(colors[val])
                        break
                if correct:
                    return
```

Another publisher node is written under the name block_color. This publishes the color identified from this call back function. This is used in other files in order to decide which color object is detected and where it is supposed to be placed. This is done by writing another subscriber node.

Step 4: Python files, to sort objects

Multiple python files are created which publish information under the topic /arm_controller/command and /gripper_controller/command respectively. Locations of all the objects, where objects are supposed to be placed are all made global in order to use them anywhere.

This is the move function which is called first. Necessary publisher nodes are initialized and some of the parameters are fixed. A function call to the sort_object() function is made.

```
def move():
    global move_pub, move_pub_gripper
    move_pub = rospy.Publisher('/arm_controller/command', JointTrajectory, queue_size=1, latch=True)
    move_pub_gripper = rospy.Publisher('/gripper_controller/command', JointTrajectory, queue_size=1, latch=True)
    rospy.init_node('arm_move', anonymous=True)
    global arm_cmd, gripper_cmd
    arm_cmd = JointTrajectory()
    gripper_cmd = JointTrajectory()

    ## ARM
    arm_cmd.joint_names = ["joint1", "joint2", "joint3", "joint4", "joint5", "joint6"]
    ## GRIPPER
    gripper_cmd.joint_names = ["f_joint1", "f_joint2"]

    global joint_point, gripper_joint_point
    ## ARM
    joint_point = JointTrajectoryPoint()
    ## GRIPPER
    gripper_joint_point = JointTrajectoryPoint()

    ## ARM
    joint_point.velocities = []
    joint_point.accelerations = []
    joint_point.effort = []
    joint_point.time_from_start = rospy.rostime.Duration(secs=1, nsecs=0)
    ## GRIPPER
    gripper_joint_point.velocities = []
    gripper_joint_point.accelerations = []
    gripper_joint_point.effort = []
    gripper_joint_point.time_from_start = rospy.rostime.Duration(secs=1, nsecs=0)
    gripper_cmd.points = [gripper_joint_point]


    sort_object(obj_1_lift,obj_1_pick,'1')

    print('\n######### EXECUTION DONE #########\n')
```

This is where the information is published. This function includes another function call of color_detect(). This function is from the color detection python file written.

```
def sort_object(lift, pick, obj_num):
    global move_pub, move_pub_gripper, arm_cmd,gripper_cmd, joint_point, gripper_joint_point
    global green_place_list,red_place_list,blue_place_list
    ## obj_1 lift
    joint_point.positions = lift
    arm_cmd.points = [joint_point]
    move_pub.publish(arm_cmd)
    print('Message Published!!! : Moved to obj_'+obj_num+'_lift')
    time.sleep(4)
    ## obj_1_pick
    joint_point.positions = pick
    arm_cmd.points = [joint_point]
    move_pub.publish(arm_cmd)
    print('Message Published!!! : Moved to obj_'+obj_num+'_pick')
    time.sleep(4)
    ## gripper_close
    gripper_joint_point.positions = [0.0150, -0.0150]
    gripper_cmd.points = [gripper_joint_point]
    move_pub_gripper.publish(gripper_cmd)
    print('Message Published!!! : Object gripped')
    time.sleep(4)

                    ##### COLOR IDENTIFICATION #####
    #global value
    def callBack(msg):
        global value
        value=msg.data

    detect_color()
    sub=rospy.Subscriber("block_color", String,callBack)
    rospy.wait_for_message("block_color",String)
    print("Object color detected : ",value)

    place=[]
    if value=='GREEN':
        place=green_place_list[int(obj_num)]
    elif value=='BLUE':
        place=blue_place_list[int(obj_num)]
    else:
        place=red_place_list[int(obj_num)]

    ## obj_1_lift
    joint_point.positions = lift
    arm_cmd.points = [joint_point]
    move_pub.publish(arm_cmd)
```

This returns us the color of the object which is detected like ‘BLUE’, ‘GREEN’ or ‘RED’. The same is print and according to the identified color, the place position is decided. The sort_object() function is as follows.

Positions to which the manipulator has to move are specified. Once the gripper fingers contract and grip the object, a call to detect_color() functions is made. This returns color as mentioned above.

After this value of color is extracted, the place position is decided and given to the joint_point and gripper_joint_point to publish the same.
Similarly, different files for other objects are created and executed individually.

INFORMATION ON PLUGIN USED

CAMERA PLUGIN:

```
<gazebo reference="finger2">
    <material>Gazebo/Green</material>
    <sensor type="camera" name="camera1">
    <update_rate>1.0</update_rate>
    <camera name="head">
        <horizontal_fov>1.57</horizontal_fov>
        <image>
            <width>800</width>
            <height>800</height>
            <format>R8G8B8</format>
        </image>
        <clip>
            <near>0.00001</near>
            <far>0.05</far>
        </clip>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
        <alwaysOn>true</alwaysOn>
        <updateRate>0.0</updateRate>
        <cameraName>mybot/camera1</cameraName>
        <imageTopicName>image_raw</imageTopicName>
        <cameraInfoTopicName>camera_info</cameraInfoTopicName>
        <frameName>camera</frameName>
        <hackBaseline>0.07</hackBaseline>
        <distortionK1>0.0</distortionK1>
        <distortionK2>0.0</distortionK2>
        <distortionK3>0.0</distortionK3>
        <distortionT1>0.0</distortionT1>
        <distortionT2>0.0</distortionT2>
    </plugin>
    </sensor>
</gazebo>
```

A plugin named camera_controller is used to sense the objects in the world. In urdf, the parent link is specified to which the camera is connected. We connected this camera to the gripper link. When the grippers are closed in order to grab the object, the camera senses the colors it sees. In order to extract information, we need to subscribe to the topic "/mybot/camera1/image_raw". This is an n dimensional array. We need to see the range in which a particular color lies. In python's openCV, the color format is BGR. So, a tuple with (B,G,R) values is to be found. If a blue object is detected, more blue pixels are present. Similarly for green and red objects. According to the value detected, we can take a call on which color object is being gripped. A function for the same is written and called every time before an object is placed where it has to be.

GRASP PLUGIN:

```
</gazebo>
    <gazebo>
    <plugin name="gazebo_grasp_fix" filename="libgazebo_grasp_fix.so">
        <arm>
            <arm_name>arm</arm_name>>
            <palm_link>link6</palm_link>
            <gripper_link>finger1</gripper_link>
            <gripper_link>finger2</gripper_link>
        </arm>
        <forces_angle_tolerance>100</forces_angle_tolerance>
        <update_rate>4</update_rate>
        <grip_count_threshold>4</grip_count_threshold>
        <max_grip_count>8</max_grip_count>
        <release_tolerance>0.005</release_tolerance>
        <disable_collisions_on_attatch>false</disable_collisions_on_attatch>
        <contact_topic>_default_topic_</contact_topic>
    </plugin>
</gazebo>
```

A plugin named gazebo_grasp_plugin is used to hold and lift the object using links in gazebo's world. The plugin is inspired by "gazebo::physics::Gripper". It fixes an object which is grasped to the robot hand to avoid problems with physics engines and to help the object stay in the robot hand without slipping out. This plugin has no dependencies on ROS. It can run in the background and automatically attaches all objects grasped by the gripper links.

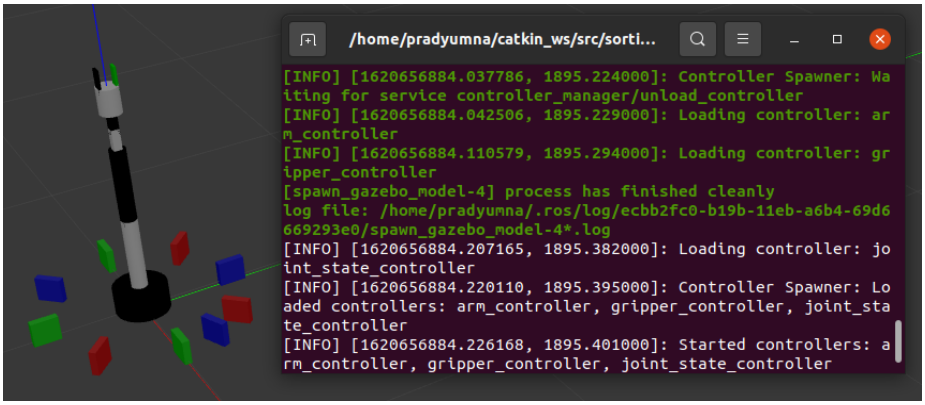
EXECUTION

Step 1: Launch gazebo.launch file

The first file to be launched is gazebo.launch. This file launches the gazebo world file which is specified. We specified a world which has red, green and blue color objects to be sort.

```
<?xml version="1.0"?>
<launch>
  <arg name="paused" default="false"/>
  <arg name="gazebo_gui" default="true"/>
  <arg name="urdf_path" default="$(find sorting_manipulator_project)/urdf/sortingManipulator_urdf.urdf"/>
  <!-- startup simulated world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <!--arg name="world_name" default="worlds/empty.world"/-->
    <!--arg name="world_name" value="$(find sorting_manipulator_project_moveit_config)/worlds/singleObject.world"/-->
    <arg name="world_name" value="$(find sorting_manipulator_project_moveit_config)/worlds/objects.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg gazebo_gui)"/>
  </include>
  <!-- send robot urdf to param server -->
  <param name="robot_description" textfile="$(arg urdf_path)" />
  <!--push robot_description to factory and spawn robot in gazebo at the origin, change x,y,z arguments to spawn in a different position-->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model" args="-urdf -param robot_description -model robot -x 0 -y 0 -z 0"
    respawn="false" output="screen" />
  <include file="$(find sorting_manipulator_project_moveit_config)/launch/ros_controllers.launch"/>
</launch>
```

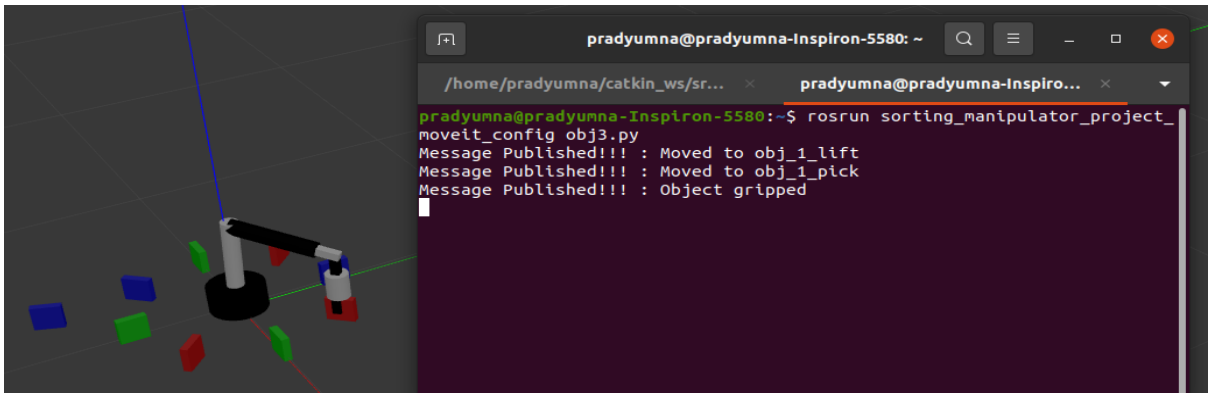
The same way, other .world files can be loaded by specifying its package and file name.
This following is the image when gazebo.launch file is executed.



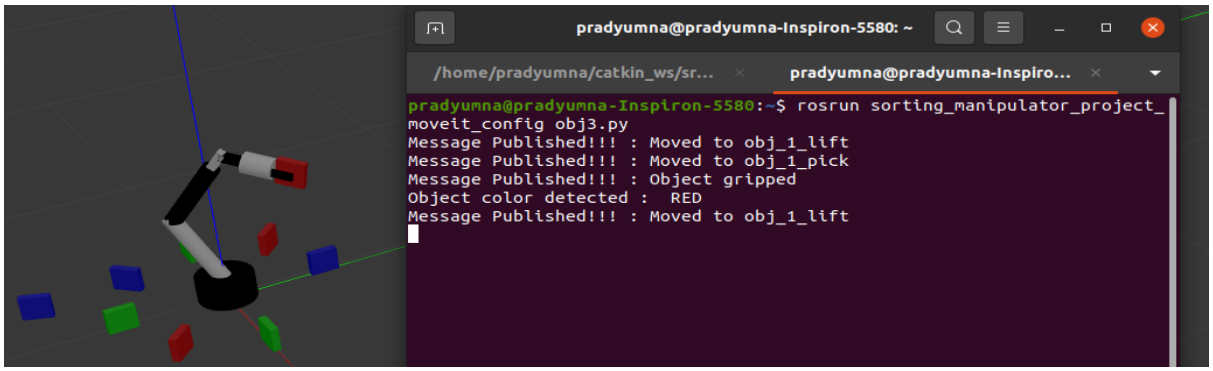
One of the fingers is green in color to show that the camera sensor is on that finger. We placed that camera on that link as it can easily identify the object's color when gripped.

Step 2: Execute the object move python file

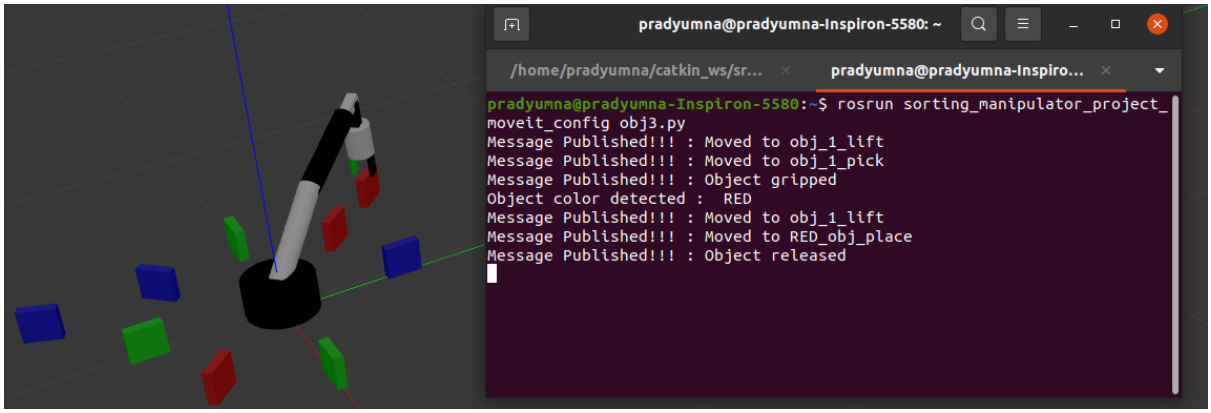
When this is done, a series of pose values is published to arm and gripper groups to reach the object, grip it, collect it, place it at the desired location and come back to initial pose.
The following is the image of how the manipulator reaches to the position of object and color being detected. It is waiting for the response from the color detect function.



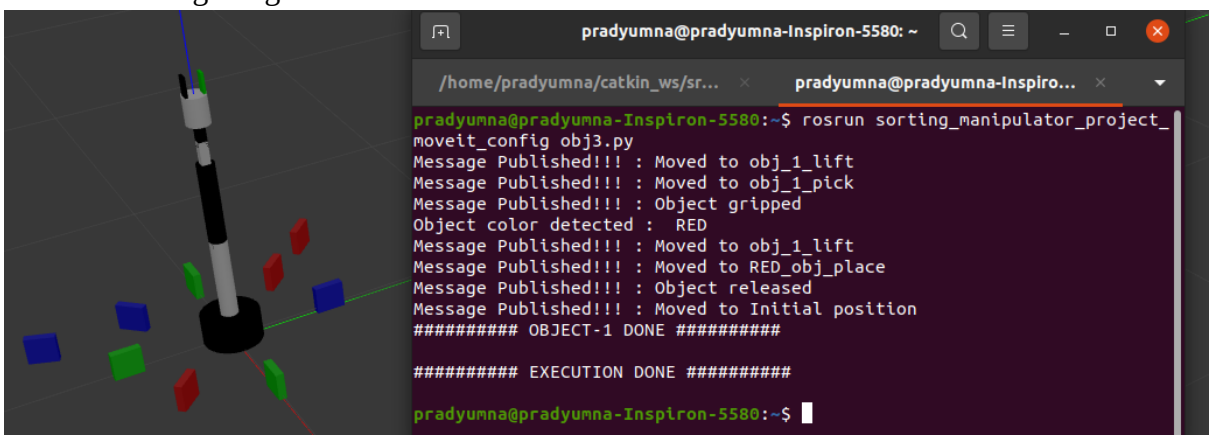
This image show how the aqquired object color is used and how the manipulator lift the object befor placing it at a desired location.



This image shows how the manipulator reached the desired position and grippers are being released in order to drom the object.



Once the object is placed, the manipulator comes bac to its initial position. The same is depicted in the following image.



Similarly, all other objects are detected and are placed at desired location. This is how the objects are sort based on color using a manipulator.

RESULT

A robotic manipulator for sorting objects based on the color has been designed and implemented. In this project, there are a total of nine objects of which three are red, three are green and three are blue. All the objects have been sort using the algorithm we developed.

CONCLUSION

Sorting manipulators are of great use in industrial application. It can be of different types like sorting based on colors, sizes, shapes etc. With rise in production of good, sorting plays a vital role in separating them and storing. This reduces man power and is also quick and accurate.

REFERENCES

Book on Programming Robots with ROS by Morgan Quigley, Brian Gerkey, and William D. Smart
https://answers.ros.org/question/210695/adding-a-camera-to-a-model-in-gazebo-beginner/