Final Project
Introductory Robot Programming

December 13, 2021

Instructors:

Z. Kootbally

Group:

Group1_Sec1

Students:

Hae Lee Kim, Yoseph Kebede, Mohammad Baaqail

Semester:
Fall 2021
Course Code:
ENPM809Y

Contents

1. Introduction

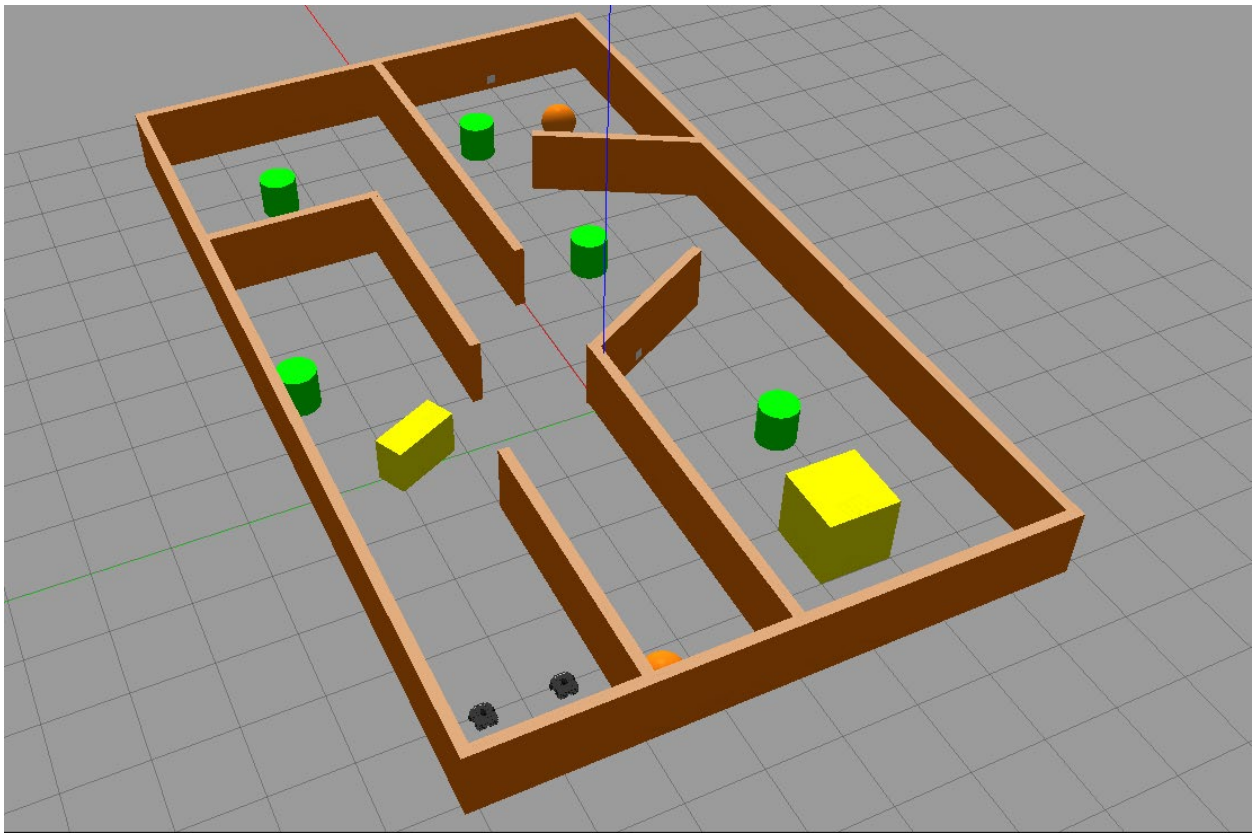

Figure 1: The simulation of the final project map.

In this project, the team mission is to program two robots of Turtlebot3 type to achieve a discovery & rescue mission. This project is inspired by the challenge of autonomous robotics for Urban Search and Rescue (US&R). The goal of this challenge is to be able to design and program a robot that can initially discover risky or unknown areas, and then locate any human victims that could be found along the discovery mission. This robot application will save many lives in events such as a collapsed building where victims get stuck or lost under the debris of the building. Such a robot can be extremely helpful as the human body may find it inconvenient to discover such risky areas.

For the purpose of this project, a real-life example is simulated where there are two robots. The robot used for this project is called Turtlebot3 which offers programmable ROS-based mobile robots. Two turtlebots will be used, each turtlebot has a specific task to do. The first robot (called Explorer) is

required to discover the map and locate the victims. Each victim will be represented using an ArUco marker which is a squared fiducial marker distributed randomly in the map. Once the Explorer is done discovering the area and locating the victims, it goes back to where it started from to let the other turtlebot complete the rescue mission. The second turtlebot (called Follower) is responsible for following the path to the discovered victims or ArUco markers in the order of discovery. There will be four ArUco markers distributed in the map and each one has a label from 0 to 3. After finding all the 4 victims, the Follower can go back to its base or start location. By doing so, the mission is considered accomplished.

2. Approach

A. Tools and Software:

- **Visual Studio Code (VSC)** : The main code editor environment to build and debug the code.
- **Robot operating system (ROS)**: A set of software libraries to ease programming robotics applications.
    - **Gazebo** : 3D simulation tool for ROS application. Physics mechanics involved in this application.
    - **RVIZ** : 3D visualization tool for ROS application. Used for testing and debugging.
- **C++** : The programming language used to build the robot.
- **Doxygen** : To generate and format documentation for the code.

B. Approach analysis:

This section will explain how the team decided to approach the problem and plan accordingly. The first step that has been taken by the group is to analyze each movement and tasks required for each robot. Since the discovery mission starts with the Explorer bot, individual tasks that might be needed for the robot to achieve its goal were brainstormed. Those tasks were listed in an outline form. For example, some of the main tasks that were listed are as follows:

- How is the robot going to find target locations?
- How will we store the four markers' locations in the map and label them?
- How will the robot detect the ArUco markers and store its position in the map?

The other topic that was initially discussed was the second robot which is the Follower. The main tasks that were discussed for this robot are as follows:

- How will the Follower bot receive the target locations from the Explorer bot?

- How can the Follower follow the proper order of ArUco markers stored by the Follower?
- How to repeat the process and eventually return to the start position?

These were the main bullet points in terms of the Explorer and the Follower robots. The next idea that was a crucial aspect of the design was implementing broadcaster/listener. The broadcaster is responsible for creating a frame of the marker and the listener would retrieve the coordinate information and store those coordinates so they can be used to guide the Follower robot. The next subsection explains how those questions were tackled and solved to achieve the final mission.

Programming Methodology

This challenge was handled via the use of Object-Oriented Programming as a means of arranging code execution. Since the Explorer would have the navigation portion while traveling to the target followed by the sensing and marker frame transformation; it was seen as convenient to use ROS's move_base package for the autonomous navigation within the main source file while handling messages from the publishers and subscribers in OOP.

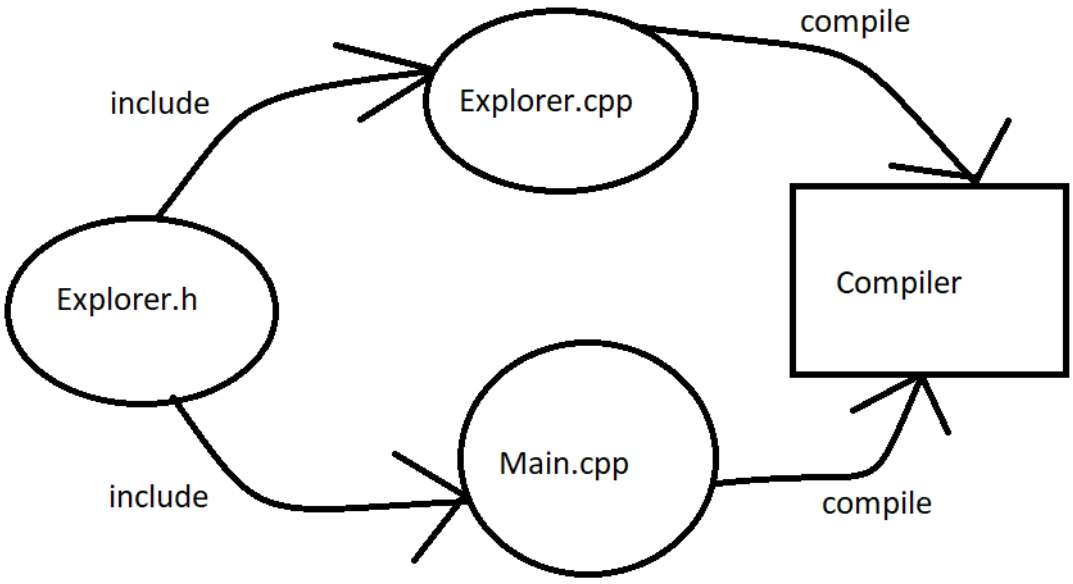

Figure 2 - OOP Implementation of the Urban Search and Rescue Mission Challenge

Both the Explorer and Follower robots share most of their features. Nonetheless, it is the Explorer robot that navigates to the assigned targets as well as gathering data from the environment. Meanwhile the Follower only travels to those areas before returning to base. As a result, the Follower is functionally a subset of the Explorer which limits the OOP portion exclusively to the Explorer, since the navigation only needs move_base and establishing action client-server all of which can be done from the main.cpp.

Hence, to break down the actions of the Explorer such as listening to incoming messages from sensors as well as publishing commands to the robot, the following key features were added in the source files. These are subscribing to the /fiducial_transforms (to access camera information) and the /amcl_pose (to obtain precise location of the robot), and publishing velocities to explorer/cmd_vel (to instruct the robot to rotate after reaching target and allow reading of the marker). As a result, by establishing callbacks to the subscribers, and posting velocity in the desired time, an Explorer instance was created from the Explorer class for easy access and simultaneous update of the Explorer attributes during the actual mission.

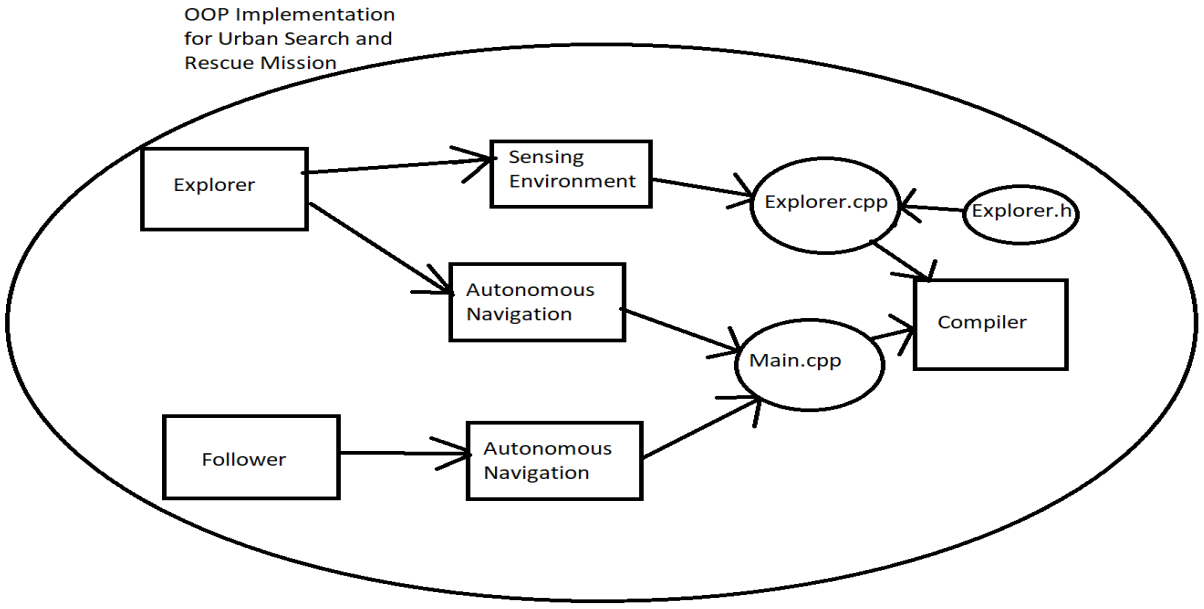

Figure 3 - Explorer and Follower in OOP approach of US&R Mission.

After structuring the code in such a manner, the program is broken down into the following subparts to complete the rescue mission successfully.

i) Target Retrieval

The first step of the program is to retrieve the ArUco markers' location in the map. Each marker has a spot in the environment where it could be detected by the Explorer. The x and y coordinates are retrieved using getParam() function from /ArUco_lookup_locations which is given by the professor . Each target coordinate is stored as an array of two numbers. The challenge in this part is how to retrieve a parameter array using XmlRPC::XmlRpcValue.

ii) Moving to Target

To implement the navigation algorithm, the move_base package is used. This package works by linking the global and local planner to achieve its navigation task. Moreover, this package allows the Explorer to perform autonomous navigation by giving the Explorer a specific goal location. There are three types of planners that work together to navigate the bot. “Planner” takes care of avoiding walls and finding the shortest path. “Global Planner” defines a global path in between stationary obstacles. “Local Planner” is used for collision avoidance and making sure to follow the global plan as closely as possible.

iii) Detecting ArUco Marker

The next task to do after moving to the target is to detect the ArUco marker in the map. The Explorer robot is instructed to slowly rotate using geometry/Twist messages and publish to the explorer/cmd_vel topic. To identify the ArUco marker, the Explorer is subscribed to topic /fiducial_transforms and a new frame is published on topic /tf if an ArUco marker is found. The

Explorer looks for the marker while rotating in place. The marker is detected using the camera in the camera rgb optical frame.

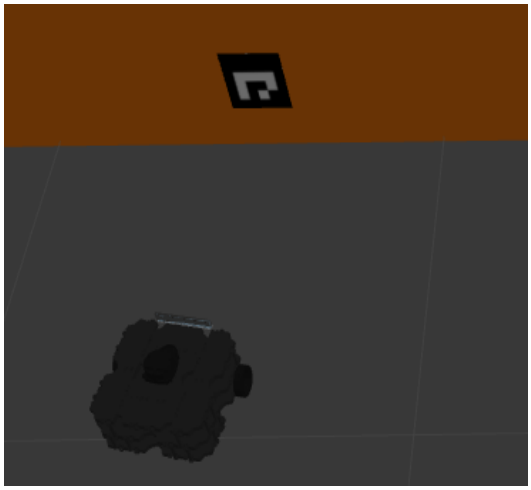

Figure 4 - Explorer seen here detecting ArUco marker

iv) Broadcaster

After the Explorer robot has detected the ArUco marker, a frame called marker_frame is created through a broadcaster since the ArUco marker frame is in the robot's camera frame instead of the map frame. This new frame is published on the /tf topic to appear in RViz's TF tree.

Within the callback function for /fiducial_transforms, if the robot is in the target location, ArUco marker is detected, and its x, y distances from the robot camera frame to the marker frame is less than 2 meters, the marker_frame is published.

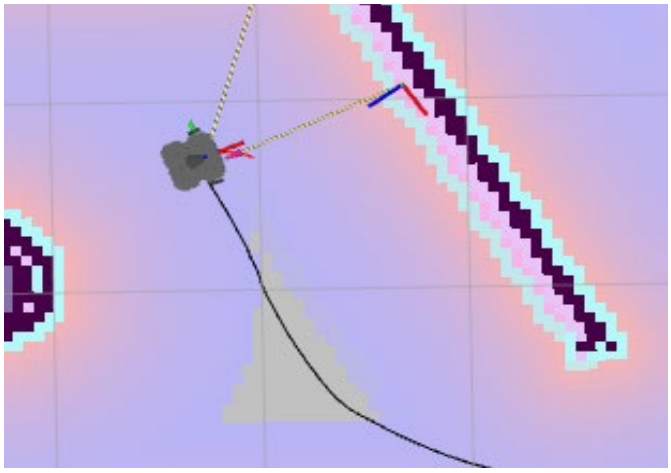

Figure 5 - Explorer broadcasting the marker frame as a child frame from the camera rgb optical frame.

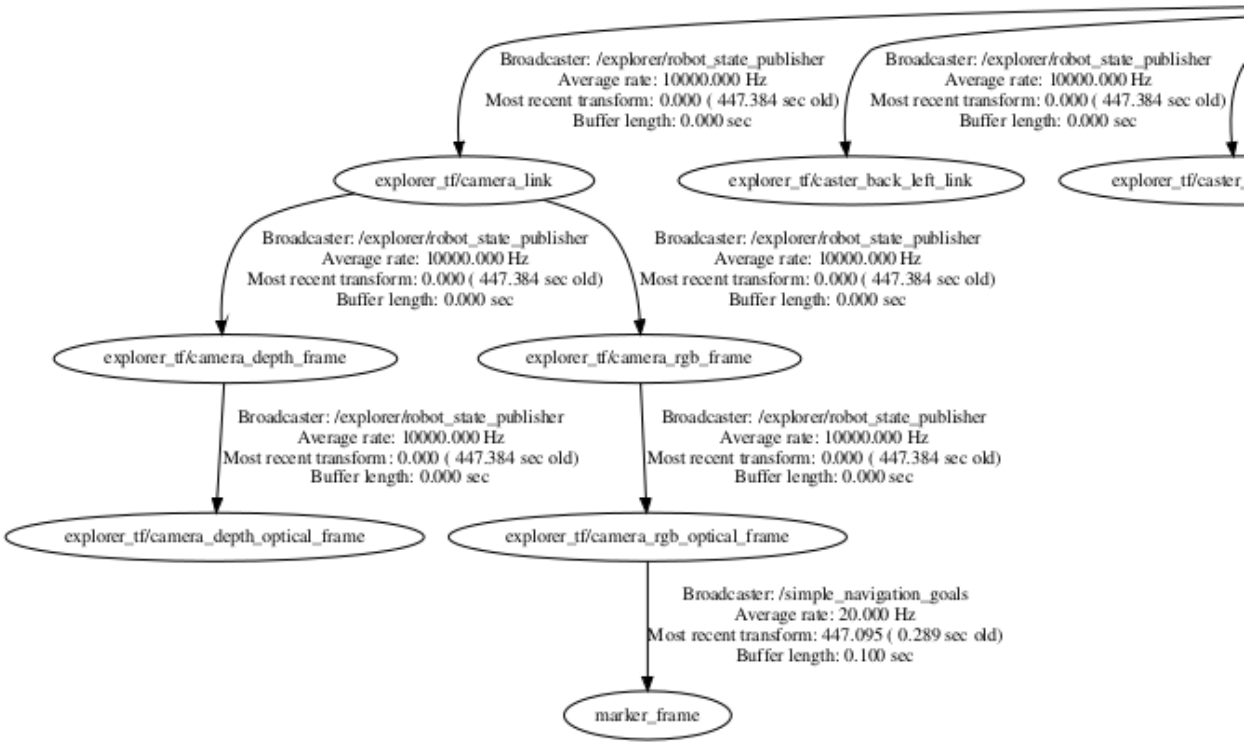

Figure 6 - Tf view_frame displaying after marker_frame has been broadcasted as a child_frame of camera rgb optical frame.

v) Listener

Once the marker frame has been broadcasted on the camera frame of the robot, the Explorer then transforms the marker frame, which is a child frame of its camera frame, on to the map frame using tf2_ros::Buffer class and store the x and y locations of the transformed frame in a local array using geometry_msgs::TransformStamped class. Since the Follower can't access the ArUco marker, as it is located on the wall, there is a need to add a tolerance to the x and y coordinates in the listener array. As a result, 0.4m has been added to the x and y locations of the ArUco marker towards the direction of the robot.

```
 INFO] [1639539493.958061619, 451.498000000]: Created Broadcaster
 INFO] [1639539493.958498984, 451.498000000]: Hooray, robot reached goal
 INFO] [1639539493.958623144, 451.498000000]: idx: 0
 INFO] [1639539493.958716963, 451.498000000]: Rotation Began
 INFO] [1639539493.958833068, 451.498000000]: ARUCO FOUND
 INFO] [1639539493.959214230, 451.498000000]: m_transforms: -1.46548 , 3.54049
 INFO] [1639539493.959320982, 451.498000000]: fiducial_id: 0
 INFO] [1639539493.959459412, 451.498000000]: Position in map frame: [-1.86548,
.94049,0.203998]
```

Figure 7 - Explorer storing location of the Fiducial ID=0, posted on ROS INFO Stream.

vi) Final Steps

After the Explorer robot has reached the final target, it is instructed to go back to the start position of the maze (-4, 2.5) utilizing the move_base package again. The Explorer robot is then shut down including all publishers and subscribers. The Follower robot does not need any information from the Explorer nodes to move to the target locations since it's using the local array that has the information already stored.

Finally, the Follower utilizes the ID targets from the local array to move to each target using the move_base package. When all 4 targets have been successfully traveled to, it is instructed to go back to the start of the maze at position (-4, 3.5). This concludes the program.

vii). Overall Program Flow

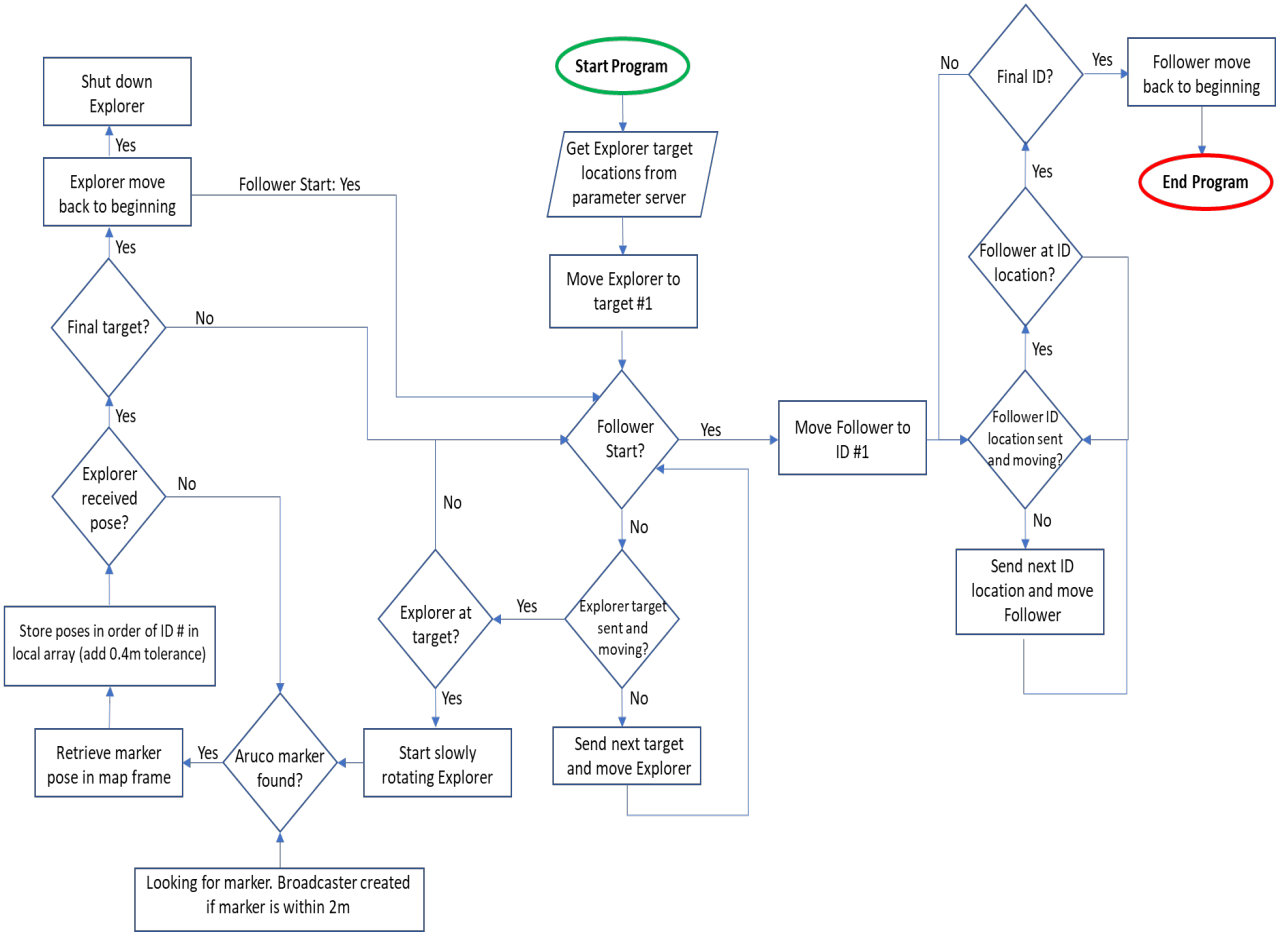

Figure 8 - Flowchart of the main program algorithm.

3. Challenges

During the code implementation, we faced many challenges. But with some troubleshooting and testing, the issues were fixed. Some of those challenges are explained below.

- **Detecting Wrong Markers**

One challenge that we faced was the Explorer robot would detect markers that were not were not intended to be detected yet. For example, after reaching the target location, the robot would rotate slowly and accidentally pick up an ArUco marker that is far away in the distance across the maze.

To avoid this, we added a condition in the fiducial callback function to only set the flag that indicates an ArUco marker was found when the robot has successfully reached the target location and both translational x and y distances from the robot's camera frame to the marker on the wall are less than 2 meters.

- **Creating marker_frame Continually**

Although marker_frame (created by Broadcaster in fiducial callback) was only created while the Explorer robot is less than ~2 meters away from the ArUco marker, the marker_frame was being created and the transforms being updated multiple times through the listener affecting the final Follower robot's path. A common occurrence of this would be when the Explorer has completed going to all 4 target locations and is on the way back to the beginning of the maze. The Explorer sees the target #4 ArUco marker in its camera view for many iterations as it moves backwards back to the start position. During this time, the marker_frame is created continually.

To fix this issue, we added a condition to only listen for the marker_frame after m_aruco_found flag is set to true when setting the broadcaster. The m_aruco_marker flag is set to true and the flag to indicate Explorer has reached the goal position (m_goal_reached) was set to false only after the

broadcaster has created the marker_frame. Then both m_aruco_found flag was set to false (reset for next target) after confirmation that the desired transform was found.

After the Explorer robot is back at the beginning of the maze, all subscribers are terminated, and this helped to not accidentally create another marker_frame and log of a fiducial ID that might be seen by the robot while moving back to the starting position.

- **Data Type Mismatch**

As the turn for reaching the targets came to the Follower, we found that the Follower was not moving at all. Upon further investigation, we determined that the issue emanated at the instance the stored location data - following the transformation of the marker frame to the map frame for the respective targets - was passed to the Follower when calling the move_base functionality. Since, we wrote the program to pass listener coordinates to the array as a XmlRpc::XmlRpcValue type, Follower's MoveBaseGoals was not interpreting the data type accurately as XmlRpcValue is designed to store and accept multiple data types. As recommended in the project guidelines, this class of data type was successfully used to store the location of the ArUco lookup spots in the given environment. However, because we assumed that XmlRpcValue is suitable to write and store data in an array, the transformed values were stored as such in an array. Therefore, after hours of recasting the various arrays used into double - with the suggestion from our instructor, we fixed the misused data type in the listening process. Once this change was made, the Follower navigated to its designated targets as expected. Therefore, we learnt that being consistent with data types and knowing whether the selected data type is suitable for each scenario will alleviate such hindrances when working on other projects in the future.

4. Project Contribution

Hae Lee Kim:

- Explorer rotating and ArUco marker detection.
- Broadcast and listener implementation.
- Follower tasks.
- Final report.

Yoseph Kebede:

- Retrieving parameter location for Explorer.
- Reaching target destination with Explorer and Follower via move_base.
- Implementing OOP for Explorer environment sensing (camera, broadcast, listener).

Mohammad Baaqail:

- Documenting the code (doxygen).
- Writing the report and doing final edits.
- Reviewing Code package.

5. Course Feedback

Overall, the class was very informative, and a lot was learned about both C++ programming and its application to robotics using ROS. Even for a beginner in C++, the course is a good introduction and the lectures are organized well into the major fundamental topics. For the C++ programming aspect, the content coverage was really helpful and thorough, but more opportunities for practice would have been useful. Same for the ROS content, it would have been helpful to get 1 or 2 more shorter assignments to apply the content learned in class before jumping into the bigger projects.

The Course has increased my confidence in C++. For someone like myself, who has never written code in C++ but has worked in one way or another in Python, MATLAB, or Java, this course is designed to help students who are not new to programming quickly get a grasp of and begin coding in intermediary levels using C++. More importantly, the Mouse maze project was challenging enough for students to implement the important skills taught in the class right away. In addition to C++, the course also does a good job at covering ROS from a high-level theoretical standpoint to actual interaction with the environment by allowing students to work on pre-designed example packages. Additionally, the use of VSC in the class is another big skill that I took away which is sought for across the robotics industry. Moreover, the final project was a very challenging but eye-opening scenario on how to apply ROS on real word problems. However, the ROS portion of the class felt really rushed as the content far outweighed the number of lectures assigned which along with the lack of practice exercises for each new concept taught left me confused on where to begin on the final project for about a week, with the class feeling more like an incomplete ROS course than a C++ only class. So, I recommend that either the final project is not set too high of a bar for students, or the course dissects the ROS contents into chewable small portions.

The professor is one of the best as he is always available to help, a great interactive lecturer, and has the students' best interests in mind! LaTeX presentations are well done!