Final Project

Python Applications for Robotics

Students:
Alec Lahr
Dhyey Patel
Jeet Patel
Mrugesh Shah

Instructors:
Z. Kootbally

Group:
2

Semester:
Spring 2021

Course Code:
ENPM809E

1 Introduction

This final project consists of two turtlebots named leader and follower. With the use of Visual servoing. Our aim is to create the desired leader-follower formation. Visual servoing, also known as vision-based robot control and abbreviated VS, is a technique which uses feedback information extracted from a vision sensor (visual feedback) to control the motion of a robot. Here waffle model of turtlebot is used as shown in figure below, since it is equipped with depth camera which publishes RGB images.

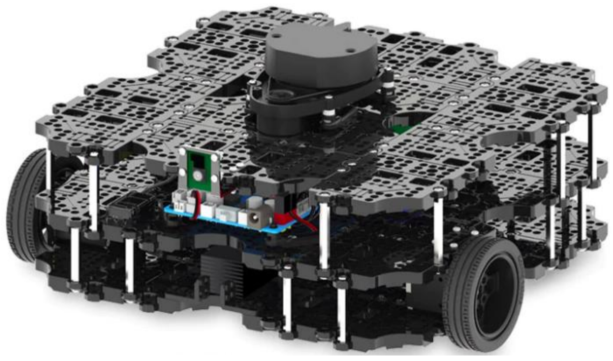

Figure-1: Turtlebot-3 Waffle Model

We can see a live stream of the camera in the tool called Rviz. RViz (ROS Visualization) is a powerful robot 3D visualization tool.

The follower camera will be used to perceive the leader in the environment and follows it by keeping some safe distance. The environment used here is aws-robomaker-small-house-world consists of the gazebo world as shown in figure below.

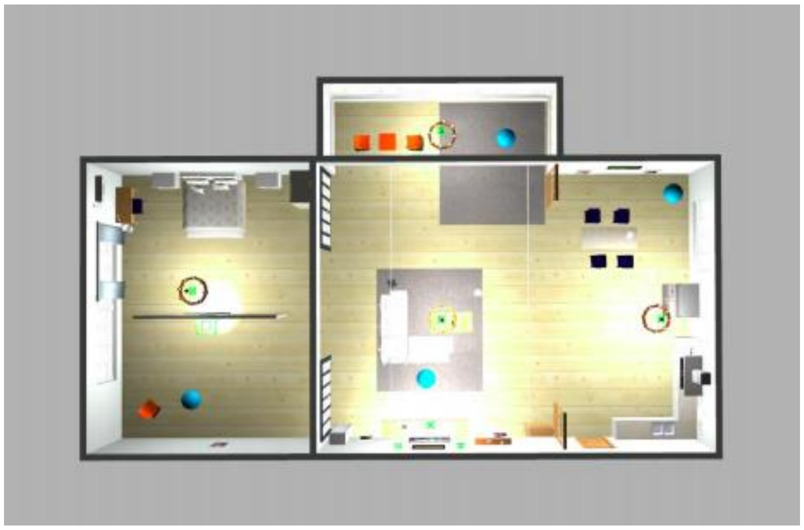

Figure-3: AWS small house world

To make the follower to follow the leader, Aruco marker is used, which is spawned on the leader. The follower will try to detect the marker using and will try to follow it based on manipulation of robot control actions. The marker attached to the leader and follower launched in the Gazebo world is shown in figure below.

Turtlebot is autonomously navigated in the above map using Adaptive Monte Carlo localization (AMCL). AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map. AMCL takes in a laser-based map, laser scans, and transforms messages, and outputs pose estimates. On startup, AMCL initializes its particle filter according to the parameters provided.

The move_base is used to drive both robots, to task to the goal position. The leader reads the YAML file in which the waypoints are already defined for each location of the house. The leader will move to the living room, recreation room, kitchen, and bedroom in successive order after running the lead.py in the package. Parameter server is updated accordingly which represents the pose of the location leader is heading to.

The goal of the follower is to follow the marker by maintaining a safe distance of 0.7 m. The follower will first try to detect the marker which is spawned on the leader. If the marker is not detected, the follower will rotate 360 degrees. If the marker is still not found, the follower will read the parameter server and start moving towards the expected goal location. If the marker is detected, the follower will stop the exploration, compute the goal location, and reaches the goal location.

2 Approach

2.1 Mapping

The task was to get the YAML file of the map to the server which we had created using the map_server package’s map_saver executable at the given path. Below is the figure that we generated from the gmapping tool provided by ROS.

Now that we have the map, we must put it to the server. For that the map_server package has an executable named map_server which takes the YAML file of the world. The goal is to get the RVIZ, the visualization tool, to get the map that we had saved.

2.2 Marker

The leader will now have to get the marker linked to it. We are using Aruco marker in this project. To attach this marker, we must link it with the URDF file of the robot. The submitted package’s spawn_marker_on_robot takes care of it.

2.3 Moving the Leader

The leader node in the submitted package will read the YAML file of the room recorder and get the data to process it and store it to the dictionary format using the YAML module in python. The leader will follow the following flow to execute the tasks.

The leader will first publish that goal to the /move_base_simple as well as update the parameters server with the current pose of the goal of the leader. The parameter will be used by the follower to go to that location if the follower cannot find the aruco marker.

Since, the /move_base_simple topic is using the type of msg that does not support the current status of the bot if it has reached the goal state or not, we have to use thresholding by getting the current pose of the leader and comparing it to the goal position assigned to it.

If the goal has been reached or is within the threshold, then go to the next room stored in the YAML file. Keep repeating this and after reaching all the rooms, go to the first room again and keep looping until the node is shut down.

2.4 Moving the Follower

The follower will have to follow the leader if it sees the Aruco marker in the camera field, and if it does not find the aruco marker, it has to look for it by rotating for specified time and if it is not found yet, it will call the subroutine to get the current goal of the robot over the parameter server and go to that location. The following flow diagram is for the follower.

The follower will subscribe to /fiducial_transform topic and get the transform between the arco marker and the /camera_rgb_optical_link. Then this TF transform is broadcasted to the /follower_tf/marker_goal. Now, we have to get the transform between this and the broadcasted frame and the map. This will give us the current coordinate of the leader. Now, we have to use the pipeline explained in moving the leader to get the follower to reach the leader location with the offset of 0.5 units.

But, if the aruco marker is not present in the camera frame, then the follower has to rotate until the specified time and the just access the parameter server to get the data of the current goal of the leader and start going there. Note, that if the follower finds the leader while going to the goal taken from the parameter server, it will exit from this subroutine and start following the leader again with the pose of arucomarker.

2.5 Pseudocodes

2.5.1 Leader

```
Locations ← YAML file of records
Go_to_goal("room"):
Room_pose ← Locations("room")
Move_base_simple ← Room_pose
Publish (Move_base_simple)
Set_param("leader_goal", Room_pose)
While True:
    Current_pose ← tf.transform("/leader/odom, /leader/base_footprint)
If Current_pose - threshold < Room_pose < Current_pose + threshold:
    Break
Main ():
    For location in Locations:
    Go_to_goal(location)
```

2.5.2 Follower

```
Go_to_marker():
    Tf.broadcast("/fiducial_transform", "leader_marker")
    Marker_pose ← Tf.transform("/Map", "/leader_marker")
    Go_to_gole("Marker_pose")

GO_to_leader_goal():
    Goal_pose ← get_param("leader_goal")
    Go_to_goal("Goal_pose")
Go_to_goal(pose_to_go):
    Move_base_simple ← pose_to_go
    Publish(Move_base_simple)
    While True:
        Current_pose ← tf.transform("/follower/odom,
        /follower/base_footprint)
        If Current_pose - threshold < pose_to_go < Current_pose + threshold:
            Break
Check_where_to_go():
    If marker_detected:
        Go_to_marker()
    Else:
        Go_to_leader_goal()
Main():
    While Not ros.isshutdown():
        Check_where_to_go()
```

2.6 Flowcharts

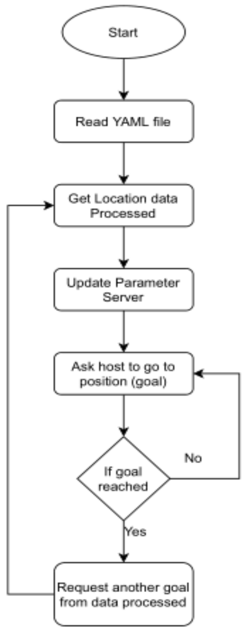
Start
Read YAML file
Get Location data Processed
Update Parameter Server
Ask host to go to position (goal)
If goal reached
No
Yes
Request another goal from data processed

Figure-5: Leader Flowchart

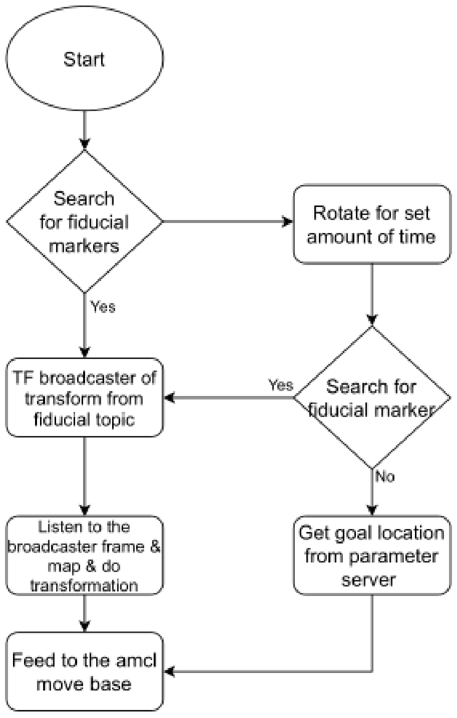
Start
Search for fiducial markers
Rotate for set amount of time
Yes
TF broadcaster of transform from fiducial topic
Yes
Search for fiducial marker
No
Listen to the broadcaster frame & map & do transformation
Get goal location from parameter server
Feed to the amcl move base

Figure-6: Follower Flowchart

3 Challenges

- The foremost challenge was to spawn the robots for which we just did git cloning into our terminal.

- The next challenge was to read the YAML file and take location points from it. In our YAML file the location points were stored as nested dictionary so based on that we extracted the location points and gave it to the leader.

- The next challenge was to attach the Aruco marker on the robot, to solve it firstly we commented.

```
<node if="$(arg has_marker)"
name="spawn_marker_on_robot"
pkg="final_project_x"
type="spawn_marker_on_robot"
output="screen" />
 single_robot.launch, comment or delete the following:
<node if="$(arg has_marker)"
```

```
name="spawn_marker_on_robot"
pkg="final_project_x"
type="spawn_marker_on_robot"
output="screen" />
```

- In spawn_marker_on_robot, we replaced the line `robot_name = rospy.get_namespace()[1:-1]` with `robot_name = "leader"`. Then, started multiple_robots.launch: `roslaunch final_project_x multiple_robots.launch` and in another terminal we ran spawn_marker_on_robot: `rosrun final_project_x spawn_marker_on_robot`, and it worked.

- Another major challenge was the move_base_simple was not able to provide the goal status and because of that even if the robot was at its goal location it was not able to detect it. So, the next thing we tried was to use move_base_action_lib but in this case the server was not responding and were stuck with the same problem. Finally, we tried to solve using move_base_simple with ODOM, but the data was not reliable enough, so we did tf. Transform between frames and that worked completely fine.

- The next big challenge was faced while detecting the aruco marker. Initially we were not able to detect the Aruco marker and there were some issues in detect Aruco function, where were getting wrong transformations. By visualizing in Rviz it was found that follower/camera_rgb_optical_frame was not over the leader turtlebot as it should be as shown in the figure below.

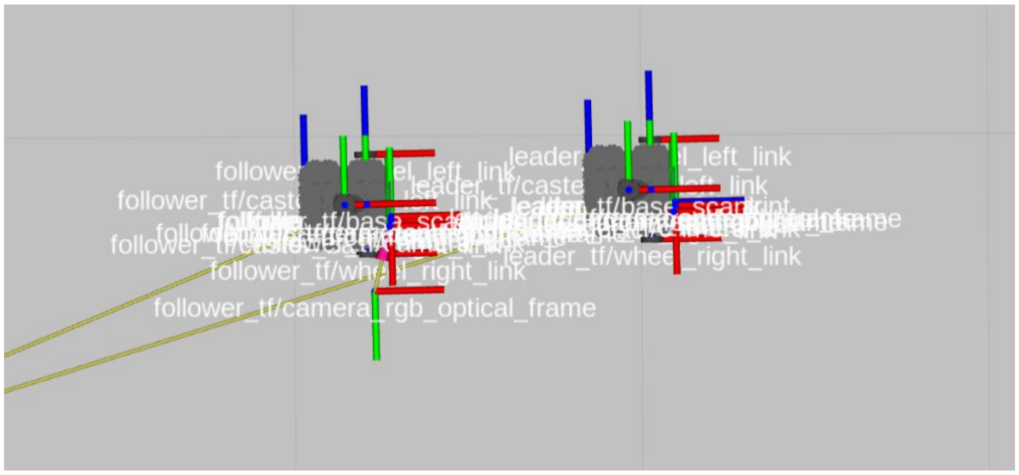
follower_tf/camera_rgb_optical_frame
follower_tf/wheel_right_link
leader_tf/wheel_right_link
leader_tf/wheel_left_link

Figure-6: Frame Transformation

- The problem was not with our broadcaster or tf lookup. We used rostopic echo to listen directly to /fiducial_transform and the raw data matched what we see in rviz. We moved the leader robot and aruco around in gazebo to see how the echo'd messages would change, and they didn't change as we would expect. Then we made some changes in our broadcaster and we also changed (trn.x, trn.y, 0) to (trn.x, trn.y, trn.z) which worked fine.

4 Project Contribution

- We decided to break this project in two parts, part1: for leader, read the yaml file created in RWA3, which contains location information, and tasks the robot to reach each location in successive order and part 2: task the follower to follow leader while keeping a safety distance of 0.5 m. and work in team of two to work on part 1 and part 2 of the project.
- Part 1 and 70% of the report: Mrugesh & Dhyey.
- Part 2 and 30% of the report: Alec & Jeet.

5 Resources

- wiki.ros.org/ROS/Tutorials

6 Course Feedback

- Overall, the course is very well structured. It starts with the basic introduction of python programming language and then explaining the core concepts of python and at the end course also covers OOP which is a very important aspect of programming.
- Moreover, right from the brief and detailed introduction of ROS framework to covering important topics of ROS and giving hands on projects in ROS the course is very helpful for someone who wants to get significant knowledge of python as well as understand ROS and get a good hands-on experience with ROS.

7 References

- “ROS/Tutorials - ROS Wiki,” Ros.org, 2020. http://wiki.ros.org/ROS/Tutorials.
- “Programming Robots with ROS: A Practical Introduction to the Robot Operating System: Quigley, Morgan, Gerkey, Brian, Smart, William D.: 9781449323899.