Final Project

# Introductory Robot Programming

December 15, 2021

*Instructors:*
Z. Kootbally

*Students:*
Nicholas Novak(118450933)

Orlandis Smith (11849307)

Jerry Pittman, Jr. (117707120)

*Group:*
5

*Semester:*
Fall 2021

*Course code:*
ENPM809Y

## Contents

## 1 Introduction

Autonomous robotics used in Urban Search  Rescue (USR) to explore unknown environment to find injured victims. Two waffle turtlebots will be used. One (*Explorer*) will use the map provided to find victims (indicated by ArUco markers) and send that data to the second turtlebot (*Follower*) to fetch/triage the victims in order of their fiducial IDs representing their health status.

## 2 Problem Statement

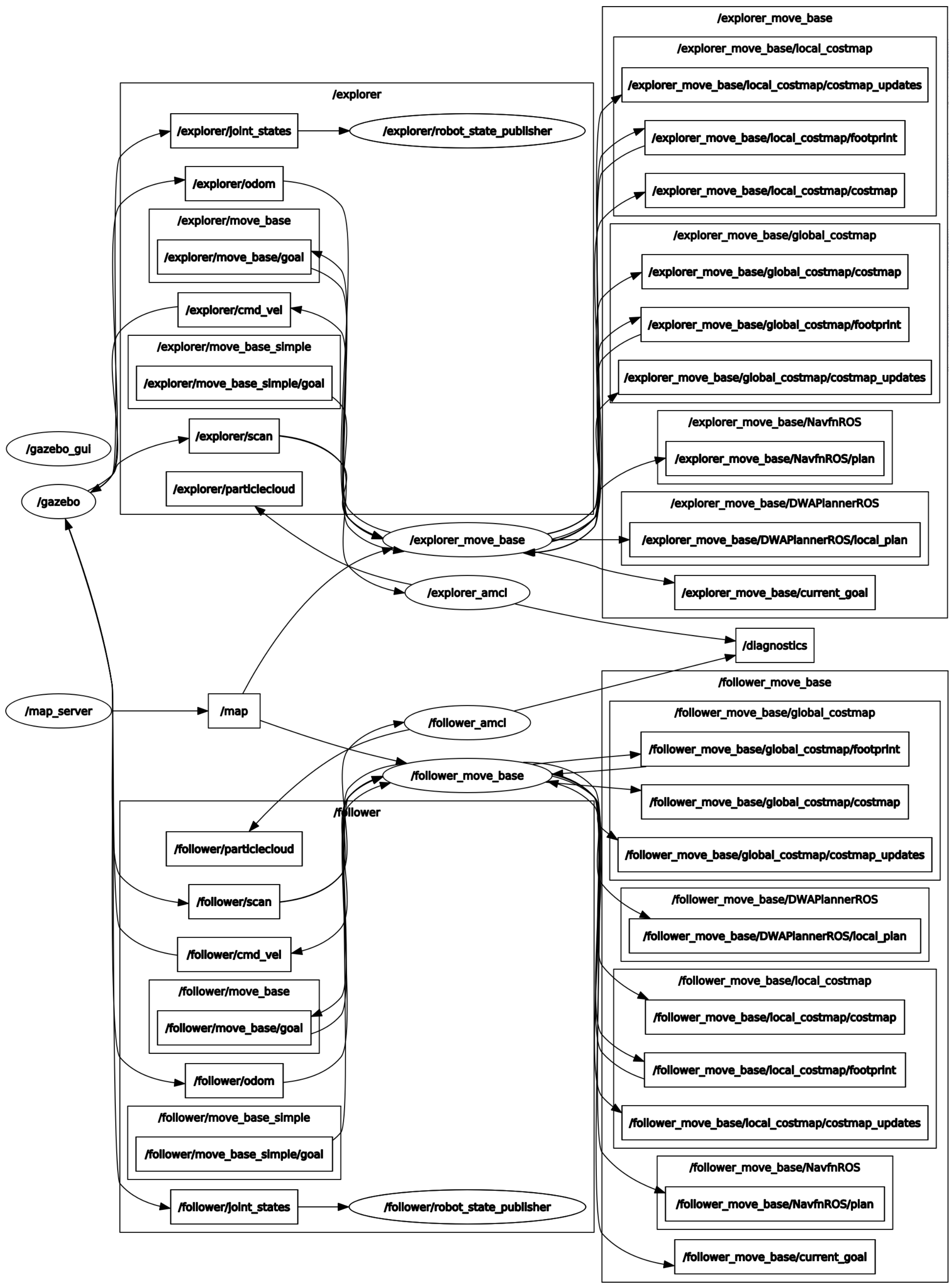
Given a scenario where two robots are assigned to perform search and rescue in an unknown environment. The robots have separated tasks with one being the explorer and the other being the follower. Being that new field is unknown the robots each have their own responsibilities, one to perform a mapping of the area and search for targets, while the other waits for the command to enter once the established coordinates has been delivered.

## 3 Design Overview

The code in doing this assignment is on a Linux OS platform and is implemented using C++. The robots used in this assignment are already preconfigured with ROS associated packages for movement, visualization, and configuration. The user loads ROS distribution packages of ROS Noetic and/or Melodic to allow for accessibility of needed header files and included functions for providing calls to the bots interfaces.In addition, the packages also allow for the user to interface with the ROS server that store pre-configured subcomponents such as camera and laser scan configuration. For example, camera parameters which could be used for adjusting of pixel size, image size, calibration of frame alignment is already established. The user would have to ensure that the launch file and header file includes the necessary variables for making calls to the server.

## 4 Explorer and Follower

The Explorer bot is configured with the Waffle configuration. The bot is packaged with a camera, laser scanner, and a PID controller for control of angular and linear speeds. The user developed in the software code an interface to allow for the control of the speeds during driving and turning maneuvers. The designed scan and camera parameters are left unknown, for the user is using the pre-defined settings within the ROS packages.


/explorer_move_base
/explorer_move_base/local_costmap
/explorer_move_base/local_costmap/costmap_updates
/explorer_move_base/local_costmap/footprint
/explorer_move_base/local_costmap/costmap
/explorer_move_base/global_costmap
/explorer_move_base/global_costmap/costmap
/explorer_move_base/global_costmap/footprint
/explorer_move_base/global_costmap/costmap_updates
/explorer_move_base/NavfnROS
/explorer_move_base/NavfnROS/plan
/explorer_move_base/DWAPlannerROS
/explorer_move_base/DWAPlannerROS/local_plan
/explorer_move_base/current_goal
/explorer
/explorer/joint_states
/explorer/robot_state_publisher
/explorer/odom
/explorer/move_base
/explorer/move_base/goal
/explorer/cmd_vel
/explorer/move_base_simple
/explorer/move_base_simple/goal
/explorer/scan
/explorer/particlecloud
/gazebo_gui
/gazebo
/explorer_move_base
/explorer_amcl
/diagnostics
/map_server
/map
/follower_amcl
/follower_move_base
/follower_move_base
/follower_move_base/global_costmap
/follower_move_base/global_costmap/footprint
/follower_move_base/global_costmap/costmap
/follower_move_base/global_costmap/costmap_updates
/follower_move_base/DWAPlannerROS
/follower_move_base/DWAPlannerROS/local_plan
/follower_move_base/local_costmap
/follower_move_base/local_costmap/costmap
/follower_move_base/local_costmap/footprint
/follower_move_base/local_costmap/costmap_updates
/follower_move_base/NavfnROS
/follower_move_base/NavfnROS/plan
/follower_move_base/current_goal
/follower
/follower/particlecloud
/follower/scan
/follower/cmd_vel
/follower/move_base
/follower/move_base/goal
/follower/odom
/follower/move_base_simple
/follower/move_base_simple/goal
/follower/joint_states
/follower/robot_state_publisher

Rosgraph Figure

## 5 Approach

- We used Object-Orientated Programming (OOP) to solve the problem. We used the "bot_controller" package as a shell and made two classes: "Follower" and "Explorer" to represent the two robots. The values of the fiducial IDs and their related positions were gathered and stored in the correct location in the *m_fiducial_callback* function of the Follower class in Follower class variables which were then pulled and saved in local variables in the main function.

## 6 Target Environment

- A simulated environment representing a collapse building is the focused area for scan and detect. A map of the area is initially performed by the Explorer bot alone, with the bot driving around the room with its laser scan operation mode only. Laser scanning prevents the bot from colliding with obstacles and the bot can safely detect its distance for timely avoidance. This is where the user defined controller comes in handy because it helps to slow down the bot's speed and increase its speed once it has found a safe opening. Once the Explorer bot is finished driving around the target environment a map of the region is saved and stored in the bots memory.

## 7 Marker Detection

- Now the Explorer bot has the layout of the map area. The next task for the Explorer bot will be for to identify participants and/or victims, fiducial markers, and learn of the defined area which is calculated on the robot internally from reference frame (center point on ID marker) to world frame (robots center origin).

## 8 Robot Initialization

Initialize the Explorer, Follower, and Camera Class to allow for the communication of the function to the nodes defined in the launch file and use of the remote server access through XML-RPC. The user can activate the defined functions for robot actions and broadcast the messages over the ROS service. To prove the success of code and the bots are being initialized correctly a message for Explorer and Follower the user will see, for example, "Waiting for the move_base action server to come up for Explorer". If this occurs then one of your robots did not spawn correctly. You should restart your computer, catkin clean, and catkin build then re-run.

## 9 Broadcaster Messages

Allows for the transmitting of robots position and orientation to the ROS service. The messages allow the user to detect when the robot is at the target or is lost. Once the robot has reached its target position, the user will be notified with "Hooray, explorer reached goal" notification. If the server is having unknown errors a repeat of the "Sending goal for explorer" message is sent to the user to alert them that there is something troubling it's movement or communication with the server.

## 10 Listener

Once all the proper commands/instructions has been sent to the Explorer bot, a "Sending goal for explorer" notification is sent over the ROS service for the user to know that Explorer is awaiting the goal information to proceed. This gives the Explorer the first prescribe location of the first marker. A confirmation of the bot reaching the goal is then followed by a "Hooray, explorer reached goal" message. The same designed structure is also common for the Follower bot with similar messages.

## 11 Target Identification

The developers are using the help of a built in ROS package library, Aruco Detection, to perform the settings for camera parameters with the camera coordinate frame already being calibrated with Z axis pointing outward. This allow the developers to set the image width and height in the XY direction. The targets are identified by a fiducial ID, represented by different shapes, which are also stored in the Aruco library. With the pre-defined markers, the developers minimize time the need for additional camera source code development

## 12 Algorithms

**Algorithm 1** Leader

```
procedure READ_WAYPOINTS(waypoints_path)
    procedure READ_YAML(waypoints_path)
        $waypoints\_path \leftarrow loadwaypointspath$
        try
        $waypoints \leftarrow readwaypointsfromthefile$
        except
        $stdout \leftarrow exception$
        return waypoints
    end procedure
    $pose \leftarrow extractpositioncoordinates$
    $frame\_orientation \leftarrow extractframecoordinates$
    $pose.extend(frame\_orientations)$
    $output \leftarrow positioncoordinates$
    return output
end procedure

procedure LEADER_CLIENT(coordinates)
    $client \leftarrow actionlib.simpleactionclient(move\_base, MoveBaseAtion)$
    client.wait_for_server()
    $goal \leftarrow MoveBaseGoal()$
    $goal.target\_position \leftarrow fromcooardinates$
    $client.send\_goal(goal)$
    $wait \leftarrow client.wait_for_result()$

    if $not\ wait$ then
    $stdout \leftarrow Actionservernotavailable!$
    $stdout\_shutdown \leftarrow Actionservernotavailable!$
    else
    $returnclient.get\_result()$
end procedure

procedure MAIN
    $rospy.init\_node() \leftarrow initializealeadernode$
    $locationsreadwaypointsfromwaypoints\_path$
    try
    for $location, coordinates.in\ locations$ do
      end
      if coordinates is not None then
    $result \leftarrow leader\_client(coordinates)$
    if result then
    $stdout \leftarrow locationreached$
    except
    $stdout \leftarrow exception$
end procedure
```

**Algorithm 2** Follower

```
procedure READ_ARUCO$_LOOKUP_LOCATIONS(waypoints\_path)$ if $coordinates$ is not $None$ then    RESULT $\leftarrow leader\_client(coordinates)$
    if DISTANCE$_to_goal > 0.05$ then    RESULT $\leftarrow \_client(coordinates)$


    $pose \leftarrow extractpositioncoordinates$
    $frame\_orientation \leftarrow extractframecoordinates$
    $pose.extend(frame\_orientations)$
    $output \leftarrow positioncoordinates$
    RETURN OUTPUT


    procedure MAIN
        $rospy.init\_node() \leftarrow initializealeadernode$
        procedure FOLLOWER_CLIENT(COORDINATES)
            $client \leftarrow actionlib.simpleactionclient(move\_base, MoveBaseAction)$
            CLIENT.WAIT_FOR_SERVER()
            CLIENT.WAIT_FOR_RESULT()
            $goal \leftarrow MoveBaseGoal()$
            $goal.target\_position \leftarrow fromcooardinates$
            try
            $m\_posit \leftarrow readwaypointsfromtransformStampedoffiducial\_msgs :: FiducialTransformArray$
            $m\_fid \leftarrow readfiducial\_idfromfiducial_msgs :: FiducialTransformArray$
            for $iterator.in$ $fiducial\_ids$ do
    end
    Save $m\_posit$ array based on $fiducial\_id$ read and store fiducial IDs into class array $m\_fid$. if COORDINATES is not None then
            Send $follower$ robot to $fiducial\_id$ waypoints using a For loop and $go\_to\_goal$ function.
            Send $follower$ robot to Starting Position (-4,3.5) once reaches Fiducial ID 3 waypoint.
            ROS::SHUTDOWN();
        end procedure
```

## 13 Challenges

- One of the challenges we encountered was how to save the fiducial IDs where they can be accessed. We decided to go with public class variables m_fid (array for fiducial IDs) and m_posit (array for fiducial ID marker positions) to get updated in the fiducial_callback function of Bot_Controller and then called in the main function and saved into local variables for iterative use.
- We had issues with using various msgs and topics and it was determined that we didn't include the header files initially.
- We also had issues with inherited classes so shifted our *Follower* and *Explorer* classes to be independent classes and removed all *bot_msgs* and *bot_controller* files.
- Once we was able to get a successful build we had errors and issues with rosrun of our main.cpp on our included header files and doxy comments. This was due to use not including "_node" at the end of our executable name, matching the cmakelists.txt file.
- Another issue we had was that we was initially subscribing and publishing to the same topics (i.e *odom*) vice */explorer/odom/* and */follower/odom/*.

- Another issue we had was the robot(s) would get to a location and publish their next goal locations but not move. This was since the "*BOT_goal_sent* was not reset to "false" after reaching a goal for the next goal to be sent in the next iteration of the respective *for* loop.

## 14 Project Contribution

- Orlandis worked on pulling the data from the camera and the m_fiducial_callback functions as well as the research into the various msgs used.
- Jerry worked on the *Follower* class member and tasks. He also worked on debugging the main for integration of the two robots.
- Nicholas built the *Explorer* and *ArucoNode* header and cpp files. He also worked on debugging for the main file in order to effectively drive the explorer to a goal, receive said goal, and detect the arUco marker at the goal.
- We all worked on the report, doxygen documentation, and cleanup of unused code.

## 15 Resources

- We used the provided bot_controller and final_projectmain files to build our files from.
- Our Github repository: https://github.com/jpittma1/809Y_final_project_Group_5.git

## 16 Course Feedback (optional)

- Nicholas Novak: Now that I have completed two courses with Dr. Kootbally, I know that his classes are both challenging and rewarding. The hours spent debugging assignments really opened my eyes to more effective techniques of comprehending and modifying C++ files. I look forward to diving more into these topics in the near future, and applying my ROS skills in a future job.
- Jerry Pittman, Jr.: I really liked the organization of the course and lecture material as well as the hands-on nature. The lecture material was so good that use of the textbooks was rarely required of me except during researching proper C++ etiquette and when working on the projects. The professor and TA were both very accessible and helpful whenever we had doubts throughout the semester making the class feel personalized and containing far less students then it actually did. I also found the explanation and teach of ROS much better than what is available on various forums or outdated web textbooks. I wish I had the ROS portion before my ENPM662 course which was very ROS heavy and didn't seem to know what I was doing until right before turning in that final project based on what I learned in this class. The other complication was that we was using python vice C++ but I would like to be fluent and competent in both languages in addition to MATLAB.
- Orlandis Smith: I did not realize until I took this course that I would be in for a journey, and well it was a journey. I came into the course thinking that it would be a basic programming course in C++ but I was wrong. I actually learned a lot and had a chance to learn Linux which is really important in industry. What I like most about the course is that it challenges you and a new developer will have the opportunity to know how to deal with real world problems, that of which are needed when going into a tough career. Thanks to Dr. Zeid, he has been patient with me and has been helpful in understanding that I do not have the strongest background in coding. I really appreciate this class and look forward to how the upcoming semesters will be.