Final Project

Introductory Robot Programming

December 16, 2021

Students:
Pradip Kathiriya

Jeffin Johny Kachappilly

Hemanth Joseph Raj

Instructors:
Z. Kootbally

Group:
7

Semester:
Fall 2021

Course code:
ENPM809Y

Contents

1 Introduction

In today's world, robots are used for a wide variety of tasks and in various environmental conditions. The need for robots has only grown in the last two decades. One reason for this growth is the unique advantage to perform tasks that are considered either dangerous or fatal for a human being to attempt.

One such unique area where autonomous robots find their application is in Urban Search and rescue (US&R). When any natural calamity strikes, these US&R robots are sent into these unknown environments to go around and map the area and locate the wounded humans, identify their location in its map and then bring back that information to the base where a team of First Responders then go and rescue the people from those identified spots.

In our final project, we have been given a similar real world application of US&R where we have to apply such an action using our robots (turtlebot3). The world in which we need to perform the US&R has been provided to us. Here there are four ArUco markers that replace the actual victims and they are placed at four, initially unknown, locations in the world.

The task is broadly divided into two parts and we make use of two robots, namely the explorer and the follower.

1. First we send out the explorer in the unknown environment to find the location where the ArUco markers are placed in the environment. The explorer in its task of exploring the world must visit and remember the location of all the ArUco markers and then come back to the home position with the information it gathered and pass it on to the follower robot.

2. Once the explorer is back, the information about the location of the ArUco markers is passed on to the follower robot, which then uses it to visit each ArUco marker and once done, return back to the home position and thus the task is completed.

2 Approach

We took a two layered approach towards solving our final project. In the first level of problem solving, we approached it like a *Procedural Programming* problem. Subsequently, once a working solution was obtained, we divided our program into different class and utilized concepts of *Object Oriented Programming* and rewrote our solution as such.

2.1 Solving the problem with Procedural Programming Paradigms

After reading the problem statement and understanding what needs to be done, we reached to the conclusion that we need to approach every major task with an appropriate function.

1. **Accessing target locations from the Parameter Server**
   These target locations which are retrieved from the parameter server give us a location with high probability of finding an ArUco marker in its immediate vicinity. We retrieve these locations to our program by using the getParam() function and store the values in an stack data structure. Each elements of this stack has two values, each denoting the X and Y values in the world frame. We create four such getParam() functions since we need to retrieved the target locations for all the four ArUco marker locations. Once all the target locations are obtained we save them in a stack to access them one after the other.

2. **Moving to a target location**
   Once the target locations are retrieved and stored in the stack, we pop them out one by one and send the data (the x,y co-ordinates of the target location) in them to the Explorer robot. By making the use of the move_base package, we give commands to the explorer to visit these location one by one.

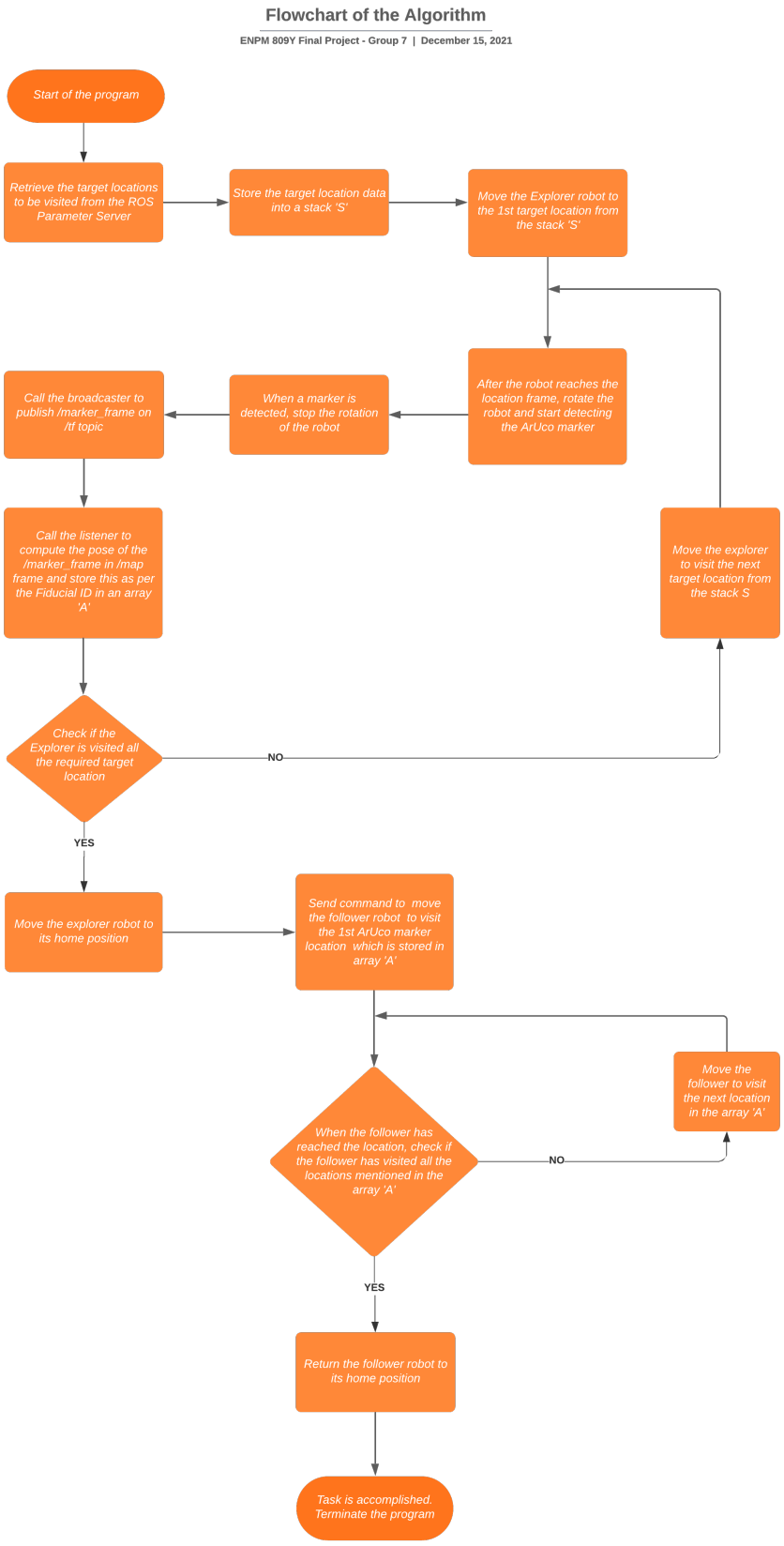
3. **Detecting the ArUco marker and broadcasting the location**
   Once the robot is in the vicinity, command is given through publisher to topic cmd_vel for the robot to rotate counter-clockwise around the Z-axis and using the laser scanner to identify the exact location of the ArUco marker. Once it is identified, a frame is generated in the camera frame and publish on /tf_ topic. This task will be accomplished by broadcaster. This broadcast is sent out only when the marker is detected by the camera frame of the explorer robot.

4. **Listening to the ArUco frame and transforming it to /map frame**
   Now the listener is made to subscribe to the /tf_topic and receive the locations of the ArUco markers with respect to the camera frame and then transform them to the world frame, i.e the map_frame. This location will be stored in an array.

5. **Visit remaining target locations** Now that the information is stored in the array. The explorer's task at that particular location is done. Now command is sent to the explorer through the move_base to go forward by accessing the new target locations of the remaining ArUco markers and repeating steps 2, 3, and 4 for each target location. Now the location of each of the marker frame is stored in the final array. Once all the four target locations are visited and the array updated. Command is given to the explorer robot to return back to its home position in the map.

6. **Visit each location with follower robot**
   Now this array is passed onto the follower robot. Now the robot accesses the location of each ArUco marker in sequential order using the move_base package. Once the robot visits each location, it is marked as visited and the follower robot moves on to the next ArUco marker. Once all the locations are visited and marked as completed, the follower robot returns back to its base location. Thus completing our project.

Now that we have approached and solved the problem through the use of conventional Procedural programming methodologies (uses of functions etc.), we then move on to adapt our through Object oriented concepts (uses of classes, objects and pillars of OOP like abstraction and encapsulation etc.)


Flowchart of the Algorithm
ENPM 809Y Final Project - Group 7 | December 15, 2021
Start of the program
Retrieve the target locations to be visited from the ROS Parameter Server
Store the target location data into a stack 'S'
Move the Explorer robot to the 1st target location from the stack 'S'
After the robot reaches the location frame, rotate the robot and start detecting the ArUco marker
When a marker is detected, stop the rotation of the robot
Call the broadcaster to publish /marker_frame on /tf topic
Call the listener to compute the pose of the /marker_frame in /map frame and store this as per the Fiducial ID in an array 'A'
Check if the Explorer is visited all the required target location
NO
Move the explorer to visit the next target location from the stack S
YES
Move the explorer robot to its home position
Send command to move the follower robot to visit the 1st ArUco marker location which is stored in array 'A'
When the follower has reached the location, check if the follower has visited all the locations mentioned in the array 'A'
NO
Move the follower to visit the next location in the array 'A'
YES
Return the follower robot to its home position
Task is accomplished. Terminate the program

2.2 Pseudo code for the algorithm

Below given is the pseudo code of the algorithm we have followed to solve this problem.

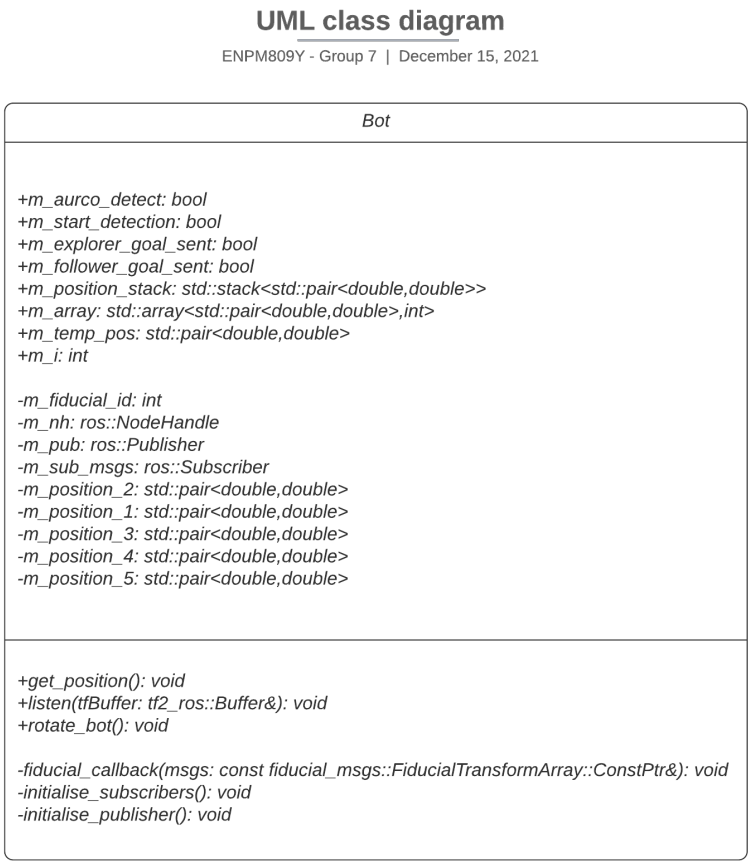
```
// s -> Stack of location to be visited by the explorer to detect marker. The last element in s is the home position.
// explorer_goal -> object to send request to /move_base
// e_goal_sent -> Boolean initialized as false
// f_goal_sent -> Boolean initialized as false
// start_detecting -> Boolean initialized as false
// broadcaster -> Function to create child frame of ArUco marker in the camera frame and publishes it to /tf tree
// listener -> Function to convert ArUco marker's pose in the /map frame
// aruco_detect -> Boolean initialized as false
// follower_goal -> Object to send request to /move_base
// my_array -> Listener store location to be visited by follower in this array. The location is as per ArUco ID. Last location in my_array is its home position

while(ros::ok){
  while(s not empty){
    temp_pose = s.top(); //temp_pose to store top element of s
    explorer goal x position = first elements of the temp_pose;
    explorer goal y position = second elements of the temp_pose;
    if(not e_goal_sent){
      send goal to client server; //server will send command to move_base to move explorer to goal position
      e_goal_sent = true;
      s.pop();
    }
    if(explorer reach goal){
      start_detection = true;
      while(true){
        rotate robot;
        call broadcaster;  // broadcaster will create marker_frame as a child of camera frame and publish it on the topic tf
        call listener;      //listner will transform the marker frame in the map_frame

        if(aruco_detect){
          break;
        }
      }
    }
    start_detection = false;
    aruco_detect = false;
    e_goal_sent = false;
  }
  if(explorer visited all the locations){
    while(i<5){
      follower goal x position = first elements of the my_array[i];
      follower goal y position = second elements of the my_array[i];
      if(not goal_sent){
        send goal to client server;
        f_goal_sent = true;
      }
      if(explorer reach goal){
        f_goal_sent = false;
        ++i;
      }
    }
  }
}
```

2.3 UML diagram of the Class

Here we have mentioned the UML diagram of our class *Bot*



2.4 Important points to be observed in the algorithm

1. Since the ArUco marker is attached to the wall the follower robot cannot reach the exact location of the ArUco marker. So we have offset the ArUco marker location so the follower can reach the said location.

2. The follower robot has to visit the ArUco markers as per the ArUco marker ID.
Below is the pseudocode to accomplish the aforementioned points 1 2.

```
// pseudocode to detect Fiducial ID
// pseudocode to place ArUco code 0.5m away from wall
// msg -> message subscribed from /fiducial transform topic
// f_fiducial_ID -> public attribute defined inside the class
void fiducial_callback(&msgs){
  //some code
  Z co-ordinate of marker frame in camera frame = (msg) -> (Z member of
    translation composit field - 0.5);
  f_fiducial_ID = (Fiducial ID from the received messages);
  // some code
  }

```

```
void listener(&tfbuffer){
  // some code
  transform marker_frame into map using lookup transform and store them into trans
     x and trans y;
  if (f_fiducial_ID == 0){
    m_array.at(First element).First = trans x;
    m_array.at(First element).Second = trans y;
  }
  if (f_fiducial_ID == 1){
    m_array.at(First element).First = trans x;
    m_array.at(First element).Second = trans y;
  }
  if (f_fiducial_ID == 2){
    m_array.at(First element).First = trans x;
    m_array.at(First element).Second = trans y;
  }
  if (f_fiducial_ID == 3){
    m_array.at(First element).First = trans x;
    m_array.at(First element).Second = trans y;
  }
}
```

3. The explorer robot must detect the ArUco marker only after reaching the target location for the said marker.
   Below is the pseudocode to accomplish the task mentioned in task 3

This is a code snippet from the main.cpp file.

```
// Start broadcasting when reached at the marker detection location
// m_start_detection -> Boolean attribute of class bot
if (explorer reach goal){
  start_detection = true
  while(true){
    rotate bot;
    call broadcaster;
    call listener;
    if (aruco_detect){
      break;
    }
  }
  start_detection = false;
  aurco_detect = false;
  e_goal_sent = false;
}
```

This is a code snippet from the class definition (Bot.cpp)

```
void fiducial_callback(&msgs){
  // some code
  if (msgs not empty & m_start_detection is true){
    // some code
    broadcast new frame on tf topic;
    // some code
  }
}
```

3 Challenges

- **Challenge:** When the robot reaches the first ArUco marker, it doesn't assign it a frame, but it does so for other markers. Sometimes the ArUco marker frame assignments are outside the world's boundaries.
  Action Taken: After debugging we have found that our subscriber was storing 1000 messages in its queue. So we then proceeded to reduce the size of the queue to 1. We understood the importance of the queue size when defining the publisher and the subscriber after facing this issue.

- **Challenge:** Since the marker is attached to the wall, the follower robot didn't reach the marker point.
  Action Taken: When assigning the frame to the ArUco marker in the camera frame, the frame is assigned 0.5 m away from the wall.

- **Challenge:** A terminal error which reads, "Quarternion has length close to zero... discarding as navigation goal".
  Action Taken: We found out that we missed the transformation of the rotation part (in the Quarternion) from the ArUco frame to the camera frame in the broadcaster. We added the necessary transformation.

- **Challenge:** The explorer robot was detecting the marker before reaching the goal position. After detecting the first ArUco marker, when the robot was moving ahead to the next target location, since the fourth ArUco marker was within the field-of-view of the robot, it was detecting the fourth ArUco marker even before it reached the fourth ArUco marker's target location (this is even before it reaches the target location of the second ArUco marker).
  Action Taken: We have set a flag so that the broadcasting of the ArUco frames will happen only after the explorer robot reaches the said ArUco marker's target location.

4 Project Contribution

1. **Pradip:**
   - Retrieving the ArUco marker's target location from the ROS parameter server
   - Detecting the ArUco marker and generating the respective frames for the ArUco markers and publishing them through broadcaster and listener
   - Storing location of the marker frame as per the Fiducial ID
   - Moving the follower to visit Aruco marker as per the received Fiducial ID
   - Class definition and documentation
   - Report writing
2. **Jeffin:**
   - Moving the explorer to visit each ArUco marker location
   - Detecting the ArUco marker and generating the respective frames for the ArUco markers and publishing them through broadcaster and listener
   - Storing location of the marker frame as per the Fiducial ID
   - Moving the follower to visit Aruco marker as per the received Fiducial ID
   - Class definition and documentation
3. **Hemanth:**

- Moving the explorer to visit each ArUco marker location
- Detecting the ArUco marker and generating the respective frames for the ArUco markers and publishing them through broadcaster and listener
- Storing location of the marker frame as per the Fiducial ID
- Moving the follower to visit Aruco marker as per the received Fiducial ID
- Report writing

5 Course Feedback (optional)

The course was really good and we learnt a lot about C++ and ROs from this course. However, we think that below points would have help use gaining more understanding about the subject.

- More RWA assignment.
- Small exercises that help student to learn how to implement standard algorithm and data structure in c++.

6 References

1. For Publisher and Subscriber
2. for tf
3. for tf