Fall 2022
Introduction to Robot Programing

Final Project

Introduction to Robot Programming

December 17, 2022

Students:
Kshitij Karnawat
119188651 kshitij@umd.edu

SaiTeja Gilukara
119369623 saitejag@umd.edu

Akashkumar Parmar
118737430 akasparm@umd.edu

Aashrita Chemakura
119398135 achemaku@umd.edu

Instructors:
Z. Kootbally

Group:
11

Course code:
ENPM809Y

Contents

1 Introduction **4**
1.1 Problem statement 4
1.2 Design Overview 4

2 Approach **5**
2.1 Algorithm Phase 5
2.2 Implementation Phase 8
2.3 Testing Phase 11

3 Challenges **14**

4 Contributions **15**

5 Resources **16**

6 Course Feedback **17**

List of Figures

1 Gazebo Environment 4
2 Flowchart for moving the robot to the pre-defined goal 5
3 Marker detection flowchart 6
4 Flowchart for detection of final destination 6
5 Flowchart for moving the robot to the final destination 7
6 RQT Graph 7
7 tf tree frames 8
8 odom updater class diagram 9
9 Target reacher class diagram 11
10 Detection of aruco marker 11
11 Destination at Origin 1 12
12 Destination at Origin 2 12
13 Destination at Origin 3 13
14 Destination at Origin 4 13

1 Introduction

A Gazebo environment is given with a aruco marker, five large dots and 16 small dots of red, blue, green and yellow colours on the floor where the robot is moving. The robot is mounted with a camera. Larger dots of red, yellow, green and blue colours represent the coordinates of static frames namely origin 1, origin 2, origin 3 and origin 4 respectively. The large white dot helps the user visualize where the marker is located in the environment. Smaller dots represent all the possible destinations that the robot can reach.

1.1 Problem statement

The aim of the project is to make the robot reach a certain pre-defined position and rotate at that position; where it detects the aruco marker which guides the robot taking the reference of the certain origins (larger dots), 1,2,3 or 4 and commands the robot to reach a destination (smaller dots) with reference to the origin generated.

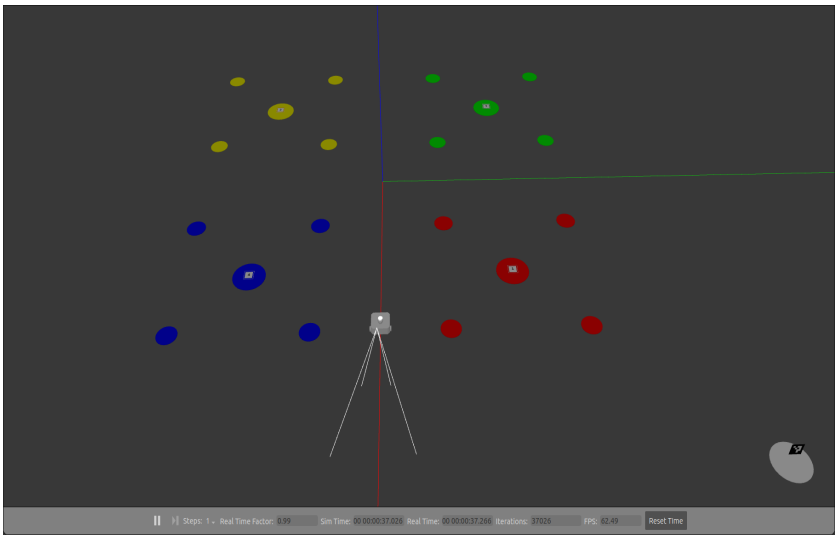

Figure 1: Gazebo Environment

1.2 Design Overview

The Linux OS platform will be used to build this project, and C++ will be used to implement it. Robot Operating System 2 (ROS) and OpenCV with corresponding packages for movement, vision, transformation, and detection are used. ROS2 galactic distribution-specific packages will make it possible to access the necessary files and functions for making calls to the bots' interfaces.

Additionally, the packages will enable communication between the created algorithms and the ROS2 server, which already has all the necessary pre-configured sub-components.

The produced code can communicate with the ROS2 server through calls in developed launch and header files by exploiting these previous codebases. These can be used to control how the robot moves about its surroundings.

By utilizing these preexisting codebases, the developed code can interact with the ROS server via calls in developed launch and header files. These can be used to influence the movement of the robot in the environment.

2 Approach

In order to complete the task at hand, coding practices have been adapted. Simultaneously, all while replicating a realistic office workflow—the following three-phase structure has been used. The three phases are algorithm, implementation, and testing, listed in chronological sequence. Code optimization and error debugging, which are not regarded as independent stages but rather combined into the aforementioned phases, were carried out toward the end of the Implementation phase and then concurrently with the Testing phase.

2.1 Algorithm Phase

A significant amount of effort was first invested in the algorithm phase to thoroughly comprehend the challenge and its goals. To grasp each particular sub-task inside the project, a comprehensive reading of the project instructions document that was provided was done. The provided final project ROS2 package and the already developed Bot Controller codes were executed and understood at the same time starting with the second read through of the paper.

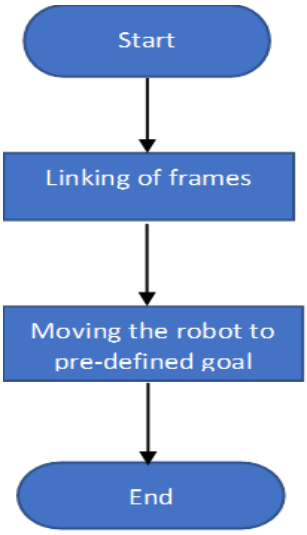

Figure 2: Flowchart for moving the robot to the pre-defined goal

The bove flowchart depicts the process of moving the robot to the first destination. This task consists of linking the /odom frame and the /base_footprint frame and moving the robot to the pre-defined destination.

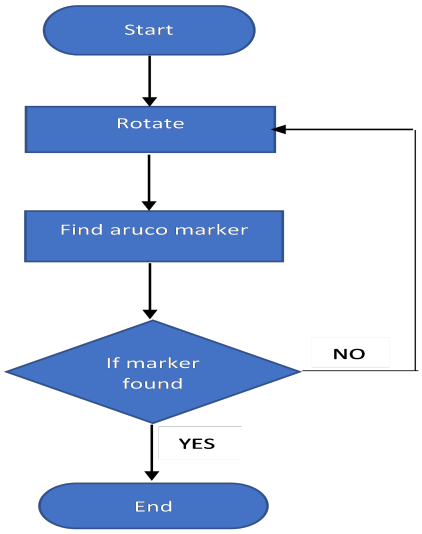

Figure 3: Marker detection flowchart

The above flowchart illustrates the process of aruco marker detection. As the robot reaches to the first destination, it starts rotating at that position until it finds an aruco marker.

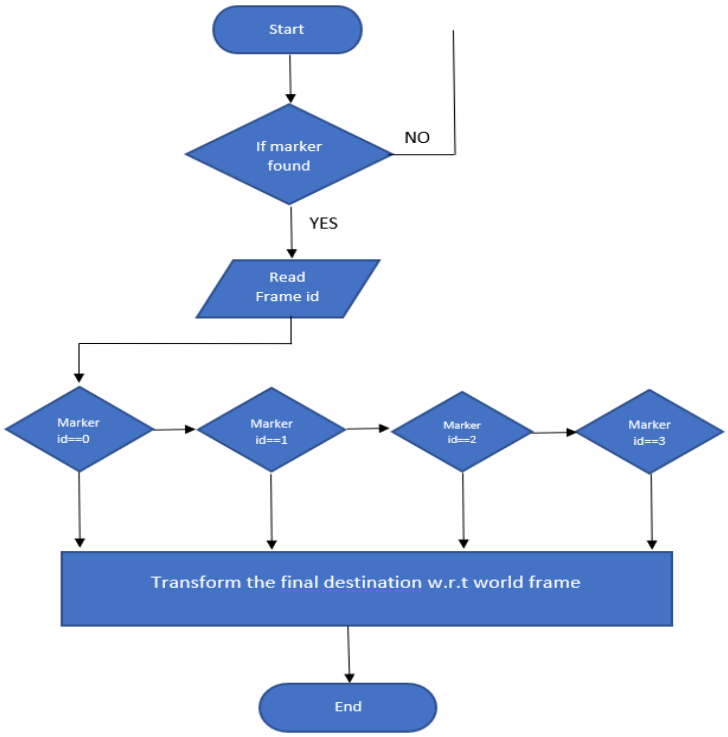

Figure 4: Flowchart for detection of final destination

The above flowchart depicts the process of transforming the relative frames to the world frame. The aruco marker gives the data of the final destination with respect to the particular origin in a

relative manner. As soon as the data is read from the aruco marker, the program computes the coordinates of the final destination with respect to the world frame.

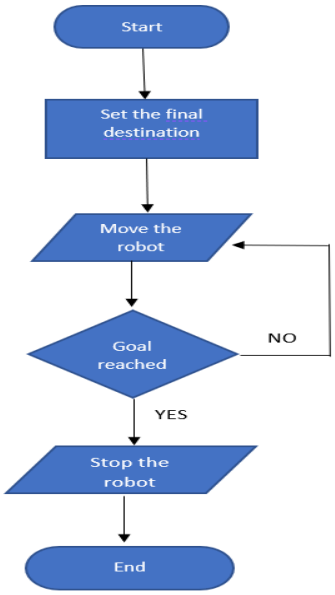

Figure 5: Flowchart for moving the robot to the final destination

The above flowchart demonstrates the process of moving the robot to its final destination. Post computation of the final goal coordinates with respect to the world frame, the goal is set using the pre-defined function and the robot moves to the desired coordinates.

The node connection of the whole project is shown in figure 6.

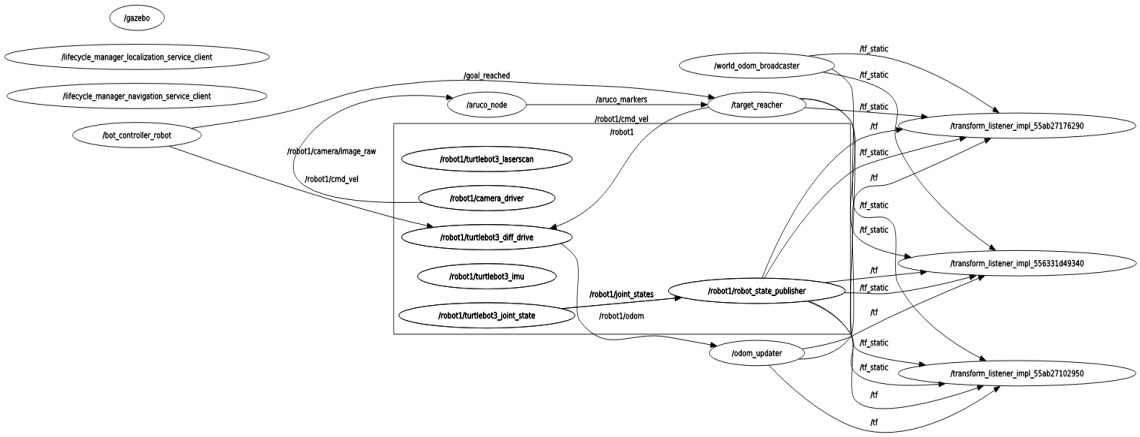

Figure 6: RQT Graph

2.2 Implementation Phase

The implementation step followed after obtaining a general understanding of the method and how the overall program operated. During this phase, each block in the flowchart shown in [Fig.] was implemented using standard procedural programming. After achieving good results, the code was changed to incorporate object-oriented programming to take use of its many benefits.

- Prerequisites for Implementation of the project:
  -Creation of a new workspace "final_ws".
  -Installation of OpenCV.
- Creation of package "odom_updater":
  -The package "odom_updater" has been created with the following dependencies: rclcpp, geometry_msgs, nav_msgs and tf2_ros.

  -This package accounts for the connection of "/robot1/base_footprint" topic as a child frame of "/robot1/odom" topic to communicate in the world frame.

The figure below shows the updated tf frames.

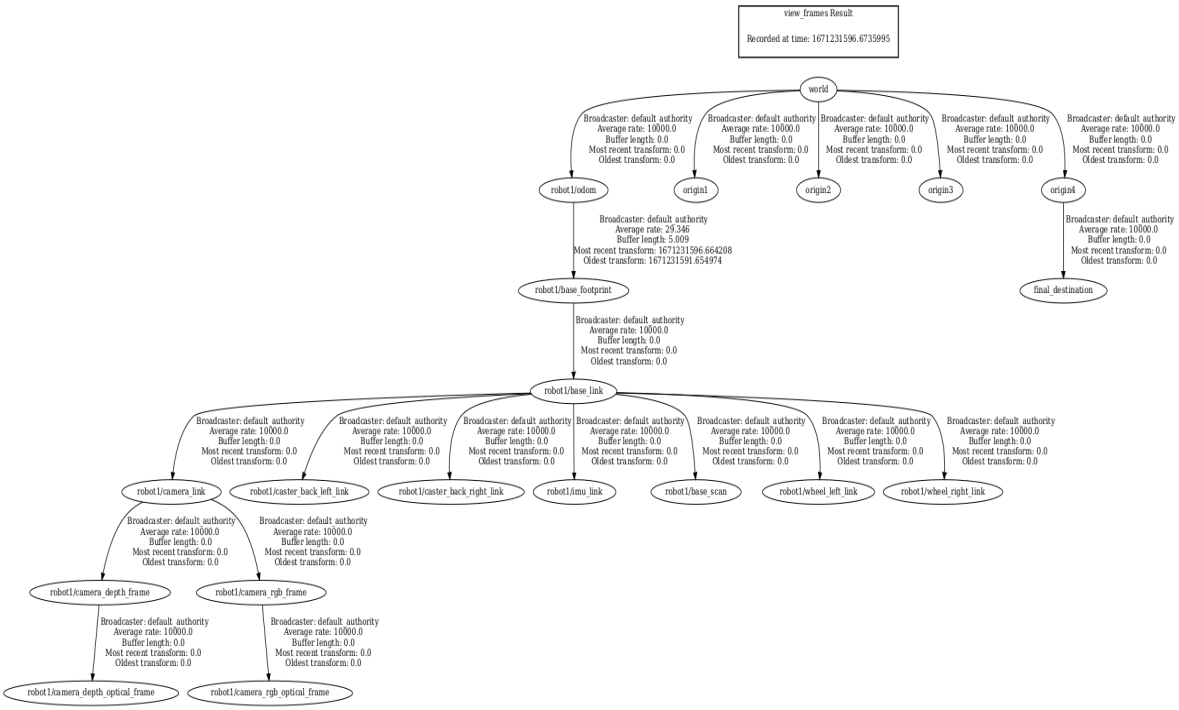

Figure 7: tf tree frames

To link the transformations, the odom_updater package is created with following dependencies: rclcpp, geometry_msgs, nav_msgs, tf2_ros

The pseudocode for the same is shown below:

Algorithm 1 odom_updater

get frame robot1/odom
parent frame ← $/robot1/odom$
child frame ← $/robot1/base_footprint$
translation.x ← $pose.position.x$
translation.y ← $pose.position.y$
translation.z ← $pose.position.z$
rotation.x ← $pose.orientation.x$
rotation.y ← $pose.orientation.y$
rotation.z ← $pose.orientation.z$
rotation.w ← $pose.orientation.w$
send transform()

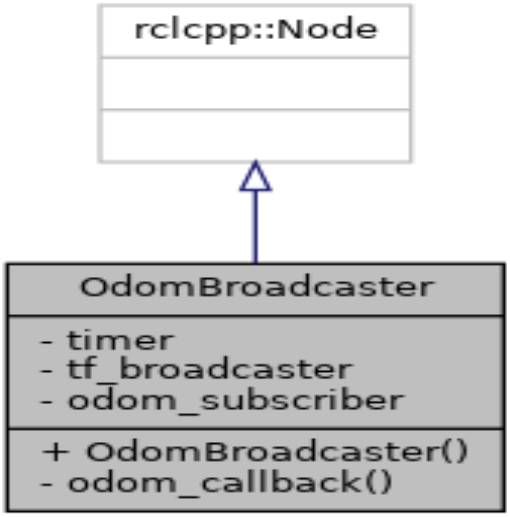

Figure 8: odom updater class diagram

After the successful linking between tf frames, the target_reacher package is edited for the completing the task.

The pseudocode for the same is shown below:

Algorithm 2 target_reacher

declare parameters
set_goal(x,y); (for the first goal)
$i \leftarrow 10$
if (goal_reached == true)
    spin robot
    set linear velocity = 0
    set angular velocity = 0
    publish (msg)
if (aruco_marker detected == true)
    grab marker id
if (id == 0)
    get parameter aruco_0_x
    get parameter aruco_0_y
    set$_goal(aruco_0_x, aruco_0_y)$
if (id == 1)
    get parameter aruco_1_x
    get parameter aruco_1_y
    set_goal(aruco_1_x, aruco_1_y)
if (id == 2)
    get parameter aruco_2_x
    get parameter aruco_2_y
    set$_goal(aruco_2_x, aruco_2_y)$
if (id == 3)
    get parameter aruco_3_x
    get parameter aruco_3_y
    set_goal(aruco_3_x, aruco_3_y)
transform final destination coordinates (relative frame to world frame)
call buffer to have frame published continuously

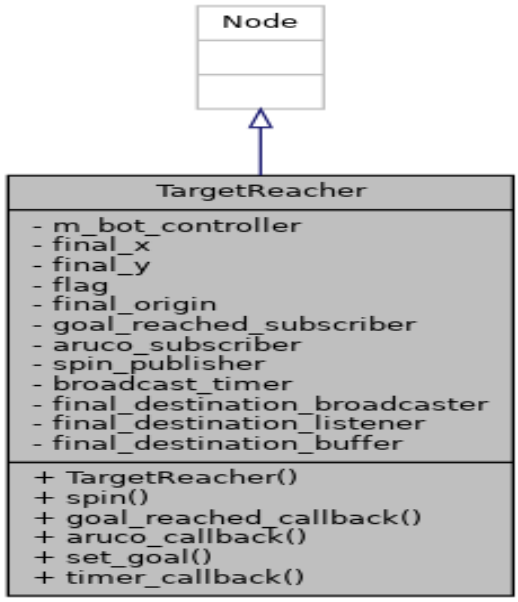

Figure 9: Target reacher class diagram

2.3 Testing Phase

The first task was to make robot reach the pre-defined destination and rotate till it detects the aruco marker. In [fig. 10], the robot is shown in Gazebo as well as in Rviz, where an aruco marker is getting detected in its camera frame.

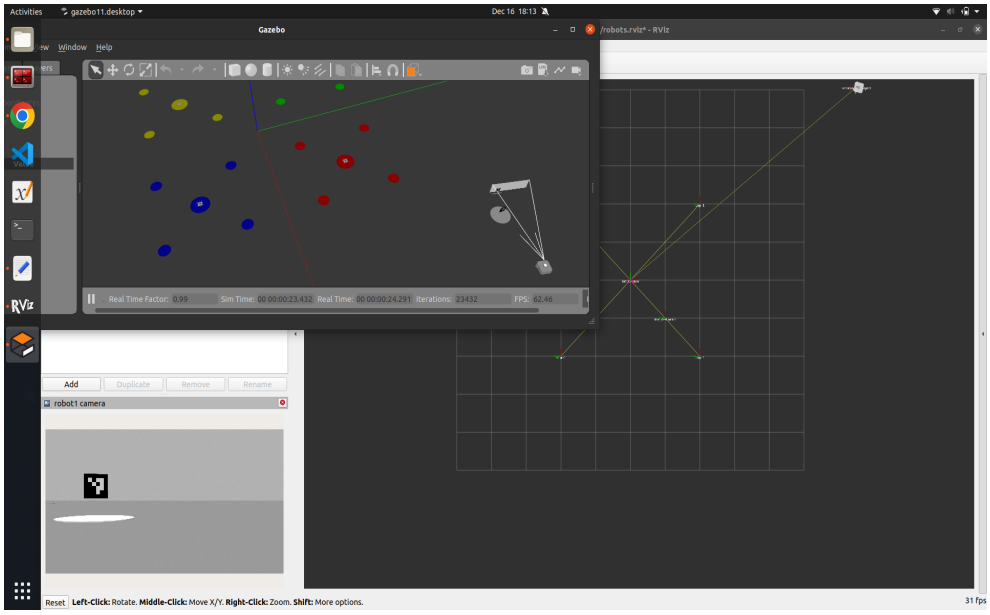

Figure 10: Detection of aruco marker

After successful implementation of the code, it is evaluated on 4 different origins. The robot should reach to the particular final destination based on the origin set in 'final_params.yaml' file. The robot is able to reach to all the 4 origin conditions. The robot's position at its final destination corresponding to appropriate origin are shown in figures 11-14.

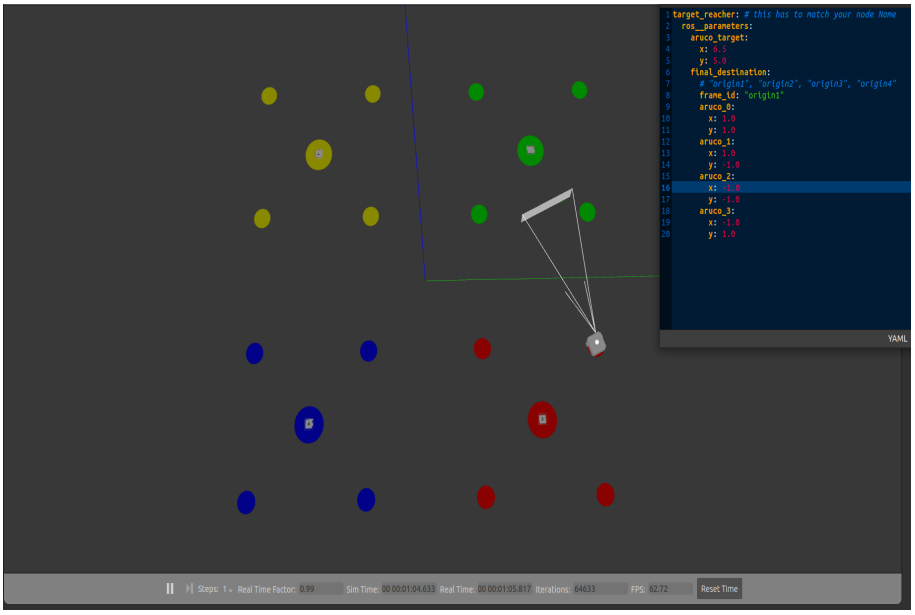

Figure 11: Destination at Origin 1

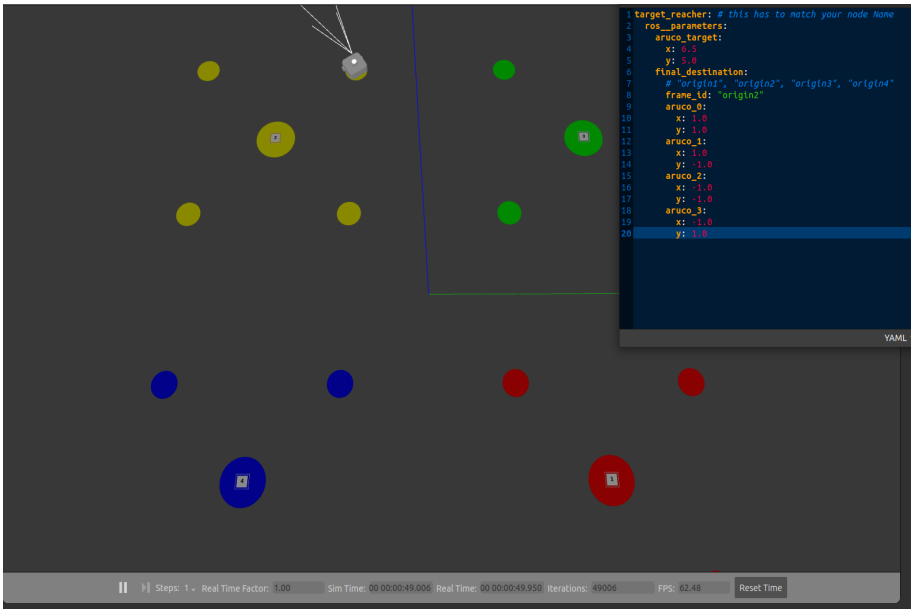

Figure 12: Destination at Origin 2

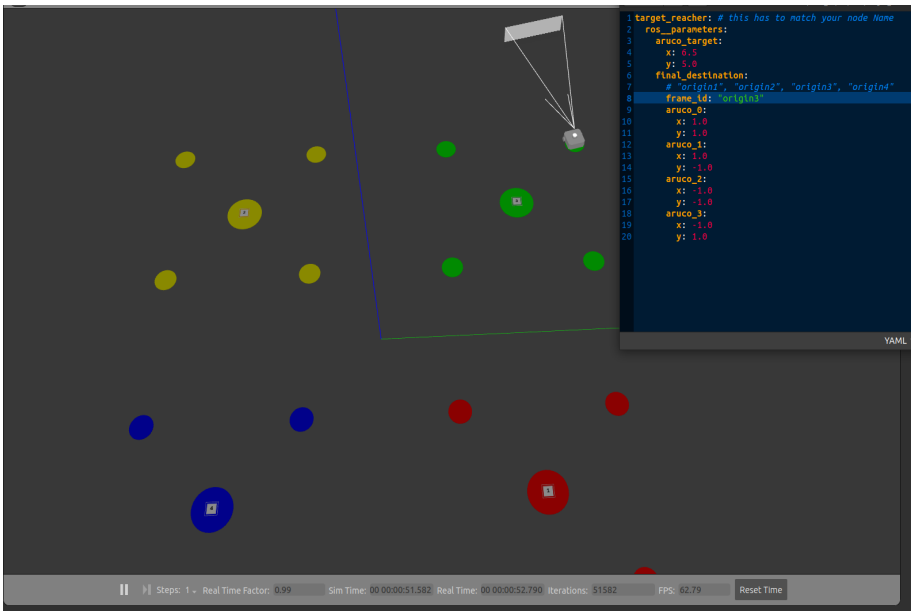

Figure 13: Destination at Origin 3

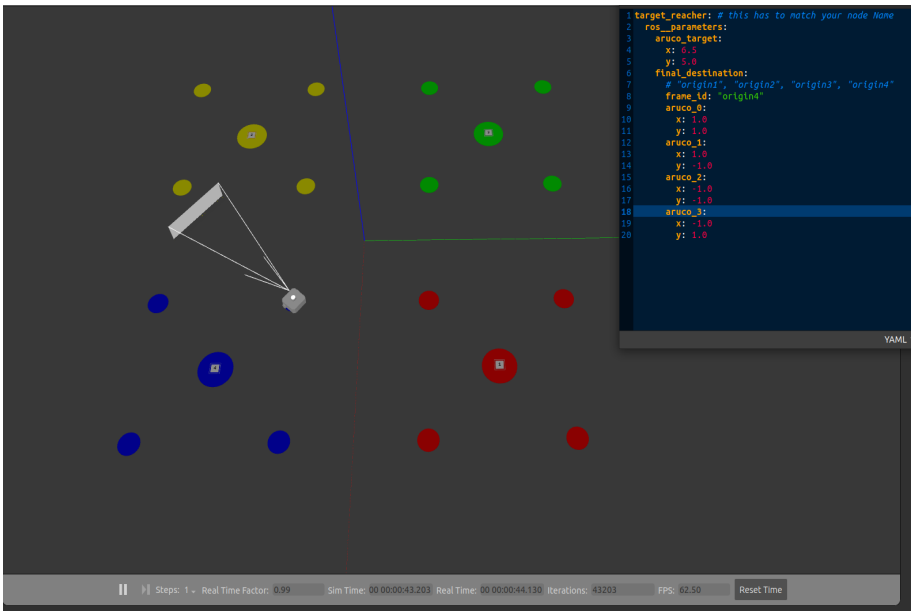

Figure 14: Destination at Origin 4

3 Challenges

Problems:

1. No display of robot model and marker in Gazebo world:
   Difficulty in detection of mesh miles during simulation

Solution:

Provision of the updated path in /.bashrc file everytime a change was made in the location of mesh files.
Restarting the terminal after the correction of the path file in /.bashrc file in order for the changes to be applied.

2. Connecting tf trees:
   Difficulty in connecting the /odom and /base_footprint.

Solution:

Understanding of the proper syntax to solve the problem.

3. Synchronization between Gazebo and Rviz:
   Difficulty in the detection of the movement of the robot in Rviz while it persisted in Gazebo.

Solution:

Obtaining the current pose of the robot.
Subscribing the current pose of the robot and updating it in Rviz.

4. Final frame overlapping:
   The final frame was getting spawned at every iteration.

Solution:

Creation of buffer and timer resolved the issue.

5. Continuous Robot spinning:
   The robot was spinning continuously by itself even after reaching at the final destination.

Solution:

By the use of flag, this error has been solved.

4 Contributions

1. Kshitij Karnawat-
   Worked on linking of tf trees, documentation of the code and transformation of the frames after the detection of aruco markers.
2. Sai Teja Gilukara-
   Worked on linking of tf trees, making the robot reach the aruco marker and rotating at that point and making the robot reach the final destination.
3. Akashkumar Parmar-
   Worked on connecting tf trees in odom updater, report and setting up the environment.
4. Aashrita Chemakura-
   Worked on the report, testing the test cases and writing the pseudo code.

Simultaneously, everyone worked on debbuging the errors and overcoming the challenges faced in the code.

5 Resources

[1] https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Broadcaster-Cpp.html
[2] https://github.com/zeidk/enpm809y_FinalFall2022/blob/main/final/world_odom_broadcaster/src/world_odom_
[3] https://cplusplus.com/
[4] https://docs.ros2.org/foxy/api/tf2_ros/index.html
[5] https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Static-Broadcaster-Cpp.html
[6] https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Listener-Cpp.html
[7] https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Adding-A-Frame-Cpp.html

6 Course Feedback

Everyone in the groupd found the course very informative. The course not only focused on the theoretical part, but it also has given the insights of practical tools with challenging assignments. The course emphasized on the use of good practices. Specially, we enjoyed solving the extra credit in RWA2 and saw ourselves growing trying to formulate efficient algorithms.