Fall 2022

Final Project

Introductory Robot Programming

December 16, 2022

Students:
119247556-Nishant Pandey

119256534-Rishikesh Jadhav

119398364-Kiran S Patil

Instructors:
Z. Kootbally

Course code:
ENPM809Y

Contents

1 Introduction

The following report is written for the final project of 809Y - Introduction to robot programming. The key highlights of the report consist of the simulation of turtlebot3 in Gazebo using fiducial markers for reaching the desired location. The project consists of multiple tasks which include writing a broadcaster, locating fiducial markers, and moving to the goal position. These steps are further explained in detail in the approach section of the report. The report also includes the challenges faced, contributions made, resources used, as well as the course feedback by all the teammates in the project.

2 Approach

The project to be made was divided into multiple sections and sub-sections, each of these segments had to be solved in a sequential manner to complete the task. The following steps were taken to complete the project:

1. The first step was to understand the package provided and read through all the topics and their types. Understanding various ROS2 commands was one of the first targets.

2. To execute all the sections it was imperative to understand the broadcaster, publisher and subscriber code syntax and how the code structure works.

3. Broadcaster code available in the material provided to complete the project was read thoroughly.

4. The odom_updater package was made. After which the broadcaster was written to connect the tree. Figure 1 below shows the final tree obtained.

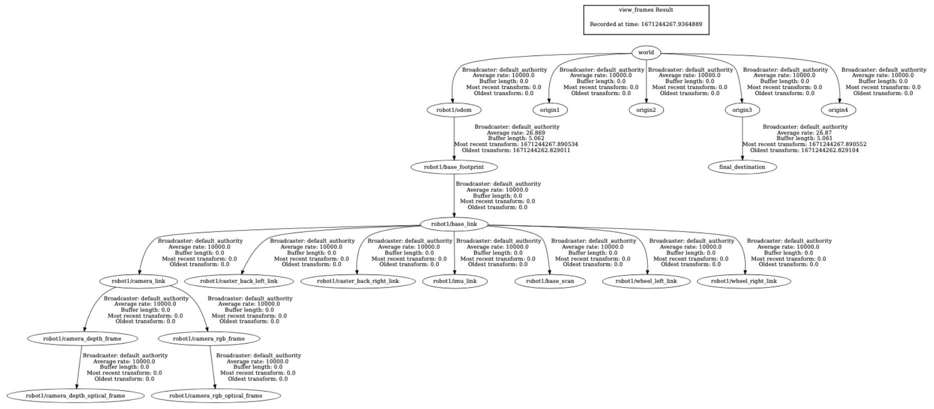

view_frames Result
Recorded at time: 1671244267.9364889
world
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
robot1/odom
origin1
origin2
origin3
origin4
Broadcaster: default_authority
Average rate: 26.869
Buffer length: 5.062
Most recent transform: 1671244267.890534
Oldest transform: 1671244262.829011
Broadcaster: default_authority
Average rate: 26.87
Buffer length: 5.061
Most recent transform: 1671244267.890552
Oldest transform: 1671244262.829104
robot1/base_footprint
final_destination
robot1/base_link
robot1/camera_link
robot1/caster_back_left_link
robot1/caster_back_right_link
robot1/imu_link
robot1/base_scan
robot1/wheel_left_link
robot1/wheel_right_link
robot1/camera_depth_frame
robot1/camera_rgb_frame
robot1/camera_depth_optical_frame
robot1/camera_rgb_optical_frame

Figure 1: final tree obtained

5. The non-static broadcaster consists of the class named frame publisher with node odom_updater (public) and a subscriber to the topic robot1/odom of type nav_msgs. The callback function odom_callback was created consisting of header and child frames as well as the pose of the robot. This entire process was done to retrieve the pose of the robot in odom. Necessary changes were made to the launch files to start the odom_updater.

6. After necessary verifications target.h file was modified to do further tasks. The first of these tasks was to start the node target_reacher. This node was created to make the bot reach the first goal where the fiducial marker can be found.

7. Initially multiple classes with different nodes were made to do most of the tasks and verify if the logic used was correct. However later they were grouped together in one class and executed all the tasks.

8. Next task was to subscribe to goal_reached, which will provide a message to verify if the goal is reached. The message provided is in bool format (true/false). If the message received from the goal_reacher is true then the robot is rotated about its z-axis (target_reacher.cpp) by publishing a twist message on /robot1/cmd_vel(linear velocity is null and angular velocity is 0.2). This process is done for the detection of the fiducial marker. The pseudo-code for the following is in Figure 2:

```
msg callback {read bool data from goal reached}
store data if data is 1(true)
implies goal is reached
Hence, start turning by publishing to topic /robot1/cmd vel
```

Figure 2: Psuedo code for goal_reached

9. To find the fiducial marker aruco_markers is subscribed to. Data obtained from the topic aruco_markers is marker_id. If the marker_id is obtained the bot stops and a switch case is applied (target_reacher.cpp) to retrieve the final destination parameters based on the id value. The pseudo code for this step is in Figure 3:

```
aruco callback {read vector of integer data from aruco markers}
if marker id is 0 or 1 or 2 or 3 then stop rotating by publishing m msg on topic /robot1/cmd vel.
switch case state case 0 then final destination.aruco i.x, final destination.aruco i.y is aruco 0 x and
aruco 0 y respectively.
case 1 then final destination.aruco i.x, final destination.aruco i.y is aruco 1 x and aruco 1 y
respectively.
case 2 then final destination.aruco i.x, final destination.aruco i.y is aruco 2 x and aruco 2 y
respectively.
case 3 then final destination.aruco i.x, final destination.aruco i.y is aruco 3 x and aruco 3 y
respectively.
These values are equated to goal.x and goal.y.
```

Figure 3: Psuedo code for aruco_marker's marker_id

10. The retrieved coordinates are in the Frame assigned to final_destination.frame_id. A frame is broadcasted (final_destination) in the frame (origin1). The retrieved goal coordinates from parameters final_destination.aruco_i.x, final_destination.aruco_i.y, (where i is the ID of the marker found by the robot) are the position for the frame origin1. To obtain the pose of the final destination in the Frame robot1/odom, the transform listener gets the transform between final_destination and /robot1/odom. Which is fed to the bot_controller to reach the target. The psuedo code for this is in Figure 4:

```
For broadcaster: odom callback{}
assigning header and child frame ids to frame id and final destination respectively
assigning position and orientation send transform to listener
For listener: listener callback{}
if the first goal is reached then
transform between final destination and /robot1/odom.
Make the bot go to the second goal by using the transformed value.
```

Figure 4: Psuedo code for broadcaster and listener in target_reacher

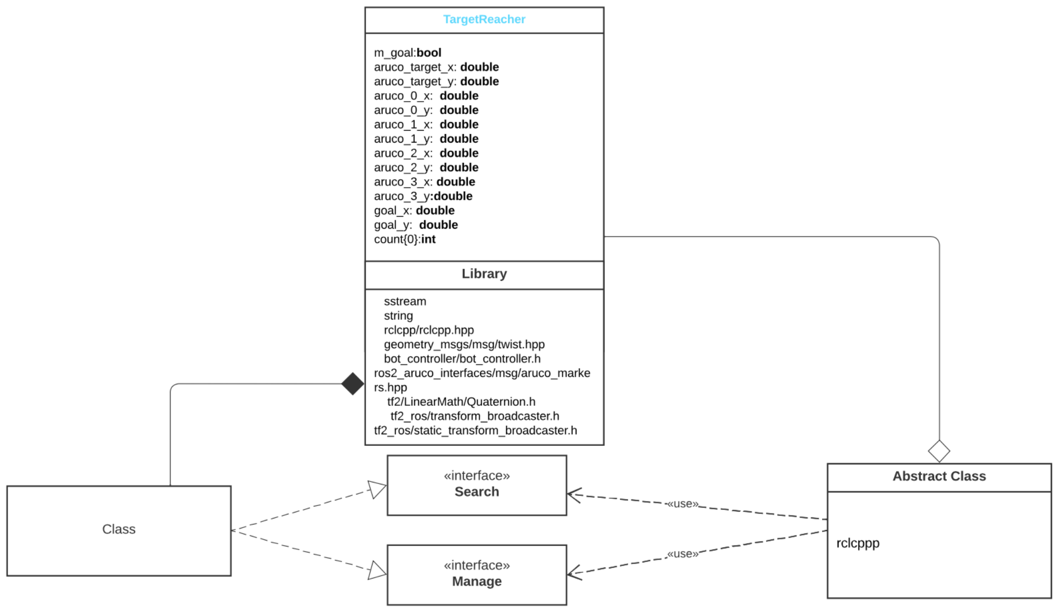


Figure 5: Class diagram for target_reacher

The class diagram to illustrate the attributes of the class target_reacher is above in Figure 5.

3 Challenges

1. Understanding the folder structure and file content to retrieve important information to use in the final code.
2. Building the packages successfully, as in the initial stages errors like building packages without including the dependencies in cmakelists and package.xml were encountered.
3. After writing the broadcaster and subscriber to robot1/odom , In the beginning, the trees were not connecting which lead to the subscriber not being able to retrieve the pose.
4. Extracting the data from the fiducial marker was a little tricky.
5. Computing the pose of the final destination in the /robot1/odom frame, writing the target reacher broadcaster and listener in the final stages of the project.

4 Contributions to the project

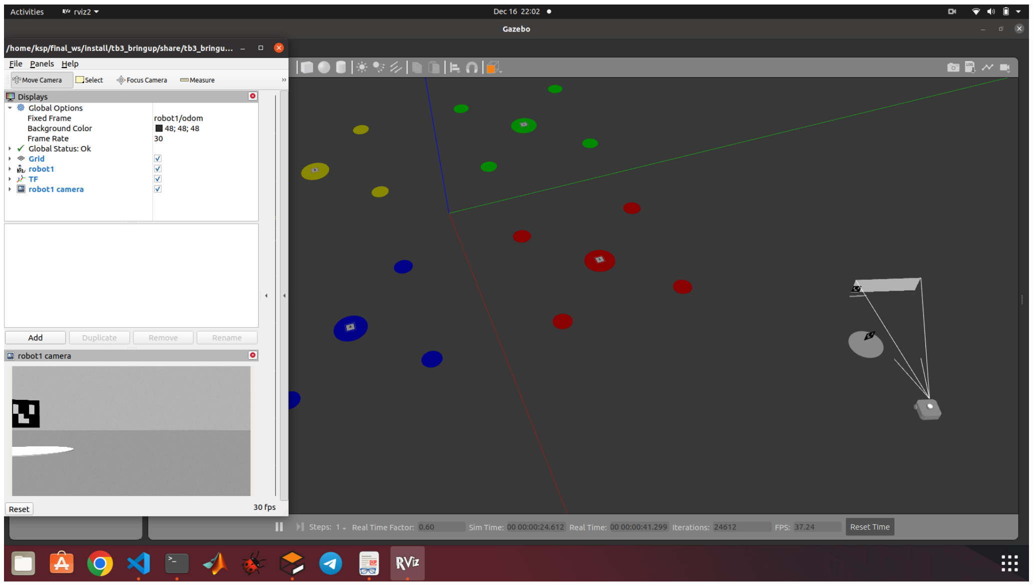
1. Kiran S Patil: Worked on the broadcaster, and code documentation.
2. Nishant Pandey: Worked on the report and target reacher code.
3. Rishikesh Jadhav: Worked on the report and on the code to find the fiducial marker.

5 Resources

- https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Broadcaster-Cpp.html
- https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html
- https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Listener-Cpp.html

6 Results

Figure 6 below shows the output obtained after launching. Click on this to go to the video of the final output



```
Global Options
    Fixed Frame          robot1/odom
    Background Color     48; 48; 48
    Frame Rate           30
Global Status: Ok
Grid                     ✓
robot1                   ✓
TF                       ✓
robot1 camera            ✓
```

```
Steps: 1  Real Time Factor: 0.60  Sim Time: 00 00:00:24.612  Real Time: 00 00:00:41.299  Iterations: 24612  FPS: 37.24
```

Figure 6: Final Results image

7 Course Feedback

- One of the most meticulously designed courses.
- Great job done overall in all aspects.
- Course material provided is very informative, crisp, and simple to understand.
- An assignment or two more for practicing OOPs would have been a better but otherwise good job.