Fall 2022

Final Project

Introductory Robot Programming

December 16,2022

Students:
Chris Neil D Souza

Shivam Sehgal

Patrik Dominik Pördi

Shameek Chandra Vangapalli

Instructors:
Z. Kootbally

Course code:
ENPM809Y

Contents

List of Figures

1 Introduction

The goal of this project is to move the turtlebot to a desired destination with the help of fiducial markers which contain coordinates to the final goal. The turtlebot moves towards the goal 1 to locate the fiducial marker, subscribes to get the info from the respective topics and moves towards the destination. To achieve this ROS 2 has been used to visualize the robot and the markers in a Gazebo environment. Rviz is used to view the frames of each point and the current robot frame. Figure 1: Shows the process flow diagram followed.

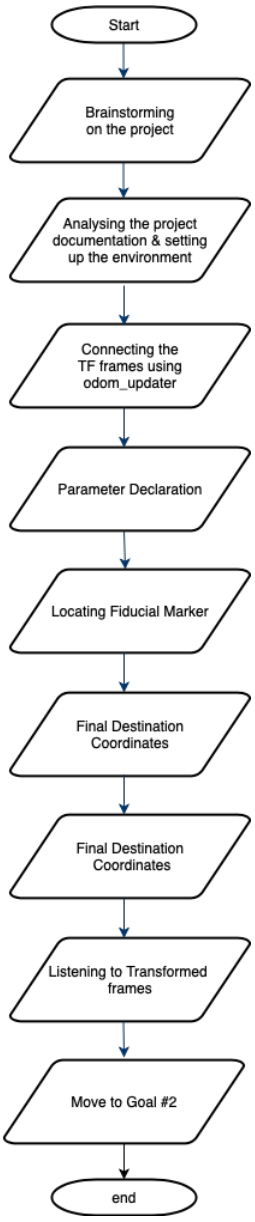

Figure 1: Flowchart of our project

2 Approach

The approach followed while working on this project can be divided into the following sections:

2.1 Brainstorming:

This part basically involves getting the overview about the project and its application. This was achieved by watching the video, coming up with a flowchart to solve this problem (figure 1) and finally, followed by several discussions on the presented approach.

2.2 Setting up the environment:

First, the document provided by the professor had to be read multiple times to understand how the given application works. After that, the required packages were downloaded and their working principles were investigated. This helped to give a brief idea of what is to be achieved and helped in solving a lot of errors which were found on run time. The work was then divided and allotted to each member of the group. Figure 2:Class Diagram shows the classes which were used for the project. All the methods and members needed in this part were defined.

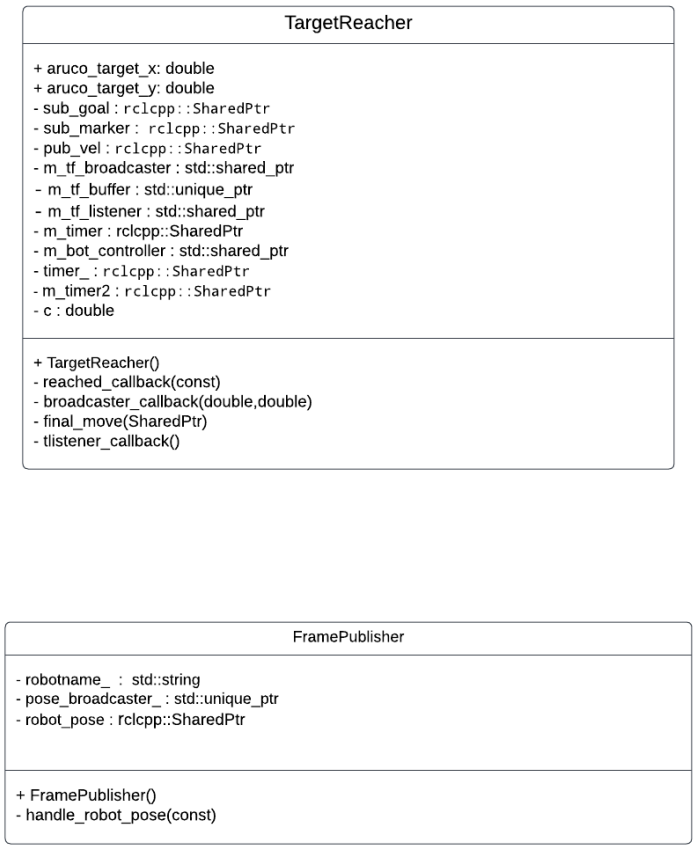

Figure 2: Class Diagram

2.3 Connecting TF Frames using odom_updater:

2.3.1 What is tf?

A ROS frame typically has many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. tf keeps track of all these frames over time. The relationship between objects of the frame works like a chain, their connection can be described by the transformation between the corresponding frames.

2.3.2 Methodology:

After running the simulation and printing out the frames, it was observed that there were two disconnected transformation trees : "/world" and "/robot1/base_footprint". The world tree has a branch named "/robot1/odom" and the first task was to connect this branch to the "/robot1/base_footprint" branch. Here, the parent is the "/robot1/odom" tree and the child tree is "/robot1/base_footprint". As per instructions provided in the document, a package named odom_updater was created, along with a node named odom_updater. The next step was to make a broadcaster to broadcast "/robot1/base_footprint" as a child of "/robot1/odom". The frame "/robot1/base_footprint" was a mobile frame and therefore, the broadcaster had to be non-static. A subscriber to the topic "/robot1/odom" was needed to retrieve the position of robot in odom. In the callback method of the subscriber the function that broadcasts the postion of robot in odom was called.Figure 3: Shows the pseudo code to connect the tf frames.

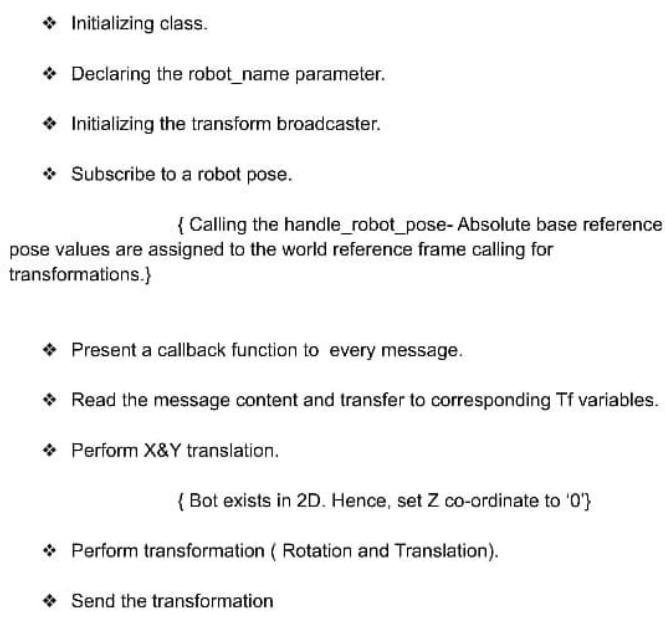

Figure 3: Pseudo code to connect tf frames

2.3.3 Results:

On running the simulation and printing out frames, it was observed that the frames had been connected with "/robot1/odom" as parent frame and "/robot1/base_footprint" as the child frame. Figure 4: Frames obtained by RQT This was the final frame tree obtained with connected branches.

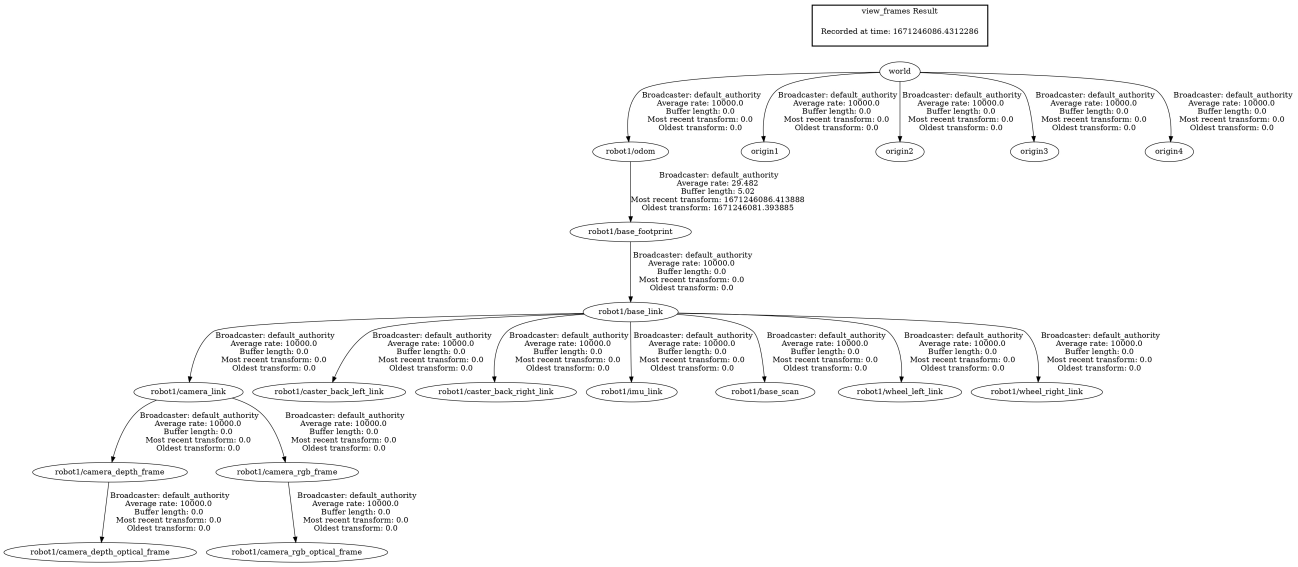

Figure 4: Frames obtained by RQT

2.4 Parameter declaration:

2.4.1 final_parameter.yaml

The required parameters are located in "final_params.yaml" file which is in the YAML format. This file contains the node name for which the following parameters will be loaded: the first goal, the robot will find the fiducial markers, the final_destination where robot needs to reach, the frame ID and fiducial marker id.

In "final_launch.py", the node target_reacher was run with the parameters in "fina_params.yaml".These parameters were declared in the class TargetReacher using: this$\langle declare_parameterType(\langle parametername\rangle);TheFigure5$
$Flowchartforlocatingfiducialmarkersshowingthedecisionstodoso.Figure6: FiducialMarkersshowthehowthefiducialm$

2.5 Locate Fiducial Markers:

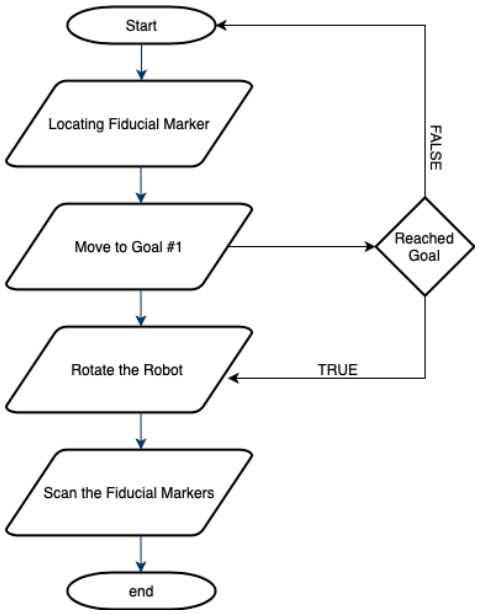

Figure 5: Flowchart for locating fiducial markers

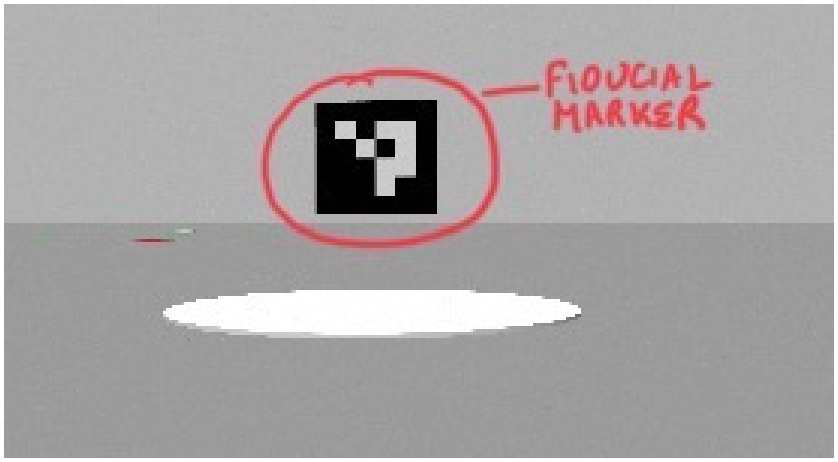

Figure 6: Fiducial Markers

2.5.1 Move the robot to Goal 1:

The fiducial markers are in a particular location in the gazebo world, to scan them on the camera the robot needs to reach goal 1. The positon is obtained through node target_reacher using final_params.yaml as x=aruco_target.x and y=aruco_target.y, using m_bot_controller-(set_goal(x, y)); the robot can move to x,y. Once robot reaches the goal a message is published on the topic /goal_reacher, subscribing to this topic an output is obtained on the terminal using reached_callback method as 'I heard "1"'.

2.5.2 Rotating the Robot:

On reaching goal 1, the robot needs to rotate to find the fiducial markers. To rotate the robot, Twist messages are published on the topic /robot1/cmd_vel keeping linear velocity to 0 and angular velocity to 0.2.

2.5.3 Scanning Fiducial Markers:

When the fiducial marker is found by camera it publishes a message on topic aruco_markers by node aruco_node. To get the message published on this topic a subscriber was made to get the marker id from this message. The marker_id is of type integer vector to get this value, marker_id.at(0) was used.

2.6 Final Destination coordinates:

Using the marker id from the last step goal parameters were retirevd using final_destination.aruco_i.x and final_destination.aruco_i.y, where "i" is the marker id. String concatanetion was used to make this name dyanmic on marker id. The values of final destination are in frame assigned to final_destination.frame_id which is defined in the .yaml file. The final destination contains positions needed to be transformed to the frame /robot1/odom . Figure 7: Pseudo code to obtain final destination shows pesudo code to obtain the final destination positions. Figure 8: World shows how the world looks in gazebo and with defined spots in green annotations.

Target Reacher (To move to the end target)

- Setting If, Elif, else statements.

{Calling broadcaster functions}

- Implementing the final goal output.
- Create broadcaster functions.
- Reading the messages and assigning to tf variables.

{Knowing or calling for position and orientation from turtlebot}

- Calling for transformations.
- Transformation between odom and object frame.
- Position in ODOM.

Figure 7: Pseudo code to obtain final destination

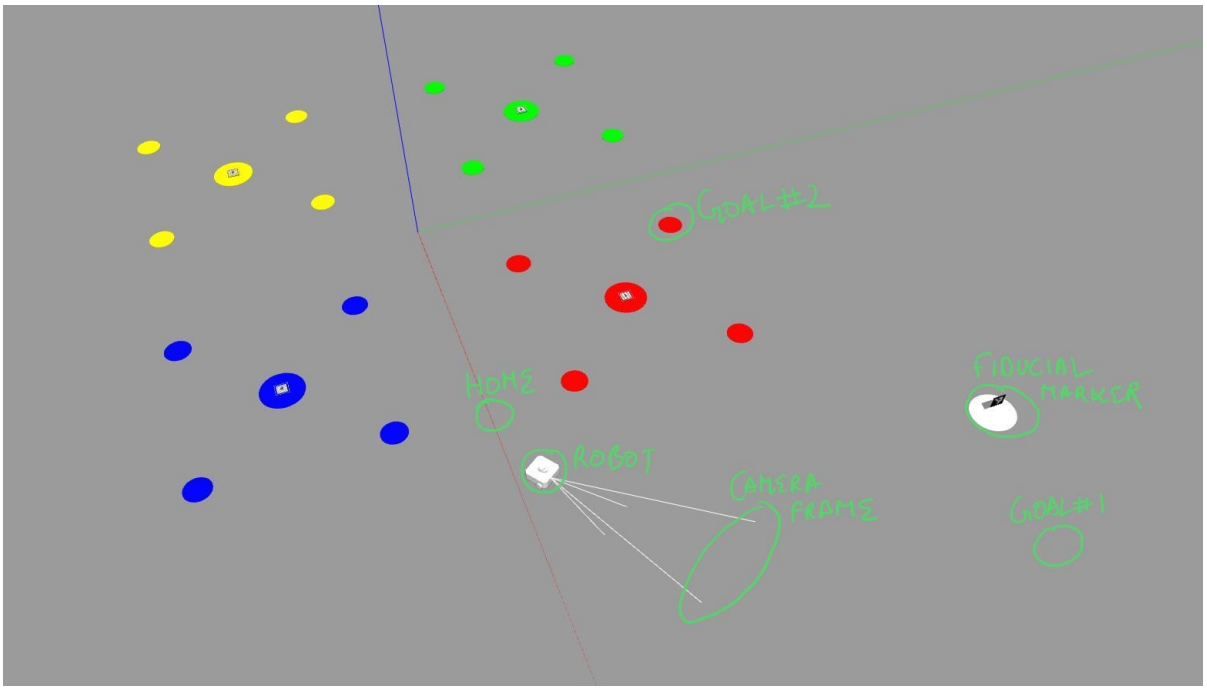

Figure 8: World

2.7 Transforming Frames:

Accomplished by broadcasting a frame final_destination in the frame retrieved from final_destination.frame_id. The position of this frame in origin is given by: x = final_destination.aruco_i.x, y = final_destination.aruco_i.y, z = 0 and orientation as (0,0,0,1). m_tf_broadcaster2 is used to send a transformation of the given position above in /robot1/odom frame.

2.8 Listener to broadcaster:

To get the transformation by the broadcaster a listener is made to listen to the broadcasted transformations. This gives the final destination positions in /robot1/odom frame.

2.9 Move to Goal 2:

After getting x,y,z from this listener. The method m_bot_controller-(set_goal(x, y)) is used to move the robot to the goal 2. At this point, the robot is at the final destination and there is nothing else to do. Figure 9: GOAL 2 is the final goal for the robot .

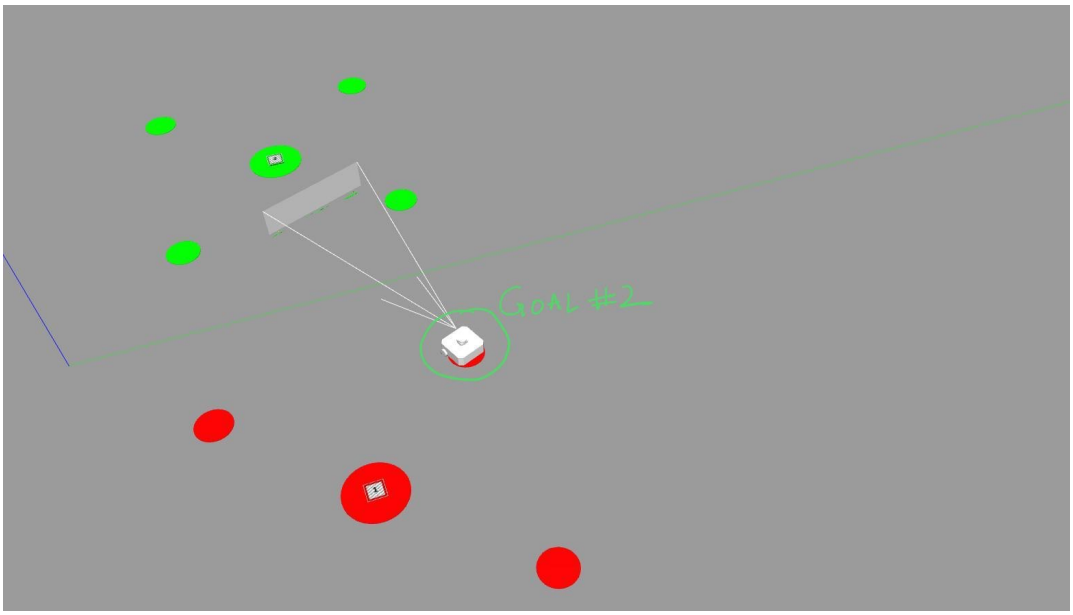

Figure 9: GOAL 2

3 Challenges

1. Building the odom_updater was challenging, writing the non-static broadcaster gave an error which took time to resolve.

   Solution A reference was used to make the broadcaster and also some help from our professor in a zoom meeting.

2. Figuring out how binding works and which functions are called and at what time.

   Solution This was solved by trial and error.

3. Making the final_destination frame dynamic.

   Solution, A string was made which was then concatenated.

4. Working simultaneously with gazebo, ROS 2 and C++ was challenging.

   Solution Displaying them on a 3 screen system to work on them simultaneously

5. Stopping the robot from rotating at the final position.

   Solution A double variable was defined for the angular.z, initializing it with 0.2 and after reaching the goal 2, c was set 0, hence stopping rotation.

4 Contribution to the project

The workflow was divided into two main parts Shivam and Patrik were responsible for completing the coding part. Chris was mainly responsible for the report and documentation which includes creating the doxygen file, while Shameek helped him by making several figures.

5 Resources

1. https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Broadcaster-Cpp.html
2. https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html
3. Lecture Slides - ENPM 809Y
4. Lecture Videos - ENPM 809Y

6 Course Feedback

The course was well organized, the provided material and used conventions made the learning process straightforward. The C++ part was the one which was more emphasized therefore the gained knowledge is strong in that field. We were also able to get in touch with ROS 2 and see some examples, however section was not that well investigated as its difficulty required. As a result, confidence was not able to be build and working on the final project was not as efficient as the other assignments. The missed assignment could have helped, if it is possible to access it that would be appreciated.

All in all we all are glad that we have taken this course, since programming knowledge on robotics related field is one of the most important skill, and professors attitude was refreshing, this kind of enthusiasm is rare. There is no doubt that the gained knowledge will be useful in the upcomings.