CMSC 489F

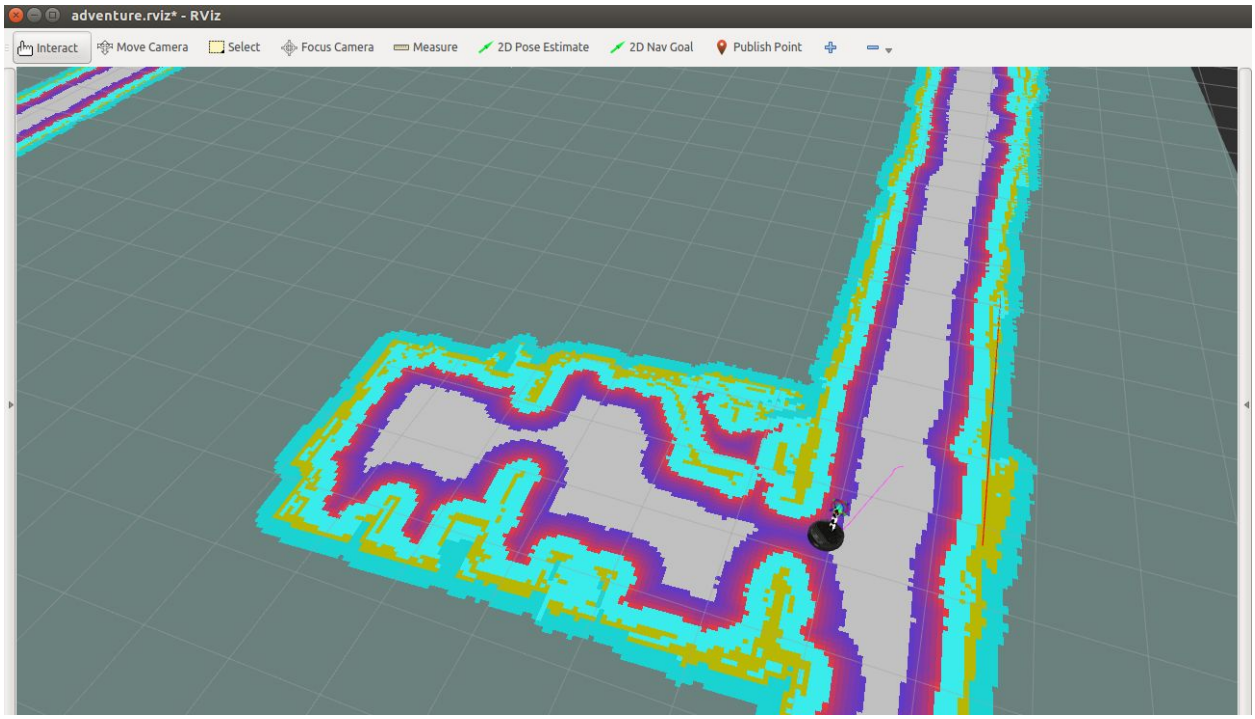
Project 6 Report

Reign of Gunters

Yash Manian, Miguel Angel Maestre Trueba and Yi Mao

1. Navigation

We used the AMCL library to plan paths in 2D space such that coordinates for the required position can be applied through the python script by sending coordinates to actionlib, and a path will be planned through the pre-existing map. This requires that the robot localizes itself in its given map before the planning starts. The localization suffers inside the lab, as the objects on the floor create an environment different from the map the robot is localizing itself in. The corridors however, allow good localization and are therefore considered for the final positions of the bottles.



2. Object Detection using SIFT

We used the openCV SIFT matching pipeline to match features with predefined training images, with the objects in the robot's environment. This pipeline was adapted for ROS, such that a node is created, which subscribes to the camera image topic from the ASUS Xtion Pro. This image is then processed to find matches with the environment. We get a bounding box around the object once enough features are detected (in this case 15). We used this bounding box to get an approximate distance to the object, as the object's size is already known. This distance is then published.

Since some of the times, the access to the robot was limited the SIFT was tested using the webcam of a computer instead of the robot's camera. The only change that has to be made to the code is to change the topic to which it subscribes.

Lastly, instead of a KNN Matcher, a Brute Force Matcher is used for this part of the code. The overall results of the SIFT Detection are very successful.

3. Using the Turtlebot Arm

The turtlebot arm uses the MoveIt library to take care of the Inverse Kinematics and the planning. This allows us to provide the end effector with 3D position and orientation, which it will then plan to and follow.

However, we encountered several errors with the library, and decided to edit the SRDF files that dictate hardware protocol. These SRDF files from the adventure package were edited to add the following poses to them:

```
<group_state name="pos1" group="arm">
    <joint name="arm_wrist_flex_joint" value="1.4" />
    <joint name="arm_elbow_flex_joint" value="2.3" />
    <joint name="arm_shoulder_lift_joint" value="-2.0" />
    <joint name="arm_shoulder_pan_joint" value="0.0" />
</group_state>
<group_state name="pos2" group="arm">
    <joint name="arm_wrist_flex_joint" value="-0.43" />
    <joint name="arm_elbow_flex_joint" value="-1.29" />
    <joint name="arm_shoulder_lift_joint" value="0.33" />
    <joint name="arm_shoulder_pan_joint" value="-0.37" />
</group_state>
```

However, upon further investigation, it was found that the package does not use any of its own SRDF files, and uses the default ROS installation SRDF files from /turtlebot_arm_moveit_config, and thus we edited those files to add the above poses to the list of available poses.

These poses cover a sweeping motion which is supposed to knock over a bottle within 15cm of the kobuki base. These poses are then called through the moveit commander class in python, allowing the robot to execute the motion on command.

4. Putting it all together

This would probably be the most challenging part of the project. For this part, a ROS intermediate script was written in python, which handles all of the various messages from the modules mentioned above.

The general flow of the system can be described as follows:

1. The robot localizes itself in the pre-existing map.
2. The intermediate “starter” code publishes the position and orientation of the first bottle on the topic **/navigation_arguments**.
3. The navigation node subscribes to the topic and takes in the pose, passing it into the actionlib to start navigating towards the first bottle.
4. Meanwhile, this message in topic **/navigation_arguments** is also sent to SIFT node, so SIFT can start to train itself using an image of the current bottle. Similarly, in the following iterations, when the robot navigates to a new bottle, it will train itself using an image of the new bottle at the same time, so it will be able to detect the new bottle after it reaches the desired pose.
5. Once the pose is reached, the navigation node publishes a boolean flag to the topic **/navigation_flag**.
6. This indicates to the starter node that the navigation has completed successfully, and that the SIFT has to be initialized, which it does by publishing another boolean flag to **/sift_arguments**.
7. The SIFT code subscribes to the topic, and starts running SIFT in the callback, where the feature matching runs. The assumption here is that the robot is close enough to the bottle that it can detect features on it. The robot rotates clockwise till it detects features matching the current bottle it has been trained on. Once the bottle has been detected, the robot moves forward with respect to the size of the bounding box. Once the size of the bounding box is within a certain threshold, the robot stops moving, and sends out the boolean flag to **/sift_flag**, which lets the starter node know that the SIFT has successfully detected the feature, and has reached the required position next to it. The call back to the subscriber on the starter code for this flag publishes a boolean flag to the topic **/gripper_arguments**, which is linked to the gripper node.
8. The gripper node subscribes to the topic, and in the callback, it goes through the poses for sweeping the bottle, and publishes another boolean flag to **/gripper_flag**. This allows the starter node to know when the gripper has successfully completed its poses. The callback for the **/gripper_flag** on the starter code publishes the next position and orientation to **/navigation_arguments**.

5. Problems Solved

1. The biggest problem was the SRDFs on the turtlebot_arm. They referenced the default turtlebot_arm_moveit_config files. This was solved by changing the on board SRDF on Finn.
2. Another major problem was the loss of localization on turning to the back of the lab. This was due to all the objects scattered around the area, which interfered with the map. This was solved by having all the objects placed outside the lab, in the corridor, where the localization was much better.
3. We also faced problems with the ROS framework, where we had device failures, random crashes and dropped messages. We solved these by trying everything over and over again, hoping that the crash wouldn't happen during the current trial (Which happened about 60% of the time).
4. The turtlebot arm would be unresponsive a lot of times. We solved this by increasing the amount of time in the rospy.sleep() function.
5. We had several cases of false detections on the SIFT, which was eventually made quite robust through a combination of increasing the number of features which form a valid detection and getting clearer training images of the objects to be detected. Also, since the camera image was inverted, we inverted the training images to get better matches.

6. Conclusion

All parts of the project have been implemented and tested on the real robot, save for the actual gripping and tossing of the bottle.