UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

ASSIGNMENT COVERSHEET

UTS: ENGINEERING & INFORMATION TECHNOLOGY

SUBJECT NUMBER & NAME	NAME OF STUDENT(s) (PRINT CLEARLY)	STUDENT ID(s)
41069 – Robotics Studio 2	Udhaya Kumar Parameswaran	24603437

STUDENT EMAIL	STUDENT CONTACT NUMBER
UdhayaKumar.Parameswaran@student.uts.edu.au	0435545307

NAME OF TUTOR	TUTORIAL GROUP	DUE DATE

ASSESSMENT ITEM NUMBER & TITLE
e-Portfolio

☐ I confirm that I have read, understood and followed the guidelines for assignment submission and presentation on page 2 of this cover sheet.
☐ I confirm that I have read, understood and followed the advice in the Subject Outline about assessment requirements.
☐ I understand that if this assignment is submitted after the due date it may incur a penalty for lateness unless I have previously had an extension of time approved and have attached the written confirmation of this extension.

Declaration of originality: The work contained in this assignment, other than that specifically attributed to another source, is that of the author(s) and has not been previously submitted for assessment. I understand that, should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with University policy and rules. In the statement below, I have indicated the extent to which I have collaborated with others, whom I have named.

Statement of collaboration:

Signature of student(s) Udhaya Kumar Parameswaran **Date** 04-Jun-2024

✂- -

ASSIGNMENT RECEIPT

To be completed by the student if a receipt is required

SUBJECT NUMBER & NAME	NAME OF TUTOR

SIGNATURE OF TUTOR	RECEIVED DATE

Table of Contents

e-Portfolio

Project Title

Pick and Place using UR3

Requirements/ Contract

Story- Pick and place a package on and off a moving TurtleBot using a camera to identify the correct package.

Subsystem- Waypoint generation, Trajectory Planning and Control.

- F: Connect with UR3 using ROS and unable to move robot joints.
- P: Send the target end-effector pose to UR3, and it will determine how to get there.
- C: Way-point generation using Dijkstra/ A*/ RRT avoiding stationary obstacles (map does not get updated).
- D: Sending smooth joint angle trajectories to the UR3 to pick and place objects off and on a stationary turtle-bot.
- HD: Pick and place objects off and on a moving turtle-bot.

System Architecture

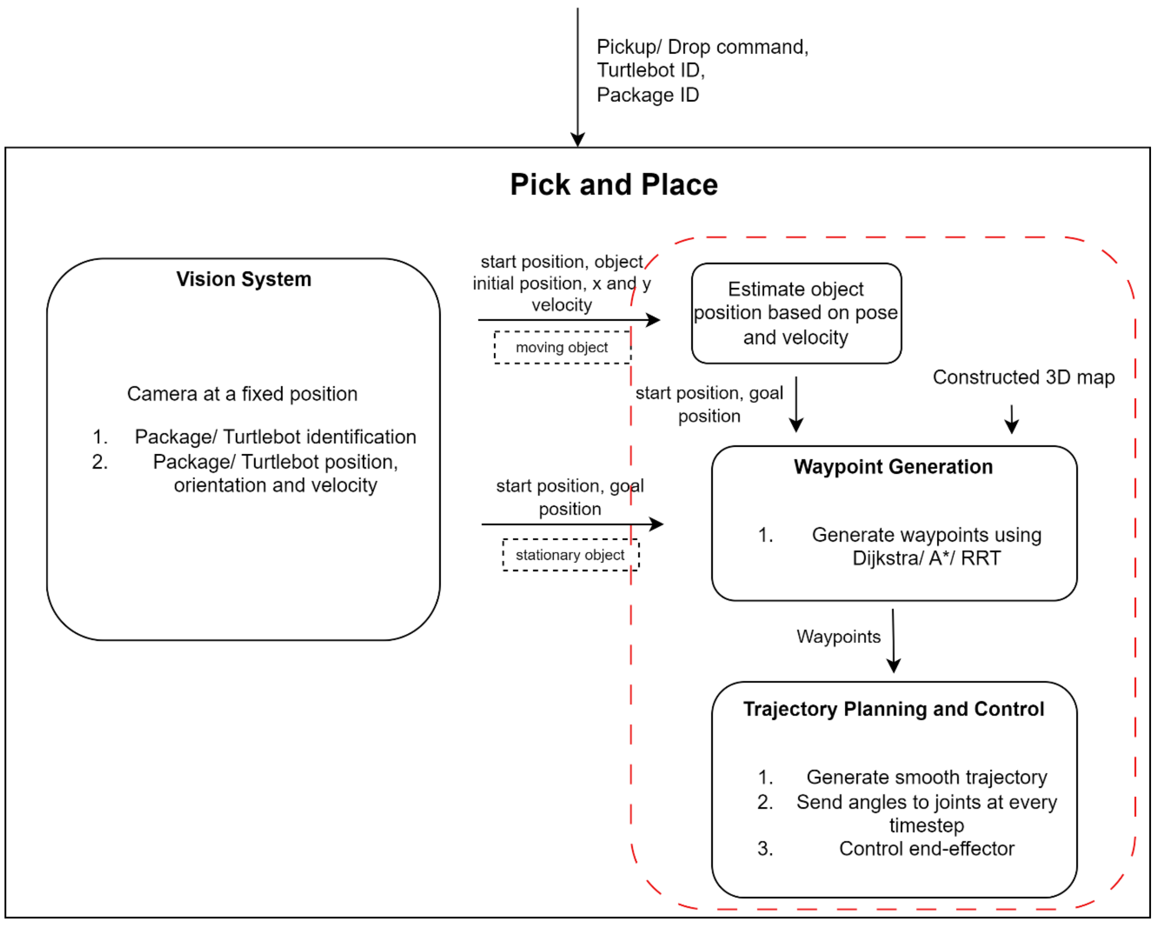
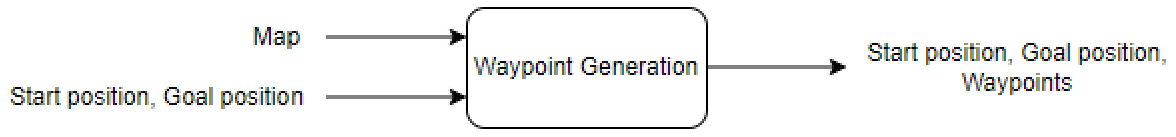

Pickup/ Drop command,
Turtlebot ID,
Package ID
Pick and Place
Vision System
Camera at a fixed position
1. Package/ Turtlebot identification
2. Package/ Turtlebot position, orientation and velocity
start position, object initial position, x and y velocity
moving object
Estimate object position based on pose and velocity
start position, goal position
Constructed 3D map
start position, goal position
stationary object
Waypoint Generation
1. Generate waypoints using Dijkstra/ A*/ RRT
Waypoints
Trajectory Planning and Control
1. Generate smooth trajectory
2. Send angles to joints at every timestep
3. Control end-effector

Fig. 1 System Architecture

Test Plans and Evidence

Unit Tests

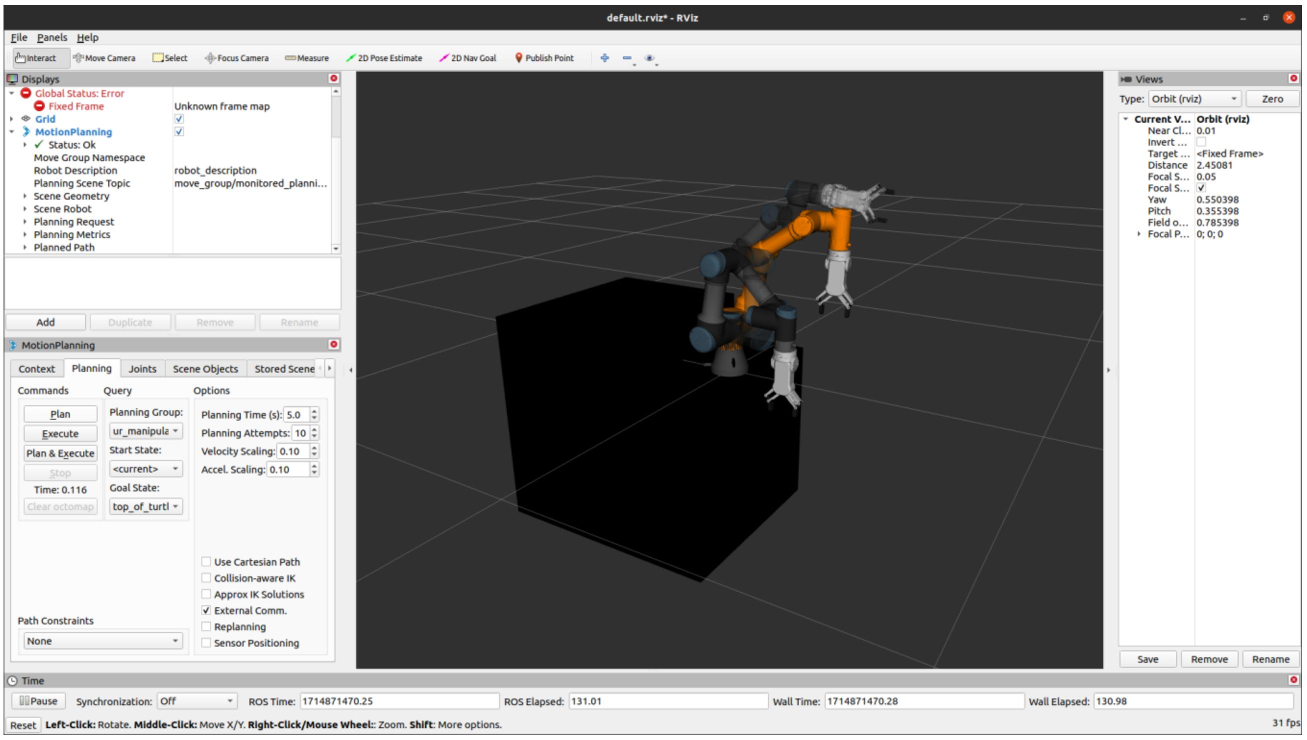
Test 1:


Map
Start position, Goal position
Waypoint Generation
Start position, Goal position, Waypoints

1. This test addresses the requirement of waypoint generation without colliding with fixed obstacles as detected in the map.
2. With the available map, arbitrary start and goal positions are given, and the output waypoints are generated.
3. The given waypoints are visualised using Moveit and examined whether the UR3 collides with fixed obstacles. If it collides, the system fails; if not, it passes.
4. A pre-constructed map, start and goal positions.

Result: Success

Comments: The model was built with stationary obstacles, and the gripper was fitted with the UR3. A Moveit configuration file was generated from the model, and this made it easy to avoid obstacles and plan from point A to point B without any issues. If poses that cannot be achieved are given, such as outside the workspace or where a collision object is located, the system simply ignores the request and waits for the next input pose.


default.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Add Duplicate Remove Rename
MotionPlanning
Context Planning Joints Scene Objects Stored Scene
Commands Query Options
Plan Execute Plan & Execute Stop Clear octomap
Views
Save Remove Rename
Time
Pause
Reset

Global Status: Error
Fixed Frame: Unknown frame map
Grid ✓
MotionPlanning ✓
Status: Ok
Move Group Namespace
Robot Description: robot_description
Planning Scene Topic: move_group/monitored_planni...
Scene Geometry
Scene Robot
Planning Request
Planning Metrics
Planned Path

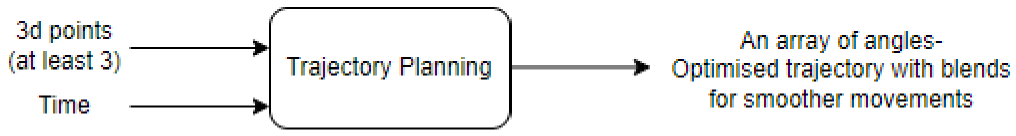
Time: 0.116
Planning Group: ur_manipula
Start State: <current>
Goal State: top_of_turtl
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
☐ Use Cartesian Path
☐ Collision-aware IK
☐ Approx IK Solutions
☑ External Comm.
☐ Replanning
☐ Sensor Positioning
Path Constraints: None

Type: Orbit (rviz) Zero
Current V... Orbit (rviz)
Near Cl... 0.01
Invert ... ☐
Target ... <Fixed Frame>
Distance 2.45081
Focal S... 0.05
Focal S... ☑
Yaw 0.550398
Pitch 0.355398
Field o... 0.785398
Focal P... 0; 0; 0

Synchronization: Off
ROS Time: 1714871470.25
ROS Elapsed: 131.01
Wall Time: 1714871470.28
Wall Elapsed: 130.98
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.
31 fps

Fig. 2 Waypoint Generation

Test 2:



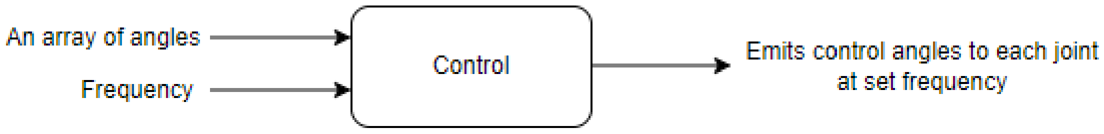
1. This test addresses the requirement of trajectory planning from the given 3d points.
2. A group of 3d points (at least 3) and time are sent, and an array of angles for individual joints are generated with blends at each point to facilitate smooth movement.
3. The 3d points given and the end-effector positions are visualised digitally. If the generated points are closer to the input points with minimal deviations and smoother transitions, the system passes; if not, it fails.
4. Some 3d points and total time.

Note: The end pose of the UR3 is not considered at this point but will be included during final testing.

Result: Not attempted

Comments: The time at which the UR moves from point A to point B is not controlled here, as the object to be picked up is stationary.

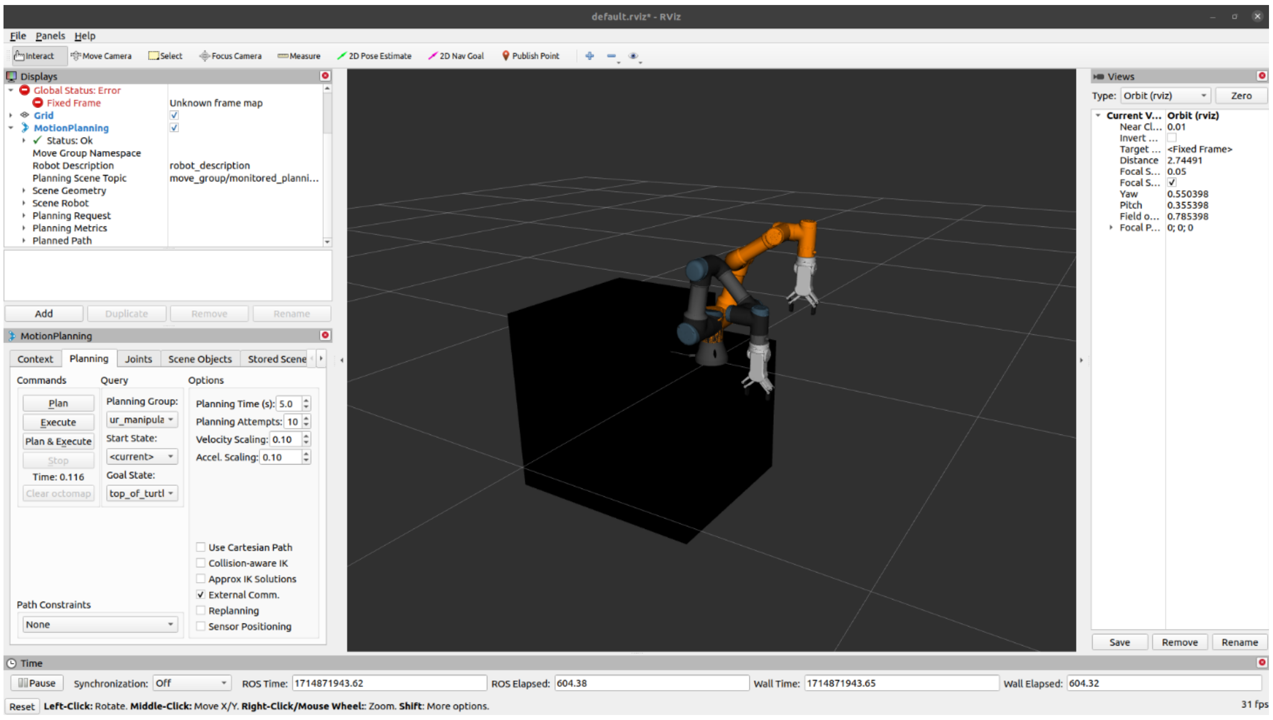
Test 3:



1. This test addresses the requirement of controlling the UR3 with the given angles by sending it out at a particular frequency.
2. An array of angles and the frequency at which the signals should be sent are given as inputs, and the individual joints' actual movements are observed as outputs.
3. The frequency at which the angles are sent out is measured using ROS, and the actual path taken versus the generated path is compared.
4. An array of angles and frequency.

Result: Success

Comments: The UR3 was able to move from point A to point B as per the plan generated in the previous step without any collision.


default.rviz* - RViz

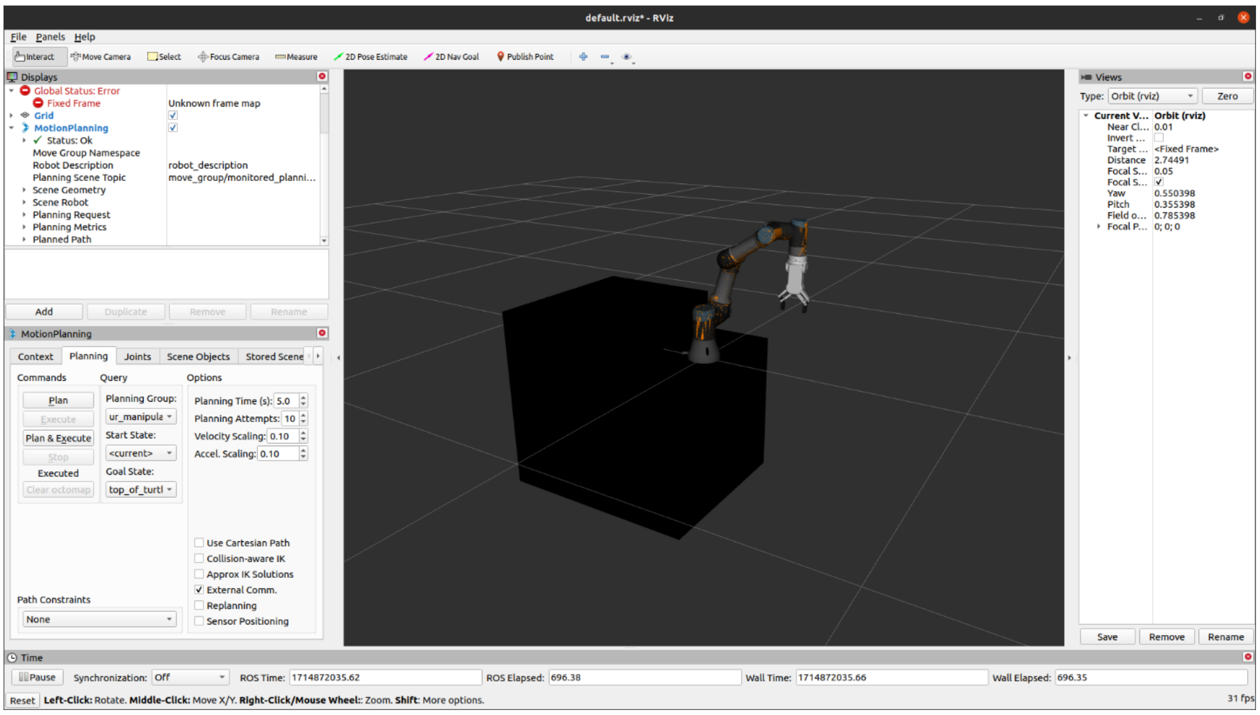
Global Status: Error
Fixed Frame: Unknown frame map
Robot Description: robot_description
Planning Scene Topic: move_group/monitored_planni...

Planning Group: ur_manipulator
Start State: <current>
Goal State: top_of_turtl
Time: 0.116
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
External Comm. ✓
Path Constraints: None

Type: Orbit (rviz)
Near Cl...: 0.01
Target ...: <Fixed Frame>
Distance: 2.74491
Focal S...: 0.05
Yaw: 0.550398
Pitch: 0.355398
Field o...: 0.785398
Focal P...: 0; 0; 0

Synchronization: Off
ROS Time: 1714871943.62
ROS Elapsed: 604.38
Wall Time: 1714871943.65
Wall Elapsed: 604.32
31 fps

Fig. 3 UR3 at point A with target point B (shown in orange)


default.rviz* - RViz

Global Status: Error
Fixed Frame: Unknown frame map
Robot Description: robot_description
Planning Scene Topic: move_group/monitored_planni...

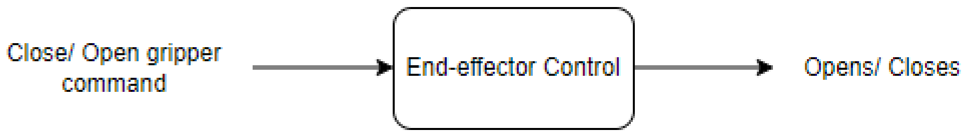
Planning Group: ur_manipulator
Start State: <current>
Goal State: top_of_turtl
Executed
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
External Comm. ✓
Path Constraints: None

Type: Orbit (rviz)
Near Cl...: 0.01
Target ...: <Fixed Frame>
Distance: 2.74491
Focal S...: 0.05
Yaw: 0.550398
Pitch: 0.355398
Field o...: 0.785398
Focal P...: 0; 0; 0

Synchronization: Off
ROS Time: 1714872035.62
ROS Elapsed: 696.38
Wall Time: 1714872035.66
Wall Elapsed: 696.35
31 fps

Fig. 4 UR3 at point B

Test 4:



1. This test addresses the requirement of controlling the end-effector of the UR3.

2. Open and close commands are given to the UR3 based on the system's state, and the end-effector acts accordingly.
3. Open command- opens, Close command- closes- pass, if not fail.
4. Appropriate command and gripper.

Result: Success

Comments: The gripper was connected to the PC using an ethernet cable and accessed using the default IP address (192.168.1.1). A simple controller node from the package (onrobot) was used to control the gripper.

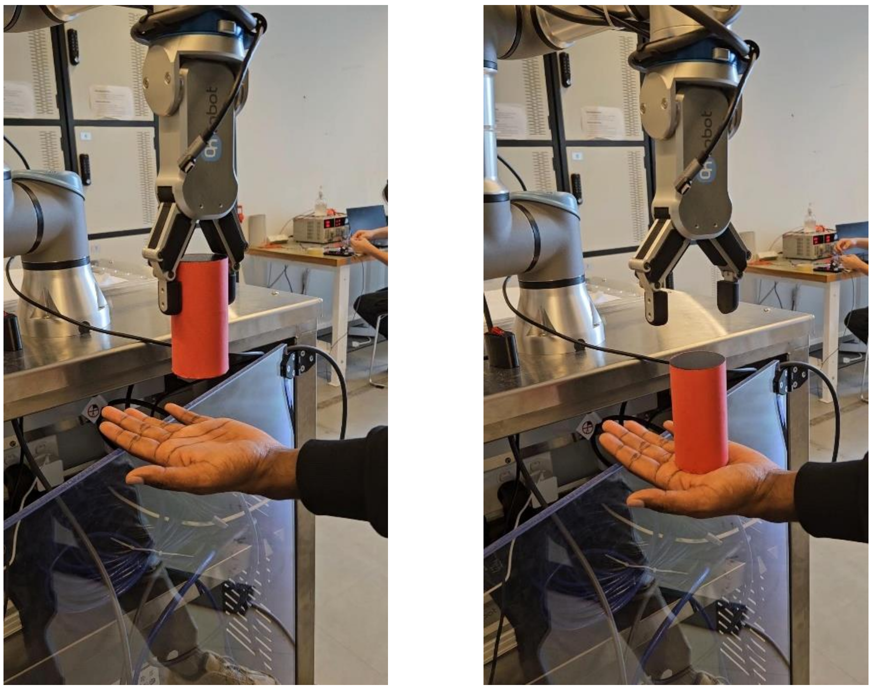
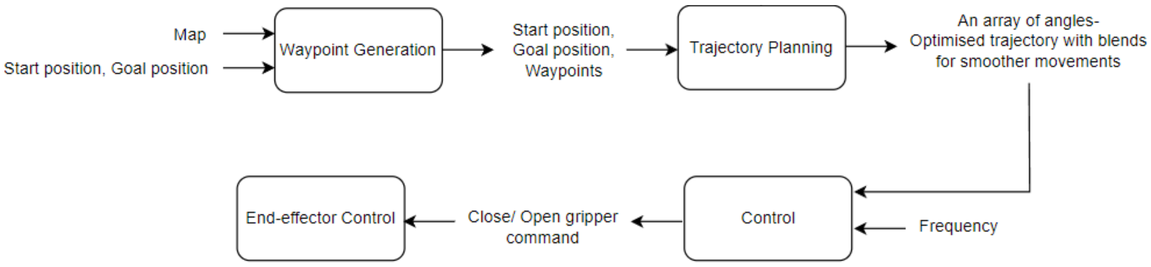


Fig. 5 RG2 Gripper closed (left) and open (right)

Integration Tests

Test 1:

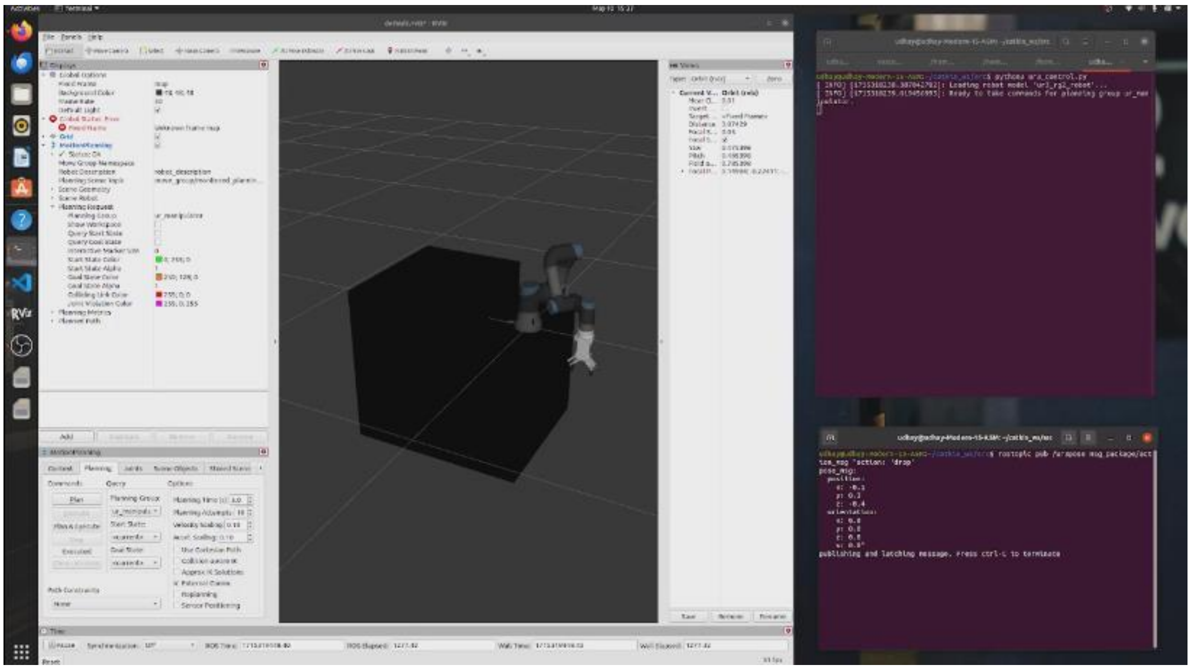

Map
Start position, Goal position
Waypoint Generation
Start position, Goal position, Waypoints
Trajectory Planning
An array of angles- Optimised trajectory with blends for smoother movements
Control
Frequency
Close/ Open gripper command
End-effector Control

1. This test addresses the system's waypoint generation, trajectory planning and control.
2. The preset map and the start and goal positions are given as inputs to the system. The system generates waypoints avoiding obstacles, plans trajectory based on these points, and reaches the end pose. The gripper is actuated at the end of this sequence to grab the object.
3. If the UR3 can hold the object at a given position in 3d space, moving from a random position at a given time, then the system passes; if not, it fails.
4. Resources needed- Map, start and end positions and gripper.

Result: Success

<u>Comments</u>:

The goal position is given to a topic through the command line, whereas the map and the start position are already known to the system. Refer to Fig 6.

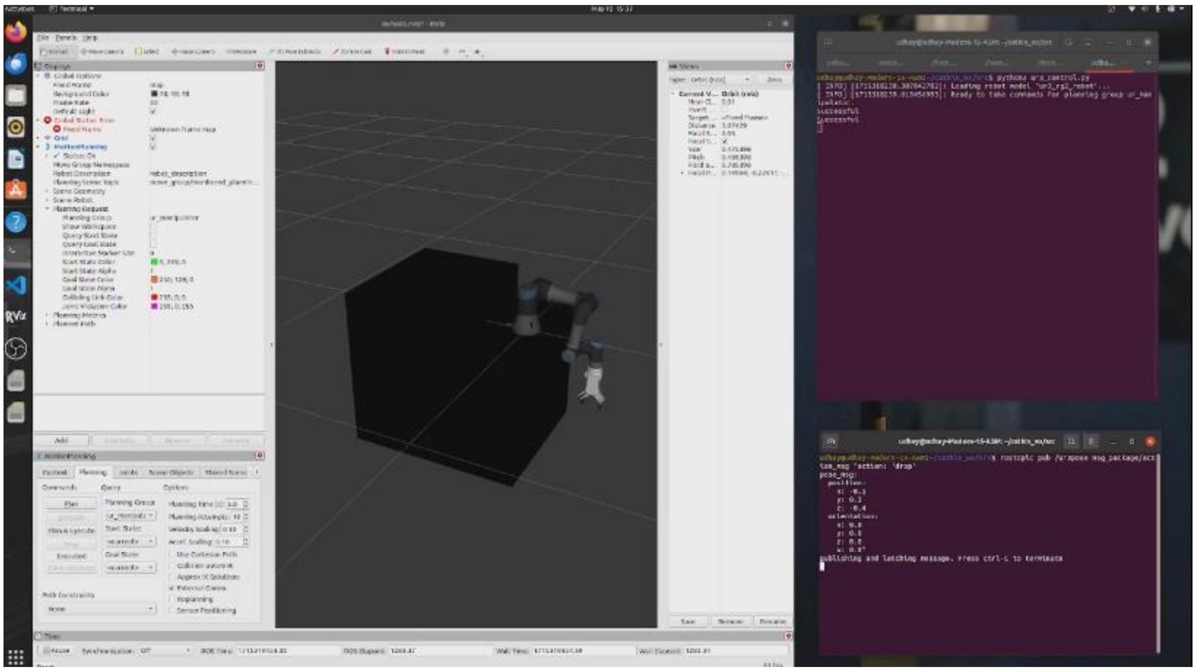


```
[illegible]$ python3 arm_control.py
[ INFO] [1715310138.387042782]: Loading robot model 'ur3_rg2_robot'...
[ INFO] [1715310139.019456993]: Ready to take commands for planning group ur_manipulator.
```

```
[illegible]$ rostopic pub /arpose msg_package/action_msg "action: 'drop'
pose_msg:
  position:
    x: -0.1
    y: 0.3
    z: -0.4
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 0.0"
publishing and latching message. Press ctrl-C to terminate
```

Fig. 6 UR3 at random start position receiving goal position

Based on these inputs, the waypoints and trajectory are computed, and the end effector reaches the desired position. Refer to Fig 7.

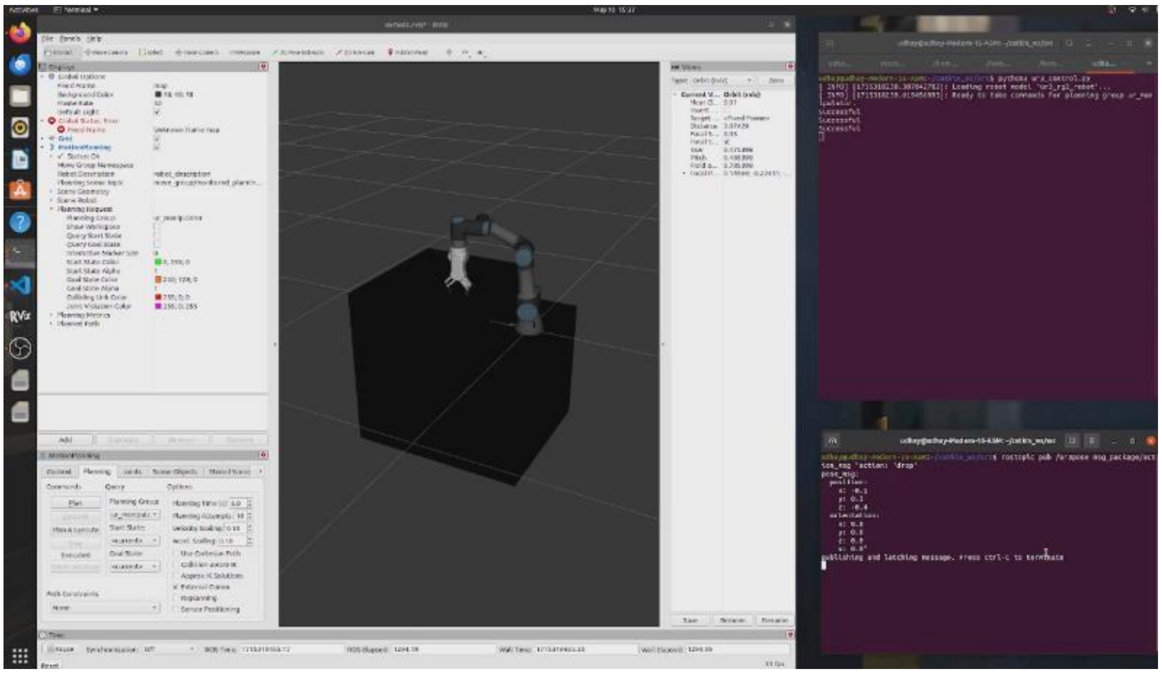


```
[illegible]$ python3 arm_control.py
[ INFO] [1715310138.387042782]: Loading robot model 'ur3_rg2_robot'...
[ INFO] [1715310139.019456993]: Ready to take commands for planning group ur_manipulator.
successful
Successful
```

```
[illegible]$ rostopic pub /arpose msg_package/action_msg "action: 'drop'
pose_msg:
  position:
    x: -0.1
    y: 0.3
    z: -0.4
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 0.0"
publishing and latching message. Press ctrl-C to terminate
```

Fig. 7 UR3 at the destination position

From the goal position, the UR3 is programmed to move to the storage locations (3 different locations), which are hard-coded. Refer to Fig 8.

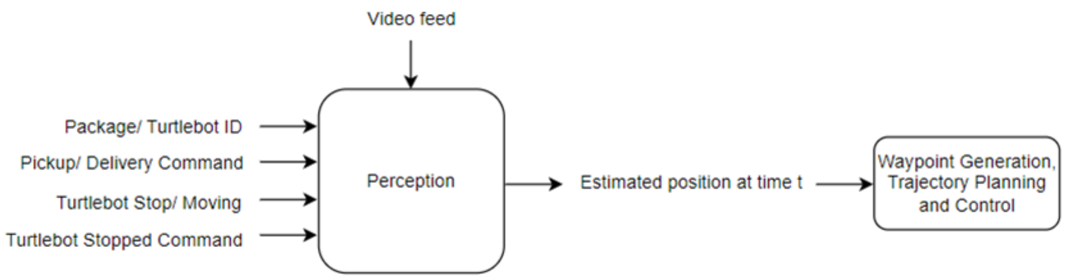


```
[illegible]$ python3 ur3_control.py
[ INFO] [1715310138.387842782]: Loading robot model 'ur3_rg2_robot'...
[ INFO] [1715310138.819456895]: Ready to take commands for planning group ur_man
ipulator.
Successful
Successful
Successful
```

```
[illegible]$ rostopic pub /arm_pose msg_package/act
ion_msg "action: 'drop'
pose_msg:
  position:
    x: -0.1
    y: 0.1
    z: -0.4
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 0.0"
publishing and latching message. Press ctrl-C to terminate
```

Fig. 8 UR3 at the storage location

Test 2:



Video feed
Package/ Turtlebot ID
Pickup/ Delivery Command
Turtlebot Stop/ Moving
Turtlebot Stopped Command
Perception
Estimated position at time t
Waypoint Generation, Trajectory Planning and Control

1. This test addresses the overall pick and place operation, combining the camera perception and the actual physical pick and place operation using UR3.
2. The object to be picked up is placed in a location accessible to the UR3 and near the camera, either stationary or on a moving platform. This scene, along with other information from the system such as package/ turtlebot ID, pickup/ delivery command, turtlebot stop/ moving, and turtlebot stopped command, is used as input, and the physical movement of the end effector from one position to the position determined by the perception system is the output.
3. If the UR3 can hold the object at the position determined by the perception system, moving from a random position at a given time, then the system passes; if not, it fails.
4. Resources needed- Map, an RGB-D camera, UR3 and gripper.

<u>Result</u>: Failed

<u>Comments</u>: The communication between the two PCs was successful. Messages can be published on the topic to which the UR3 control program is subscribed. However, the position given to the UR3 from the vision system after being transformed with respect to the world frame is off by a few centimetres randomly, making it challenging to pick up the package.

Description

The first step was to create a URDF file with the UR3, gripper, the cart and other objects surrounding the UR3. This file was used as input to the moveit setup assistant to create a moveit config file. This generates a series of launch files that can be used to access various functionalities.

In the Python code used to control the UR3, the moveit_commander module was used to access the manipulator. A node was initialised with one publisher to control the gripper and a subscriber to the topic '/ur3pose', which is a custom message with the type of action to be performed (string) and the goal position (pose).

The UR3 was initialised with a series of constraints for five different joints in order to restrict moveit from performing unnecessary and complicated moves. The different constraints are,

#	Joint Name	Range	Comments
1	shoulder_pan_joint	-2.00 to 2.50 rad	To restrict the base to move in one direction
2	shoulder_lift_joint	-2.3 to 0.7 rad	To restrict the arm from accessing the storage location in weird positions
3	elbow_joint	0.17 to 3.17 rad	To restrict the arm from passing through a singularity
4	wrist_2_joint	-1.27 to -1.87 rad	To restrict wrist two from spinning continuously (because of moveit)
5	wrist_3_joint	-3.14 to 3.14 rad	To limit the rotation of the gripper to prevent damage to the cable

The system waits for a message to be published on the topic '/ur3pose'. Whenever the system receives position information, the end-effector's orientation is maintained to avoid unnecessary movement, twisting and orientation. When it receives one that is unreachable or in a collision path, the system simply ignores it and waits for the following message. The task sequence depends on the command received if the system receives a valid input.

The UR3 needs to access a storage location on its side for both sequences. This consists of three locations 10 cm apart and on the cart, as shown in the image below. These locations are placed in a dictionary along with a flag to indicate whether that position is occupied.

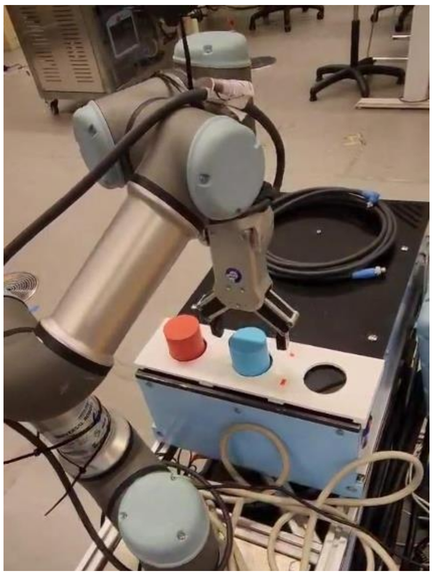


Fig. 9 Storage locations

Action msg – ‘pickup’

The string ‘pickup’ represents that the turtlebot is here to pick up a parcel. This is bundled along with the position of the turtlebot where the package should be placed. The sequence is as follows:

1. The dictionary with the storage details is checked for available packages.
2. If packages are available, the package's location first in the list is assigned as the destination.
3. The UR3 reaches a position just above 5cm from the actual position of the package.
4. Then, it travels 5cm down the Z axis, and the gripper is closed to pick up the package.
5. The UR3 retreats from this position by moving 5cm from the current location.
6. The position that was passed in the topic is assigned as the destination, and the accessibility of the location is checked.
7. If accessible, the UR3 moves just 5cm above the destination to the actual location and opens the gripper.

Action msg- ‘drop’

The string ‘drop’ represents that the turtlebot is here to drop off a parcel. This is bundled along with the position of the package on top of the turtlebot. The sequence is as follows:

1. The dictionary with the storage details is checked for available empty locations.
2. If empty spots are available, the package's location is assigned as the destination.
3. The UR3 reaches a position just above 5cm from the actual position of the package.
4. Then, it travels 5cm down the Z axis, and the gripper is closed to pick up the package.
5. The UR3 retreats from this position by moving 5cm from the current location.
6. The empty slot in the storage is assigned as the destination.
7. Then, the UR3 moves just 5cm above the destination to the actual location and opens the gripper.

GitHub Repo

https://github.com/UdhayROB/Robotics_Studio_2.git