Fall 2022

Final Project

Introductory Programming for Robotics

December 16, 2022

Students:
Rohit Reddy Pakhala (rpakhala)

Sameer Arjun S (ssarjun)

Vineet Kumar Singh (vsingh03)

Instructors:
Z. Kootbally

Course code:
ENPM809Y

Contents

1 Introduction **3**

2 Approach **3**
2.1 Broadcaster . . . 4
2.2 Initializing Parameters and Reaching Goal 1 . . . 5
2.3 Fiducial Marker Detection . . . 6
2.4 Navigation to final goal . . . 7

3 Challenges **8**

4 Contributions to the project **10**

5 Resources **10**

6 Course Feedback **10**

List of Figures

1 Spawned TurtleBot . . . 3
2 Project procedure flow-cart . . . 4
3 Fully Connected TF tree . . . 5
4 Parameters from yaml . . . 6
5 Detection of Aruco marker . . . 7
6 Detection of Aruco marker . . . 8
7 Aruco Callback message on terminal . . . 8
8 Various Goal positions based on Frame IDs . . . 9

1 Introduction

The objective of this project is to use ROS2 to control a Turtlebot3 and simulate the results in Gazebo. The TurtleBot is equipped with an onboard camera to scan its surrounding and identify fiducial markers. The Fiducial markers are 2D representations used for image guidance, and the robot uses an onboard camera to read the marker. The marker contains information about the final goal and our objective to properly extract the data from the marker and use it to guide the bot to its final goal. When the robot is near the marker location it is given an input to rotate about its current axis to scan the environment and detect the marker. Upon successful identification of the marker, relevant data is published and the ros package reads the data and extracts the destination location in terms of (x,y) coordinates. When the robot reaches the final goal position, the task is completed.

At the start of the project, the robot is spawned at the home location as shown in Figure-1 and is later navigated towards the Fiducial marker, using a PID controller which is a closed loop controller system. This control is achieved using the *bot_controller* package. And the fiducial marker image recognition occurs using OpenCV. During the entire operation of the robot, separate publishers and subscribers are written for robot control and detection of environment. The package *ros2/tf* is used for all the frame transformations. In the next section, a breakdown of approach taken to solve the above objectives is discussed in detail.

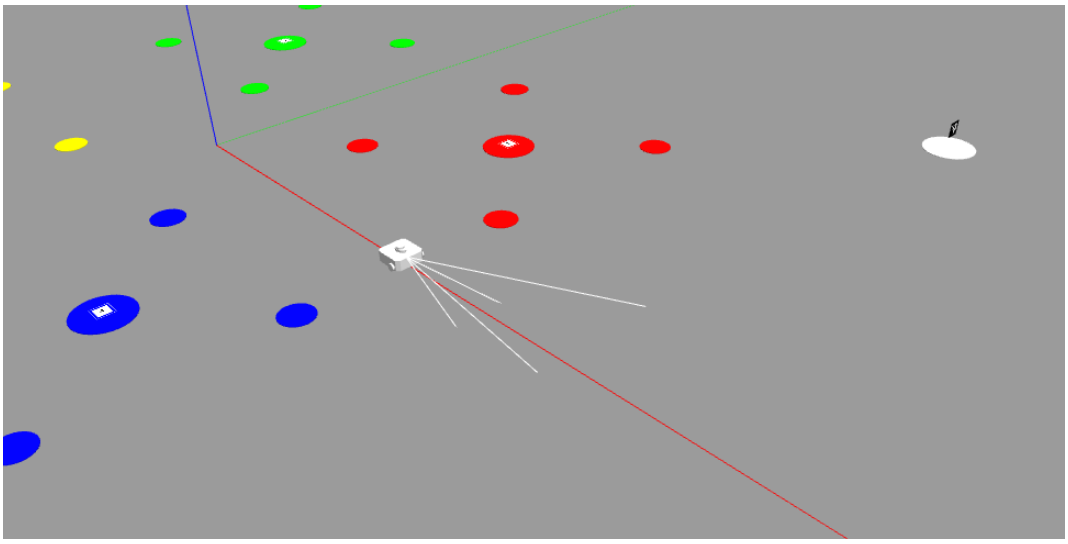


Figure 1: Spawned TurtleBot

2 Approach

The problem statement had 4 main objectives and each have been explained in detail in this section. A simple flow chart highlighting the problem breakdown and logic diagram is shown in Figure 2

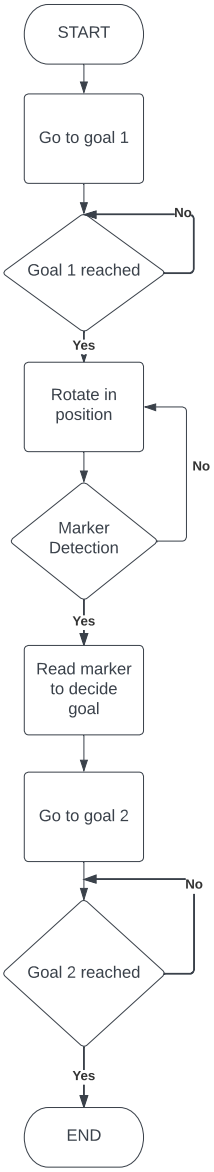


Figure 2: Project procedure flow-cart

2.1 Broadcaster

The first objective of the project is to analyze the transform (tf) tree. In the initial state of tf tree there was a disconnection between the frames *robot1/base_footprint* and *robot1/odom* and the objective is to fully connect the tree to enable the robot to get coordinates in the world frame. The *odom_updater* package was created to connect the tf tree. This package works as a dynamic broadcaster and broadcasts */robot1/base-footprint* as a child of *robot1/odom*.

The POSE (Position and orientation) of the TurtleBot was obtained by using a subscriber to the

/robot1/odom topic. A reference of type nav_msgs::msg::Odometry is used to grab the pose of the robot from *robot1/odom* topic. The received pose from the topic is transformed to the *base_footprint* in the callback function of subscriber. The callback function is also used by the broadcaster to broadcast the transformed. So this package works as a dynamic frame broadcaster. Both the subscriber and broadcaster is implemented in the same node named *dynamic_tf2_frame_publisher*.

The new package was then included in the *final.launch.py* launch file to run the node during start of the simulation. The updated tf tree after launch of simulation is visualized in Figure 3 where the two frames are connected.

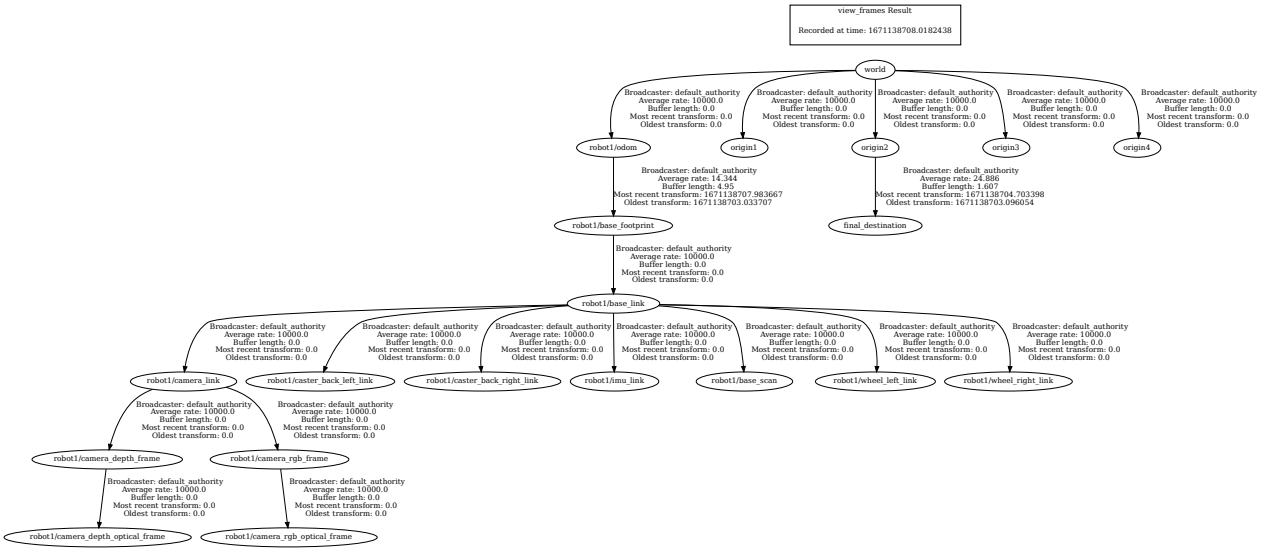


Figure 3: Fully Connected TF tree

2.2 Initializing Parameters and Reaching Goal 1

This section of the project was on initializing all the parameters of the *target_reacher* node located in *final_params.yaml* file and then including them in the launch file *final_launch.py*. Parameter initialization was done in the constructor of the class TargetReacher. With this approach the *target_reacher* node was started with the default values of the parameters. The parameters list for *target_reacher* node is shown in Figure 4. Once the parameters were initialized, the position of the fiducial marker was pulled from the default parameter list. The position of the first goal was provided by the *aruco_target* parameter in the yaml file . The robot was then sent to the goal position using the retrieved parameter values. This task was achieved using a pointer to the *m_bot_controller* package and pointing to the function goal(x,y) where the values of x and y indicate the coordinate location of the first goal.

```yaml
target_reacher: # this has to match your node Name
  ros__parameters:
    aruco_target:
      x: 6.5
      y: 5.0
    final_destination:
      # "origin1", "origin2", "origin3", "origin4"
      frame_id: "origin1"
      aruco_0:
        x: 1.0
        y: 1.0
      aruco_1:
        x: 1.0
        y: -1.0
      aruco_2:
        x: -1.0
        y: -1.0
      aruco_3:
        x: -1.0
        y: 1.0
```

Figure 4: Parameters from yaml

2.3 Fiducial Marker Detection

The robot approaching the goal position is shown in Figure 5. Fiducial markers are located near the first goal position and it contains the details of the final goal that the robot is suppose to reach. Once the robot is at goal location, the message *true* of type *std_msgs/msg/Bool* is published on the topic */goal_reached*. To read this topic, a subscriber to this topic was created named *goal_detect_subscriber* and uses the callback method *goal_detect_callback* to read the topic continuously.

Listing 1: Message published on /aruco_marker

```
---
header:
  stamp:
    sec: 24
    nanosec: 370000000
  frame_id: /robot1/camera_rgb_optical_frame
marker_ids:
- 0
poses:
- position:
    x: 0.029409973179839035
    y: -0.0374974985082491
    z: 0.24302599339159997
  orientation:
    x: 0.9953363183884367
    y: 0.00021174539509414422
    z: -0.09632379850685333
    w: -0.00522439489810022
```

Once the robot is at the goal position, and the message *true* is published, the robot is at its goal and should search for fiducial marker. To detect the fiducial marker, the TurtleBot is rotated about vertical axis so that the onboard camera scans its surrounding to locate the marker. This is achieved by publishing *velocity.angular.z* =0.2 messages on the topic */robot1/cmd_vel*. Once the marker is found by the robot camera as shown in Figure 6, a message is published on the topic */aruco_makers* by *aruco_node* as shown in Listing 1. This message shows the *marker_id* at which the final goal is needed for completing the next task. To read this topic, a subscriber named *aruco_detection_subscriber* and it subscribes to the topic */aruco_makers* using the callback method *aruco_detect_callback*. Once the callback is started, it publishes a message on the terminal indicating its start as seen in Figure 7.

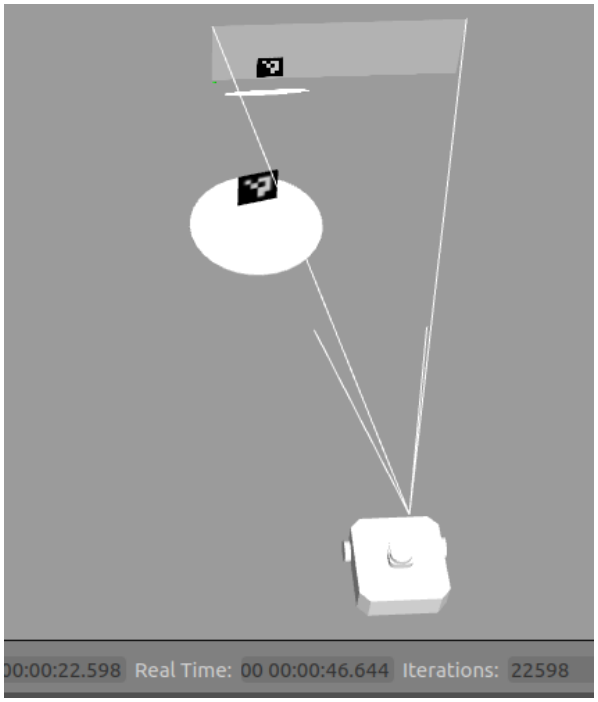


Figure 5: Detection of Aruco marker

This message is shown in the below $Figure - 3$.

2.4 Navigation to final goal

The data on the fiducial marker provides the TurtleBot with coordinates of the end goal, which is retrieved from the parameters *final-destination.aruco_i.x* and *final_destination.aruco_i.y*, with i being the ID of one of the four fiducial markers. Based on this reading of *marker_id*, the end destination coordinates are pulled from the parameter file. String concatenation is used to achieve dynamic naming of the parameter to obtain values based on the value of detected marker id.

The positions retrieved from the parameter file are in the *final_destination.frame_id* which is the world frame. Hence, a transformation from this world frame to the body frame, */robot1/odom* needs to be generated to obtain the required POSE data. For achieving this, a frame is broadcaster is created which broadcasts the frame of *final_destination.frame_id* as a new frame named *final_destination* as shown in tf tree in Figure 3. The values of transforms used is partly extracted from the parameter file as per the problem statement. This frame broadcast is achieved using a simple subscriber, and

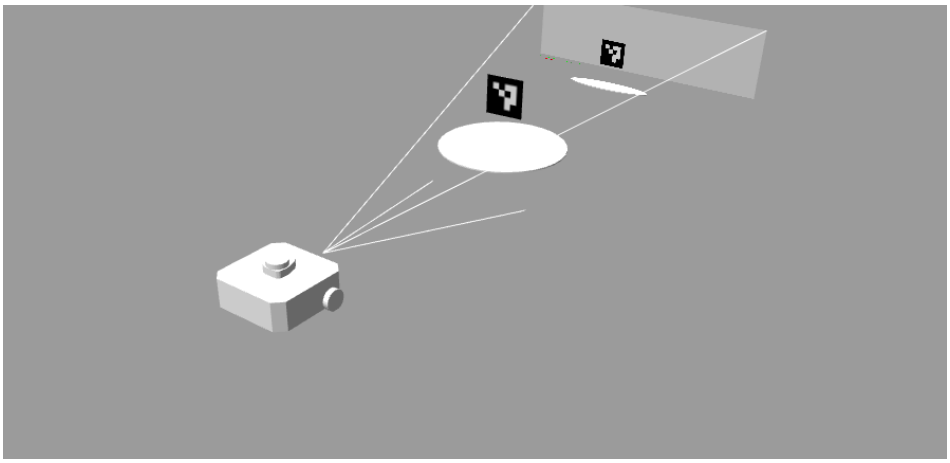


Figure 6: Detection of Aruco marker

```
[target_reacher-6] [INFO] [1671147875.565161094] [bot_controller_robot]: ********** Goal reached **********
[target_reacher-6] [INFO] [1671147891.074302447] [target_reacher]: Aruco detect callback
[target_reacher-6] [INFO] [1671147891.074388799] [target_reacher]: Broadcaster callback
[target_reacher-6] [INFO] [1671147891.074500141] [target_reacher]: Broadcaster callback
```

Figure 7: Aruco Callback message on terminal

this function is called from within the *aruco_detect_callback* method when the aruco marker is detected and its id is extracted. This new frame is then subscribed by an another method which returns the transform of final goal position in the *robot1/odom* frame.

After successfully transforming the POSE data using *ros2/tf*, the goal position is then passed on to the bot controller command and the robot moves to the final location. To check the robustness of developed control logic, the bot was tested on different fiducial markers with different frame ids and some of the positions are as shown in Figure 8., the With these steps, the project was completed and the final simulation worked as expected.

The final simulation video of the project can be accessed by clicking here.

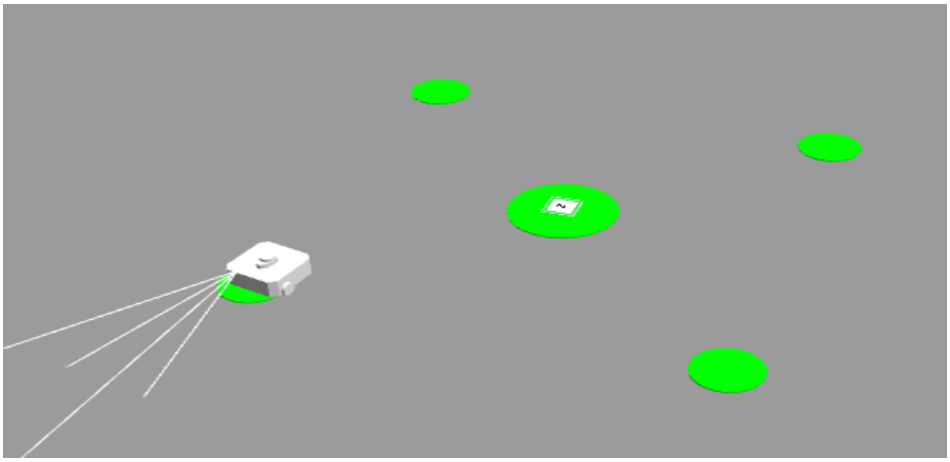
3 Challenges

Multiple challenges occurred at different stages of this project and are highlighted below in sequence.

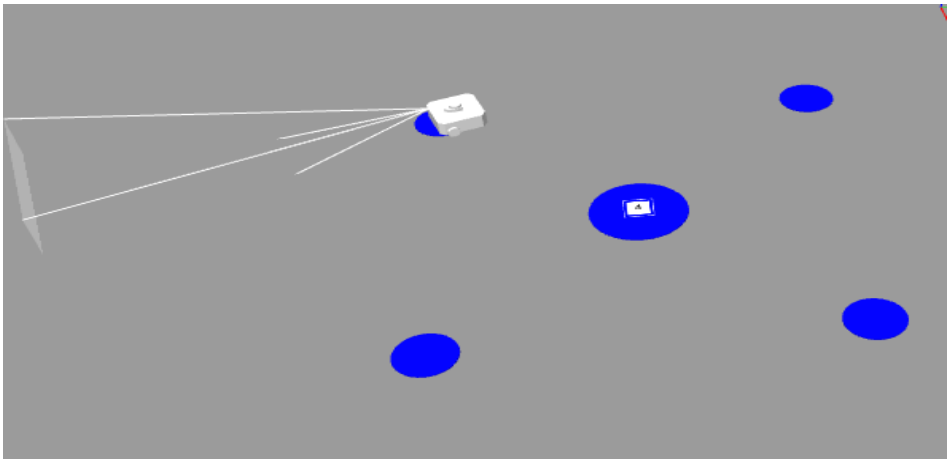
1. Writing the dynamic broadcaster was challenging as achieving simultaneous implementation of subscriber and publisher within the same node with proper callback was complicated.

2. In objective 2, implementation of subscribers to get inputs from the topics of goal reached and aruco marker detection.

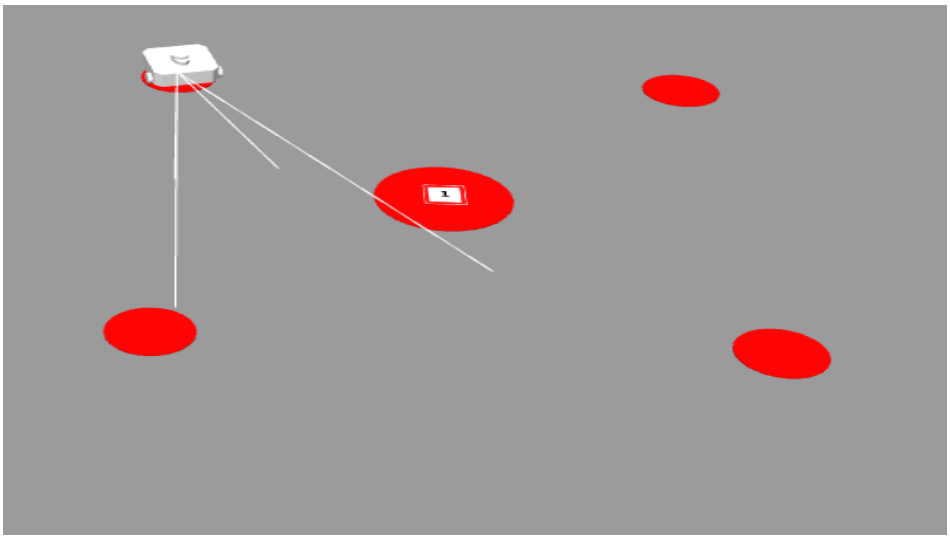
3. Since there was a frame transformation required form the world frame to the robot frame based on fiducial marker id, implementation of function for publishing new frame from within the *aruco_detection_callback* function was challenging. This issue was solved by calling the broadcaster and listener in sequential order from within the callback function of aruco detection .



(a) Frame Id as Origin 1



(b) Frame Id as Origin 2



(c) Frame Id as Origin 4

Figure 8: Various Goal positions based on Frame IDs

4 Contributions to the project

- Rohit Reddy Pakhala: Worked on aruco detection, subscribers to topics for goal reached and aruco detection, final report.
- Sameer Arjun S: Worked on parameter initialization, subscriber for robot rotation and aruco detection, final report.
- Vineet Kumar Singh: Worked on dynamic frame broadcaster for connecting tree, final goal tf transform from detected marker id, final report.

5 Resources

- Writing a Broadcaster in ROS2 (C++)
- Adding a frame in ROS2 (C++)

6 Course Feedback

- Rohit Reddy Pakhala : I really liked the course structure, lecture material and hands-on nature. The lecture material was so good that use of textbook was rarely required. came into the course thinking that it would be a basic programming course in C++ but I was wrong. I actually learned a lot and had a chance to learn Linux which is really important in industry. The problems and the assignments are of real-world problems and the way professor made us solve is also industry oriented. I really appreciate this class and look forward to how the upcoming semesters will be.
- Sameer Arjun S : I personally loved the organization of the course along with proper documentation of lecture notes and reading material, which is very useful both now and in the future for reference. The course instruction was excellent with Dr. Kootbally, and it created a well structured learning atmosphere for the students. Even though, some of us were from a non programming background, the course is well designed such that everyone is included. Additionally the weekly quizzes made sure that we are always following up with the classes, and the projects were helpful in expanding our understanding of the fundamental concepts and their application.
- Vineet Kumar Singh : The course was very well organized and implementation of code in live classes helped a lot to understand the fundamentals properly. The pace for the course was good to properly understand the topics and the projects were very hands on and really helped the learning curve. A few more assignments on ROS would help even more. Overall the course helped me learn C++ and ROS2 and the best practices to be used while using these languages.