Final Project

Introductory Robot Programming

December 16, 2021

Instructors:
Z. Kootbally

Group:
10

Semester:
Fall 2021

Course code:
ENPM809Y

Students:
Bhargav Kumar Soothram

Koundinya Vinnakota

Madhu Narra Chittibabu

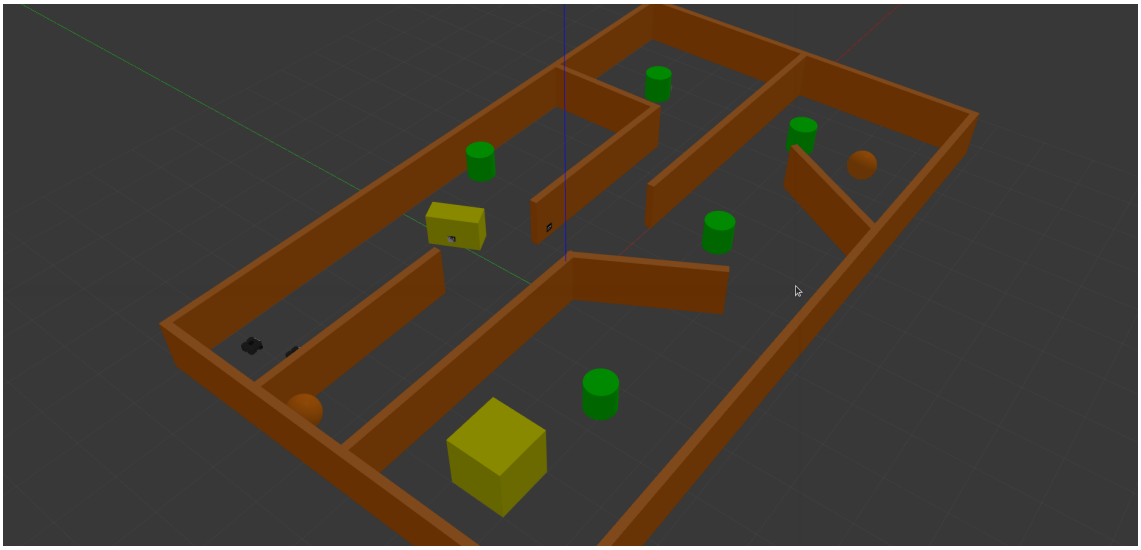
Contents

1 Introduction

The project is inspired by the challenge of autonomous robotics for Urban Search & Rescue (US&R). Disaster response is one of the most important use cases for robotics where in we strongly intend to prevent human intervention. Even when human intervention is necessitated for such an application, the scenario should be well read and mapped. When a disaster has occurred, there is debris everywhere (such as a collapsed building) and unconscious human victims of the disaster need to be rescued. In such a scenario, a robot is used to explore the environment, build a map of the disaster inflicted area and place markers in the map where the victims are located. This map is then given to trained First Responders who use it to go into the building and rescue the victims.

In this project, we try to simulate such a situation - an explorer robot (a turtlebot) uses a map of the building to find victims. Instead of simulated victims, we use ArUco markers (squared fiducial markers) here. Another turtlebot, called the follower is then used to fetch the victims. There are four ArUco markers provided at different points in the map and the explorer has to go through these four locations. It then rotates until it detects an ArUco marker and stores the pose of this marker in the program. Once all four markers are found, it goes back to its initial position. To simulate the task of rescuing victims, the follower robot will then visit each of the markers in the order of their marker IDs. Once all the markers have been visited, the follower returns to its initial position too.

We use an ROS program to simulate such a scenario and our code executables are generated using the C++ language.



2 Approach

The project starts with the explorer robot setting out to search for the four ArUco markers. The locations of the ArUco markers are stored on the parameter server (parameter name `/aruco_lookup_locations`) and the first task was to retrieve these values from the parameter server. For accomplishing this, we used the getParam() function to fetch the parameter array from the server and the values obtained are stored in an Xml::Rpc array. We then stored these values in an array for easier access any point later.

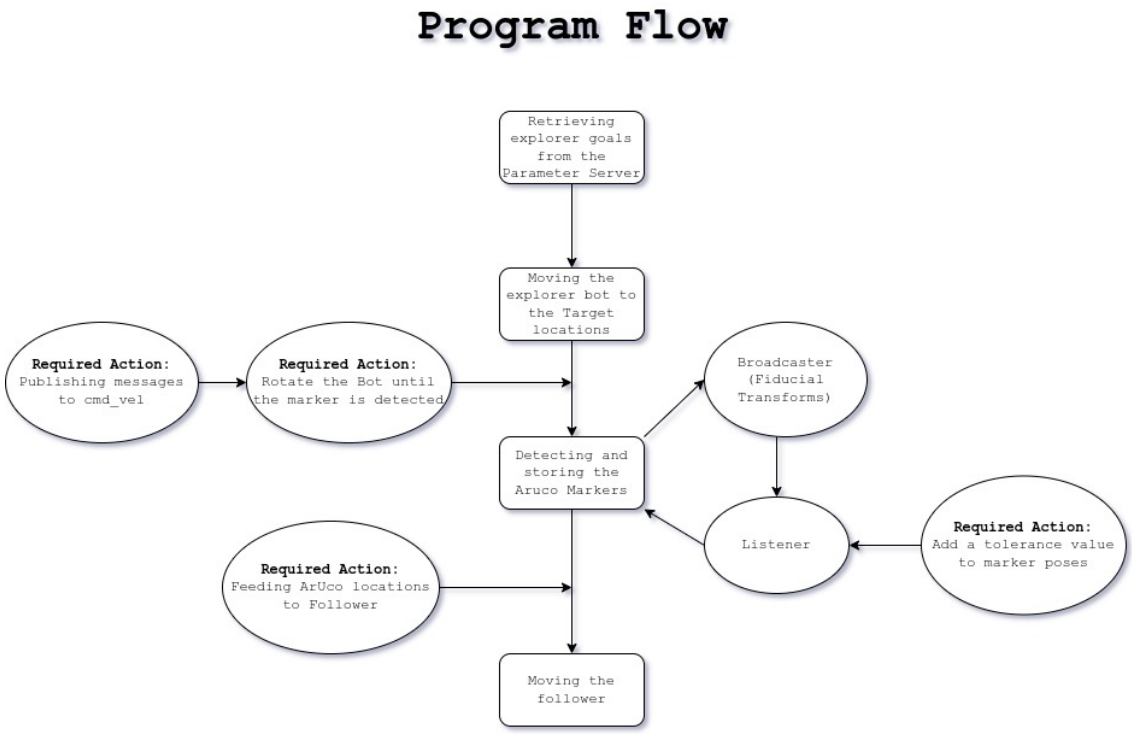
Now we have everything we need to assign the custom goals to our explorer and we go ahead with updating the explorer goals section. To assign the custom goals to the explorer, we loop over all the values in the parameter array. The robot now will move to each of the locations given and our next task would be to store the ArUco marker locations. The robot is moved around in the map using the `move_base` client for ROS.

For storing these locations, we first need to identify the ArUco markers. The node `aruco_detect` publishes on the topic `/fiducial_transforms` and our task of storing the marker locations is accomplished by reading data from the `/fiducial_transforms` topic.

We first publish values of marker locations on to a new frame `/marker_frame` on the `/tf` topic using a broadcaster present in the callback function `fiducial_callback`. This new frame is a child of the explorer's camera frame (`/explorer_tf/camera_rgb_optical_frame`). It is important to note here that the follower has to later visit the marker locations that get detected by the explorer and the goals for the follower need to be set in the `/map` frame. Since our marker locations are stored in the `/explorer_tf/camera_rgb_optical_frame`, we need to transform the marker locations in the `/explorer_tf/camera_rgb_optical_frame` to `/map` frame. This task is accomplished by a listener function.

Our listener function takes two input arguments, transform buffer and a zero-initialized ArUco location array and updates the array with the transformed ArUco marker locations in the `/map` frame. The follower robot then takes the locations from this array and moves to the markers in the order for their fiducial IDs. The figure below gives a glimpse of our program flow - how everything works in the code!

2.1 Program Flow



We now go through all the custom functions written in the code, one by one, along with the algorithm we used written as pseudocode.

2.2 `fiducial_callback` - Callback function and Broadcaster

This function is called each time ros::Spin() is called in the loop and it looks for all the messages on the topic `/fiducial_transforms`. If it finds the marker (the array associated with the key `transforms` in the topic messages is non-empty), it takes all the data (linear and angular) from the message and stores it for the listener to process the transformation. We have also declared an integer value to store fiducial ID here, which will be of use when listener comes into the picture.

2.3 `listener`

The function `listener` takes two input arguments : transform buffer and an empty ArUco marker location array. Firstly, it transforms all the data in the `/explorer_tf/camera_rgb_optical_frame` into the `/map` frame. But sometimes what happens is that the robot reads a coordinate that is inside the wall/ some other impossible location for the path planner to plan a path. So, we use a tolerance value here to reduce "pull" the follower goal to a point in the map for which the planner can plan a path. Then, based on the fiducial IDs, these marker location values are assigned to the location array given as an argument to the function. This array will then help guide the follower to the marker locations.

2.4 Pseudocode

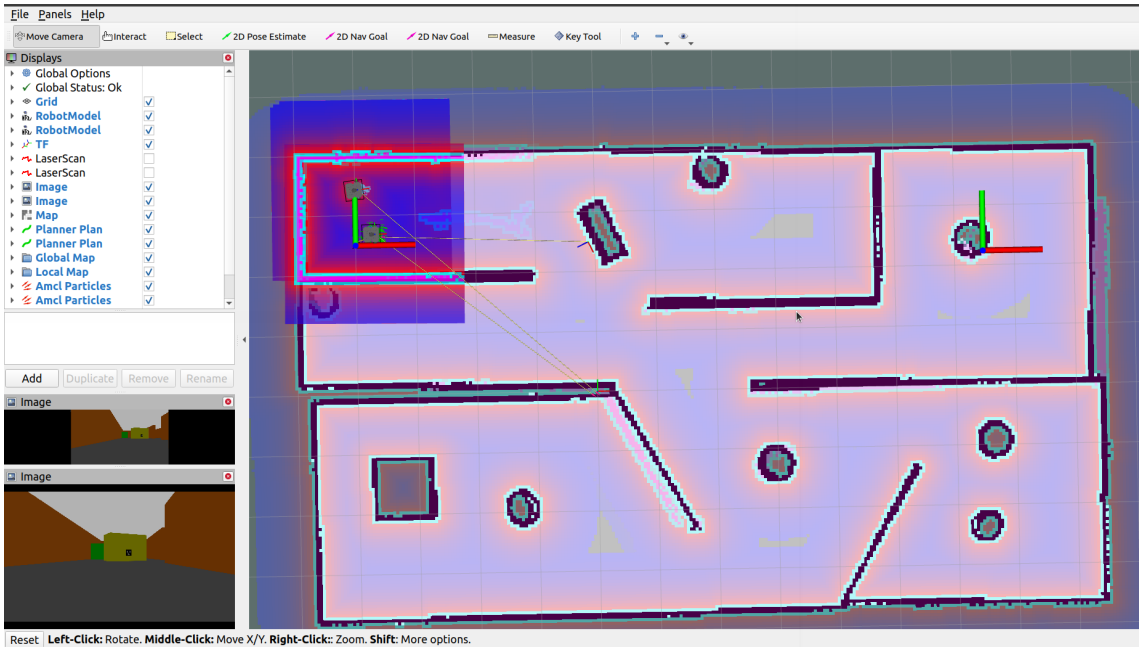
Algorithm 1 US&R

retrieve and store values of the target locations on the parameter server
create a `MoveBaseGoal` object for the explorer and follower goals.
for $explorer\_goals = 1, 2, 3 \ldots$ **do**
    assign the target locations as explorer goals
end for
while `ros::ok()` **do**
    send the goals to the explorer client
    if *explorer_goal_reached* **then**
        publish messages to `/explorer/cmd_vel` to rotate the explorer
        `ros::spinOnce()` to initiate the callback function
        `listen()`, to get the transform between `/marker_frame` and `/map` frame
        store the marker pose in the array in the order of fiducial IDs
    end if
    if *explorer_returned_home* **then**
        assign the stored marker poses as follower goals
    end if
    if *follower_reached_home* **then**
        `ros::shutdown()`
    end if

3 Challenges

We highlight some of the most important challenges that we faced in this section.

- The biggest challenge we faced whilst doing the project was that the tolerance value was distorting the follower goal location. Since we initially gave a constant tolerance value of 0.5, there were certain locations obtained for the follower that the planner could not build a path for. To correct this, we employed an averages approach, where we take the take the mid point between the wall position and the explorer position as the goal position for the follower. This solution proved to be effective in the follower moving properly to all the goal positions.
- The next big issue that is worth mentioning is that of the explorer robot being unable to rotate properly when it reaches a goal position. The issue was, with both `move_base` as well as our publisher publishing values to the topic `/cmd_vel` topic, the values given by `move_base` were overwriting the values given by the publisher. To, solve this issue, we gave a `loop_rate()` command that would stop `move_base` from overwriting the values on `/cmd_vel` topic.
- As mentioned earlier, we used an array to store the values of the fiducial marker locations - these get stored when the explorer visits each marker. We initially tried to use the same array to give the follower goals, but the follower did not move as intended because the order in which the explorer and follower visit the targets is different. This issue was resolved by initializing another array for the follower goals and this array would be indexed in the order of the fiducial IDs.



Target locations inside objects/walls

4 Project Contribution

The following table summarizes the contributions of each of the team members in a pictorial way:

TEAM / TOPICS	Parameter Retrieval	Broadcaster	Bot Rotation	Listener	Tolerance Handling	Code Optimization	Code Documentation	Report
Koundinya Vinnakota	☑	☑	☑	☑	☐	☑	☑	☑
Bhargav Kumar Soothram	☑	☑	☐	☑	☑	☑	☐	☑
Madhu Narra Chittibabu	☐	☑	☐	☑	☑	☑	☑	☑

5 Resources

- Basic ROS Functionality : `http://wiki.ros.org/ROS/Tutorials`
- Move Base Client : `http://wiki.ros.org/move_base`
- `ROS::spin()` : `https://answers.ros.org/question/357705/stop-rosspin/`
- Buffer : `http://docs.ros.org/en/jade/api/tf2_ros/html/c++/classtf2__ros_1_1Buffer.html`