Fall 2022

Final Project

Introductory Robot Programming

December 17, 2022

Student:
Aneesh Chodisetty (117359893),
Sharmitha Ganesan (117518931),
Orlandis Smith (118493077)

Instructors:
Z. Kootbally

Group:
18

Course code:
ENPM809Y

Contents

1 Introduction

This project requires the robot to locate the fiducial marker and go to the goal position with the coordinates provided by the fiducial marker. The robot uses a proportional controller to calculate linear and angular velocity for navigation. Using Gazebo in ROS Galactic, the turtlebot 3 model is spawned in the given environment at the center of four dot markers. The four dot markers are green, yellow, red, and blue in color for better user visualization. Once the turtlebot recognizes the goal location from the fiducial marker, the robot goes to the corresponding location among the four dots. The pipeline is as follows:

1. Go to the goal 1 location to find fiducial marker
2. Publish goal reached message
3. Search for fiducial marker
4. Subscribe to aruco_marker topic to get goal 2 coordinates
5. Go to goal 2 location

2 Approach

The given git repository contained packages like bot_controller, final_plugins, ros2_aruco, target_reacher, tb3_bringup, and tb3_gazebo. A new package odom_updater has been created to connect the disconnected link between frames robot_odom and robot_basefootprint. The given dot markers are static frames, meaning that they will maintain the same pose relative to the world frame whereas the robot pose is dynamic as in Figures 1 and 2.

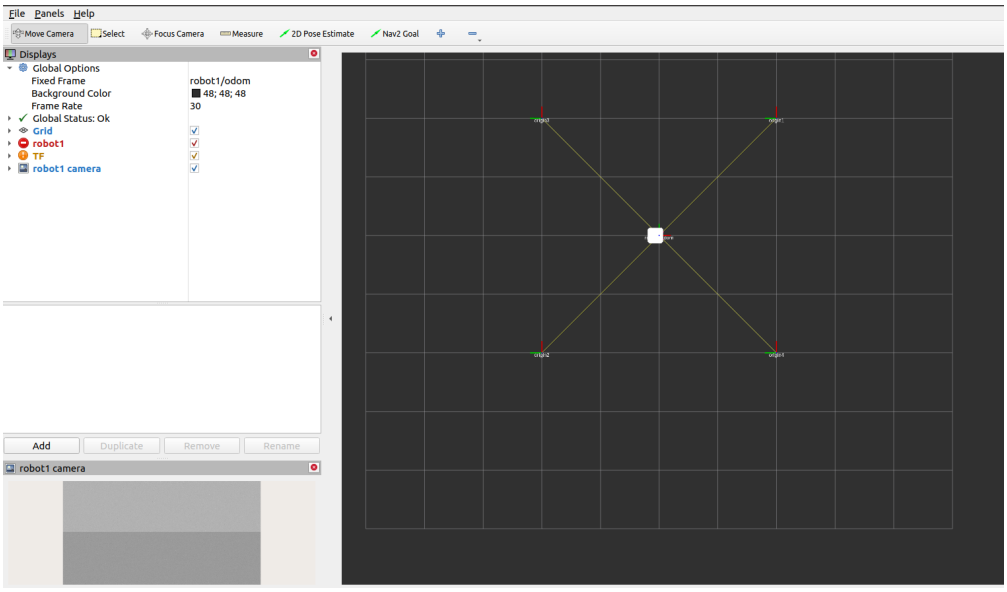

Figure 1: Static Transform Only - Markers

For the success of accurate autonomous navigation, a ROS tf transform from the robot to the world frame must be published and/or generated for the robot to have a true sense of reference (0,0,0) origin and orientation. A connection between the marker's coordinate systems and that of the robot's coordinated system is established using ROS tf2 transformations libraries. The "odom_updater" algorithm is applied for this purpose. The robot has

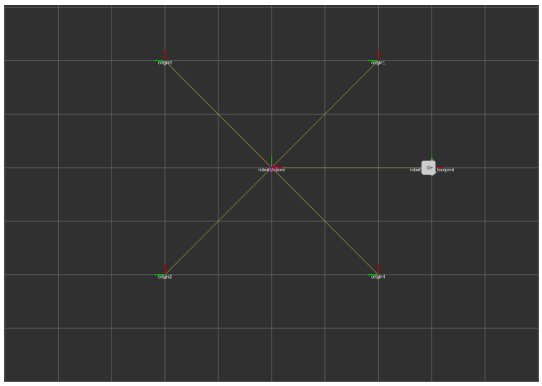

Figure 2: Static w/ Dynamic Transform - Markers and Robot

- world_frame - has its origin at some arbitrarily chosen point, coordinate frame is fixed.
- odom_frame - has its origin at the point where the robot is initialized, and the coordinate is fixed to the world.
- base_footprint - has its origin directly under the center of the robot, and it has the 2D pose of the robot. This coordinate frame moves with the robot.
- base_link - has its origin directly at the pivot point or center of the robot, and its coordinate frame moves as the robot moves.

Algorithm 1 odom_updater

function ROBOT_FOOTPRINT(msg)
    $tf\_broadcaster \leftarrow sendTransfrom(msg)$
end function

function ODOM_CALLBACK(msg)
    $msg \leftarrow clock$
    $msg \leftarrow$ parent_link$\leftarrow$ /robot1/odom
    $msg \leftarrow$ child_link$\leftarrow$ /robot1/base_footprint
    $msg \leftarrow$ position_x
    $msg \leftarrow$ position_y
    $msg \leftarrow$ position_z
    $msg \leftarrow$ orientation_x
    $msg \leftarrow$ orientation_y
    $msg \leftarrow$ orientation_z
    $msg \leftarrow$ orientation_w
    sendTransform $\leftarrow$ ROBOT_FOOTPRINT(msg)
    define t object via geometry_msgs
    $t \leftarrow 0$
    $t \leftarrow 0$
    $t \leftarrow 0$
    $t \leftarrow 1$
    sendTransform $\leftarrow$ ROBOT_FOOTPRINT(t)
end function

The odom to the base_footprint transform is not static because as the robot moves around the world, the footprint frame will constantly change. Algorithm 1 the implemented odom_updater algorithm.

As shown in Figure 3, a connection between the markers and robot frames is established, with the robot having a sense of the world's reference origin (0,0,0).

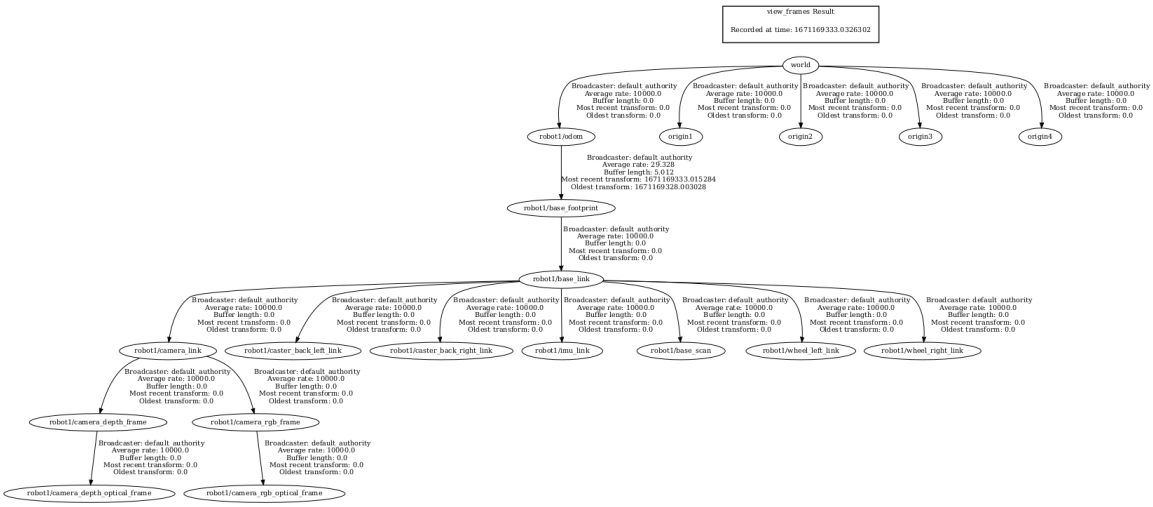

Figure 3: Connecting Frames - Odom -to- Base_Footprint

Further modifications are done in the package target_reacher. Firstly, in the TargetReacher constructor, declaring the parameters such as x-y coordinates of aruco_target, frame_id, and x-y coordinates for four of the aruco markers as final_destination were done. Secondly, in the control_loop method of the class TargetReacher, aruco_target coordinates were given to bot_controller to move the robot to goal 1 as in Figure 4.

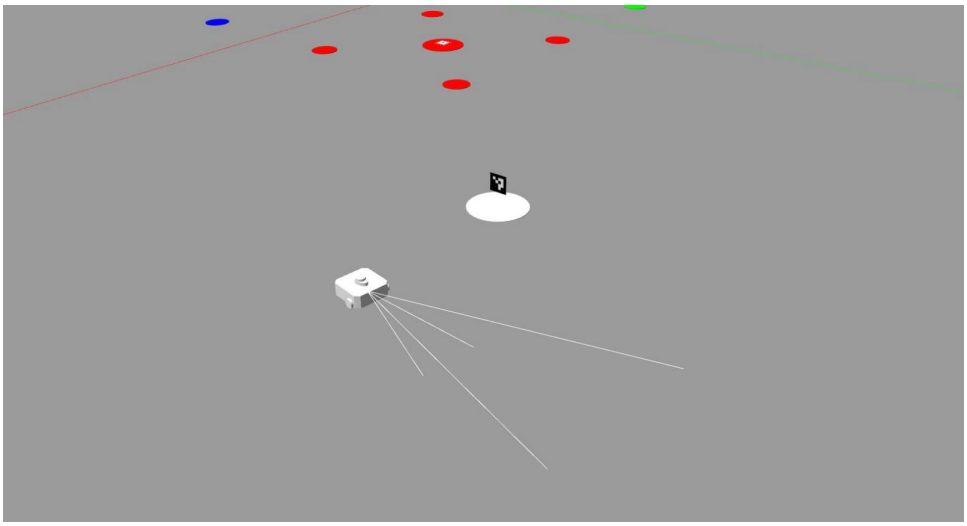

Figure 4: Robot at Goal 1

Thirdly, twist messages are published to the robot until the marker is found. Once the marker is found (Figure 5), the method aruco_callback changes the status of aruco_found to true and subscribes to the message published by the topic aruco_markers to get the final_destination frame id. The method goal_check_callback subscribes to the message published by the topic goal_reached.

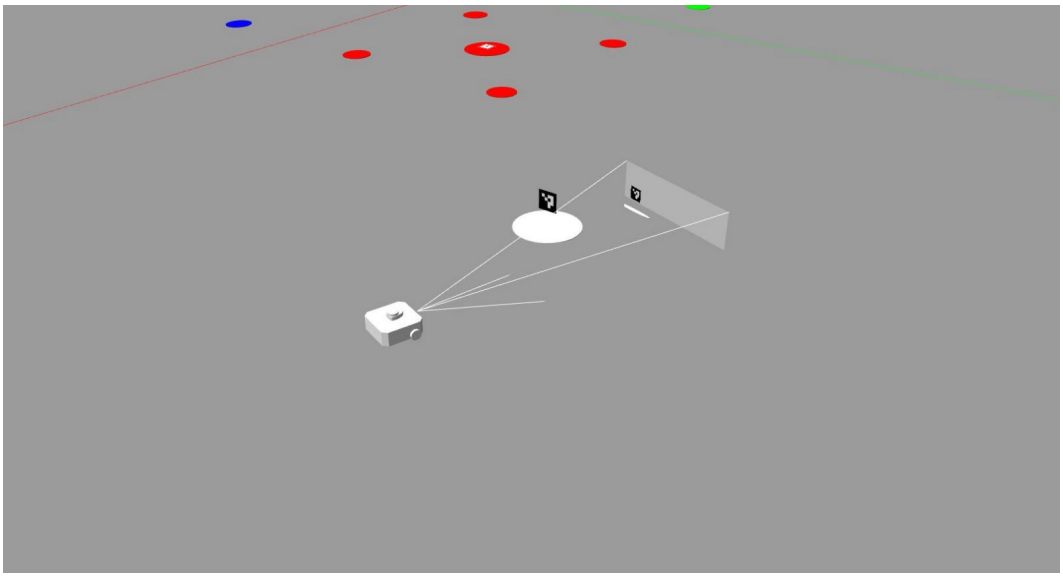

Figure 5: Robot found fiducial marker

Lastly, the a static transform is created between the ($final\_destination$) and $final\_destination.frame\_id$.The method $transform\_and\_send\_goal$ looks up for the final_destination frame, retrieves the goal 2 coordinates and the robot then moves to the transformed goal as shown in Figure 6.

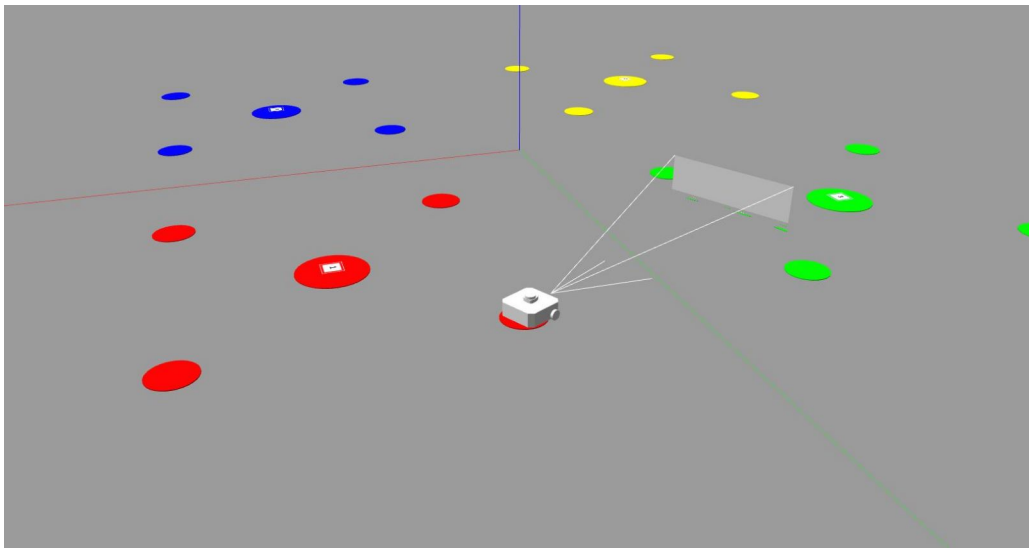

Figure 6: Robot went to final goal

Algorithm 2 is the pseudo-code of target_reacher. The target_reacher class contains the main control loop for this project.

After broadcasting transformation final_destination, the frames graph and the corresponding rviz images are shown in Figures 7 and 8.

The entire pipeline is explained using UML activity and class diagrams. The activity diagram of the approach is given in Figure 9.

The class diagrams are given in Figures 10,11,12.

Algorithm 2 target_reacher

function TargetReacher::TargetReacher(*bot_controller*)
declare parameters ← final_params.yaml**end function**

function TargetReacher::control_loop( )

if $aruco\_goal\_sent is false$ **then**
$m\_bot\_controller \rightarrow set\_goal$(aruco_target.x,aruco_target.y)
SET *aruco_goal_sent* to true

else if $aruco\_found is false$ **then**
publish $twist.angular.z = 0.2$

else if $aruco\_found is true and transform\_created\_is false$ **then**
publish $twist.angular.z = 0$
create static transform with $final\_transform()$
set transform_created$_a$*true*

else
call $transform\_and\_send\_goal()$

end if

end function

function TargetReacher::goal_check_callback(*msg*)
$msg \leftarrow$ data

end function

function TargetReacher::aruco_callback(*msg*)
set $aruco\_found$ to true
$msg \leftarrow$ marked_id

end function

function TargetReacher::final_transform( )
Broadcast ($final\_destination$) transformation as child to $final\_destination.frame\_id$
Store corresponding x-y coordinates

end function

function TargetReacher::transform_and_send_goal( )
transform $/final\_destination$ to $/robot1/odom$
$m\_bot\_controller \rightarrow set\_goal$(odom_transform.transform.translation.x,odom_transform.transform.translation.y)

end function

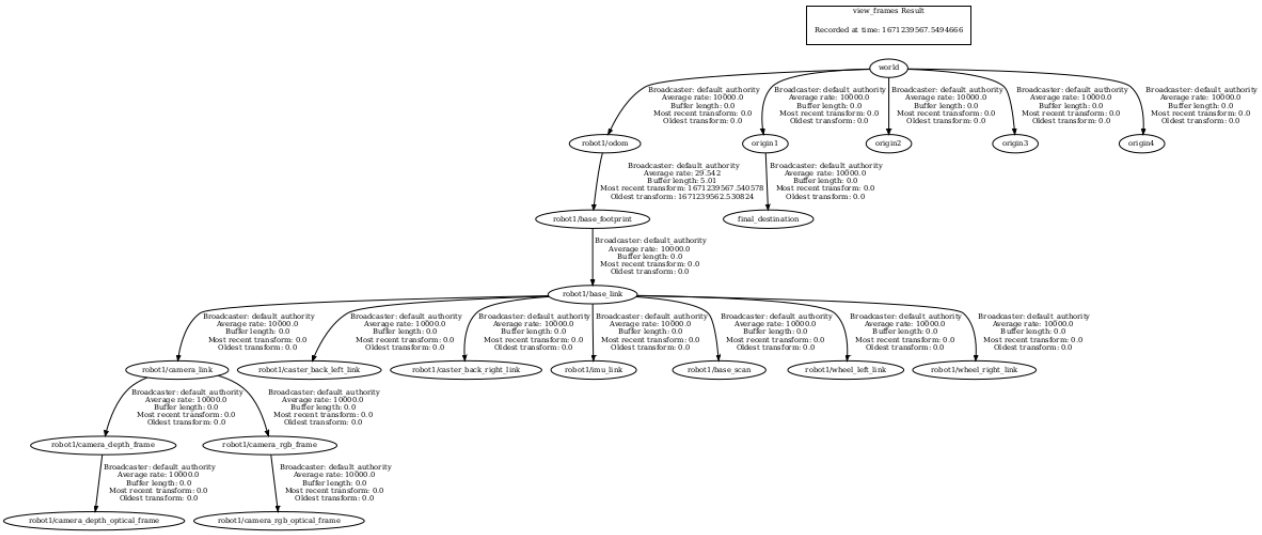

Figure 7: Final Frames

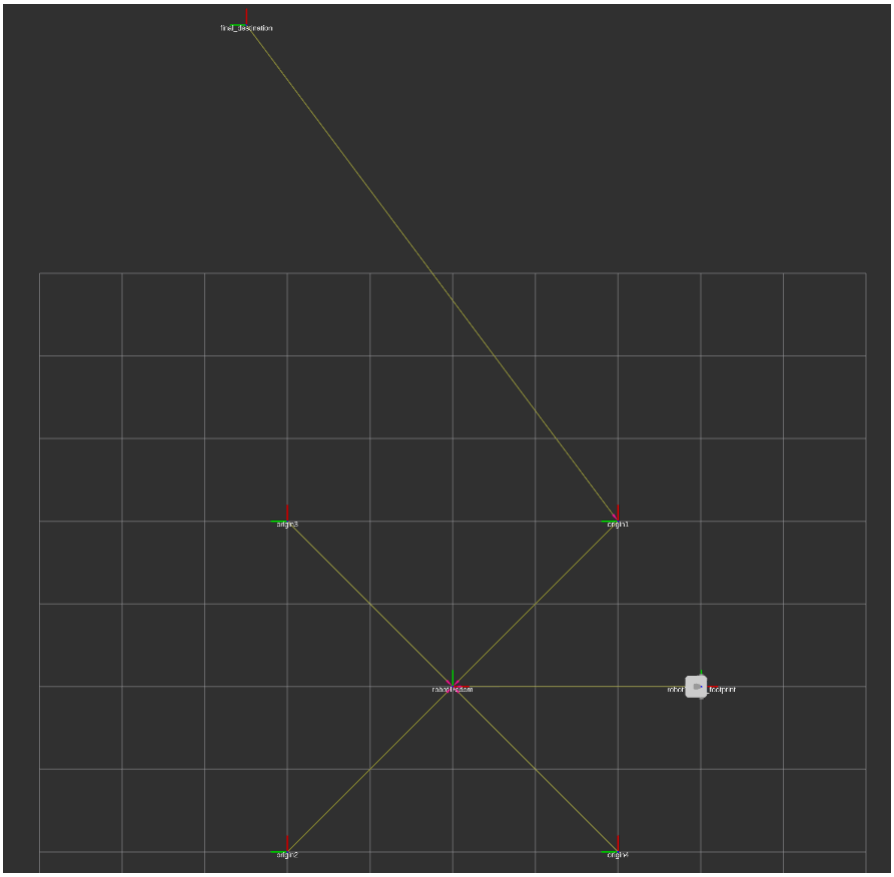

Figure 8: Final Frames - Rviz

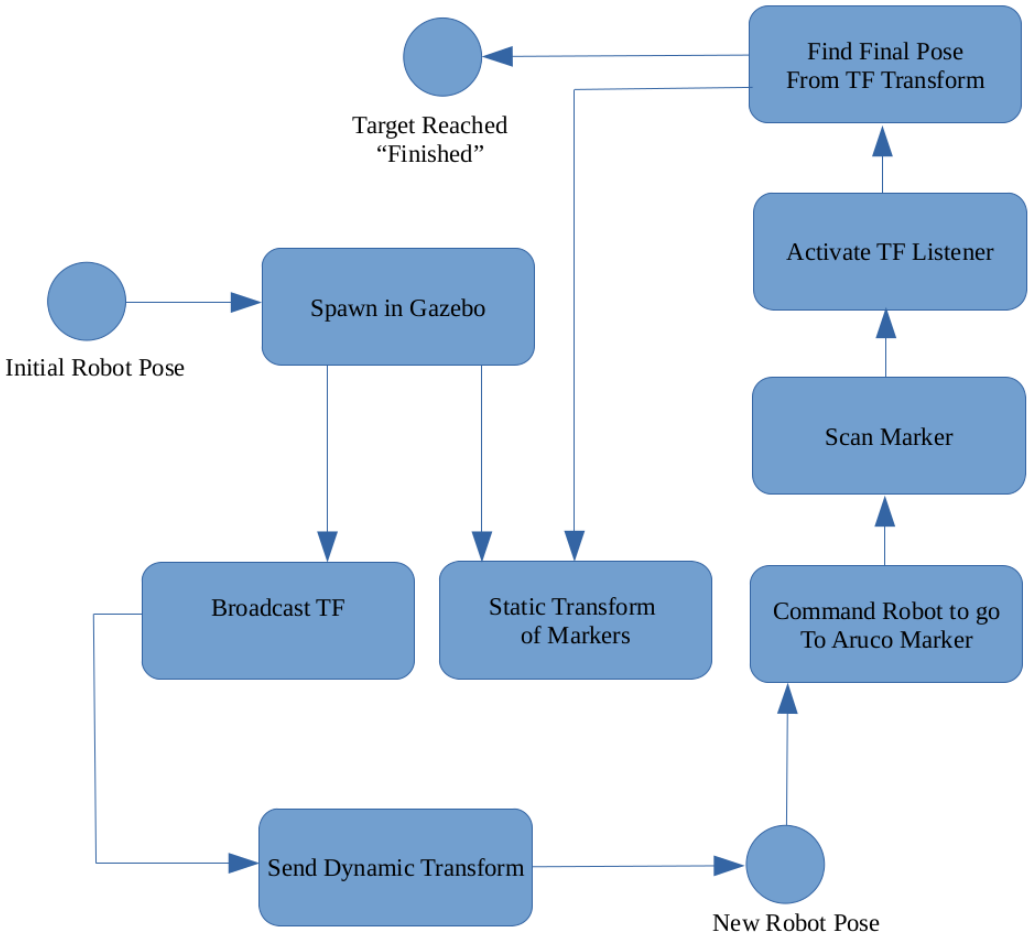

Figure 9: Activity Diagram

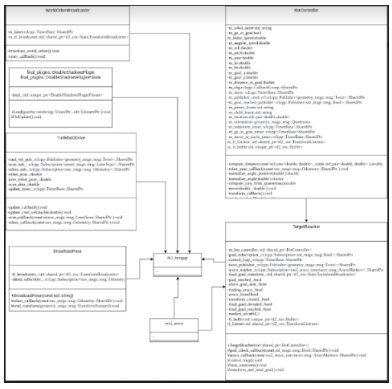

Figure 10: UML class Diagram

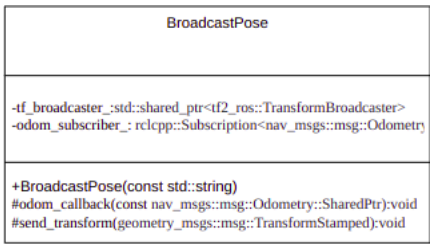

Figure 11: Class Diagram for odom updater

Figure 12: Class Diagram for target reacher

3 Challenges

- Initially, it took time to understand the purpose of all the provided packages and to link all of them as one entity.
- In developing the odom_updater algorithm, it was initially challenging to understand whether or not the entire tree of connections from the world to base_link should be created. It was then realized from searching ROS reference documentation that the odom frame is already connected to the world, so in that case, the algorithm only focused on connecting the base_footprint to the odom link.

4 Contribution in the project

- Aneesh Chodishetty - Worked on the target_reacher algorithm
- Orlandis Smith - Wrote the odom_updater algorithm and assisted with the report.
- Sharmitha Ganesan - Worked on the report, pseudocode and UML diagrams.

5 Resources

- `https://github.com/zeidk/enpm809y_FinalFall2022`.

6 Course Feedback

- The quizzes helped do get a real grasp on the basic C++ concepts.
- The coursework covered essential parts of robot programming providing a good overview.
- The course presentations and lectures were very helpful, for this course to cover a lot of material it was well organized. A lot of advanced and complex concepts were learned which will be very useful in industry.
- A future improvement can be done by enabling students to write project packages from scratch instead of starting off with pre-written packages.