Fall 2022

Final Project

Introduction to Robot Programming

December 16, 2022

Students:
Aaqib Barodawala(abarodaw)

Ankur Chavan (achavan1)

Krishna Hundekari (krishnah)

Instructors:
Z. Kootbally

Course code:
ENPM809Y

Contents

1 Introduction

The main goal of the project is to develop a ROS2 package for the turtlebot to make it travel to a place if the coordinates are given as a goal. It uses OpenCV to read the fiducial markers to retrieve the goal position. There are a lot of applications similar to the problem statement that is tackled in this project. One such application could be in warehouse management. On the warehouse floor, there can be multiple storage units and multiple stacks of items. Using the fiducial markers on the items to mark on which storage units these should be stored, the turtlebot can be used to take these items to the respective storage units.

The turtlebot is supposed to go to the first goal, from where it can find the fiducial marker that has the information about the final destination goal. There is a total of 16 final destinations. Once identified, the fiducial marker and the final destination goal, the turtlebot is supposed to reach the final destination.

In this project, OpenCV is used to identify the fiducial marker and extract the final destination from the fiducial marker. Using a proportional controller, the turtlebot reaches the set goal. The deliverable tasks in this project submission are the odom_updater and target_reacher packages. The approach used (refer Figure2) to develop these packages and the challenges faced during the development are expounded in this report.

2 Approach

1. **Cloning the package :**

   Created a new workspace and cloned the package given in the project guidelines. In tb3_gazebo exported the environment variable by passing the absolute path of the model to incorporate the marker and other aspects of the project as well as created an alias to create a frame by updating the .bashrc. Installed the package for OpenCV.

2. **Create a Broadcaster :**

   Initially, when checked for frames, the frame robot1/base_footprint was not connected to frame robot1/odom. The first task was to connect the frame robot1/base_footprint as the child of the frame robot1/odom.

   To do this, there needed a broadcaster node called odom_updater. Created a package odom_updator with rclcpp, geometry_ msgs, nav_msgs, and tf2_ros2 as dependencies.

   As frame robot1/base_footprint is a moving frame which caused the frame to change relative to the frame robot1/odom, hence node odom_updater is a non-static broadcaster. There is a callback method in the subscriber that broadcasts the pose information to create the transform.

   As shown in the pseudocode this method takes in a shared pointer to an Odometry message and broadcasts a TransformStamped message based on the data in the Odometry message. The TransformStamped message contains information about the position and orientation of frame robot1/base_footprint in relation to frame robot1/odom. The function sets frame IDs of the TransformStamped message passed in the program as child and parent frames, as well as its position and orientation values, and then uses a tf broadcaster to send the message.

   To start the node odom_updater and run the node indefinitely, the launch file “final.launch.py” was updated with a variable “start_odom_updater_cmd” to start the node of the broadcaster and the command ld.add_action(start_odom_updater_cmd) is added at the end so it will keep on running until its explicitly stopped by pressing “Ctr + c”. To validate that odom_updater is working properly, refer the Figure 1 where the robot1/base_footprint is now a child frame of robot1/odom. Figure with all connected frames:

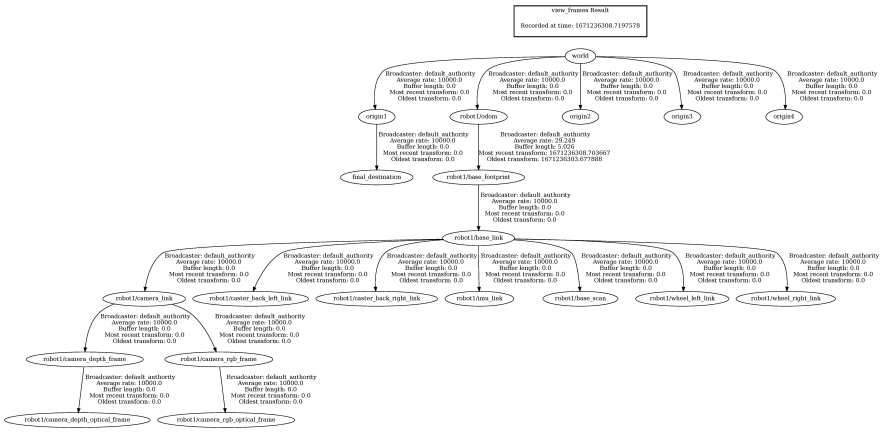

Figure 1: Connected frame tree

3. **Load parameters from final_params.yaml file :**

   Created a target_reacher node, and there defined a constructor with class members. The target_reacher node attributes of the constructor are initiated from the parameters from final_params.yaml. The syntax used to retrieve the parameters from the final_params.yaml:

   this-¿declare_parameter¡type¿(¡parameter name¿);

   All the parameters like goal one coordinates, findestination frame Id, and final destination coordinates associated with four different fiducial markers are retrieved and initiated different variables in the constructor of the target_reacher which will accessed in the further steps. To start the node target_reacher and run the node indefinitely, the launch file "final.launch.py" was updated with a variable "start_target_reacher_cmd" to start the node and the command ld.add_action(start_target_reacher_cmd) is added at the end so it will keep on running until its explicitly stopped by pressing "Ctr + c".

4. **Move to goal number#1:**

   The parameters that are needed to define the coordinates of the goal #1 are aruco_target_x and aruco_target_y which are already retrieved from the final_params.yaml and defined in the constructor class of target_reacher.

   Using the retrieved parameters aruco_taget_x and aruco_target_y as the respective coordinates X and Y, the set_goal method is called from the m_bot_controller class. The method used the proportional controller to drive the turtlebot to the target location coorrdinates.

   Once goal #1 is reached, the message is published on the topic goal_reached that the goal has been reached. From this position, it is guaranteed that the fiducial marker can be found.

5. **Find the fiducial marker:**

   Subscribed to topic goal_reached, and to check if the goal has actually reached or not. Once the goal#1 is reached, the turtlebot is made to rotate in in the same position until it finds the fiducial marker.

   m_ publisher_cmd_vel: To rotate in the place, the time_callaback is defined to publish the angular velocity on the the frame-robot1/cmd_vel. This twist message makes the turtlebot turn about itself.

   Once aruco marker is found by the turtlebot, the message is published on the topic aruco_markers.

6. **Retrieve goal coordinates:**

   Next task is to subscribe to aruco markers using subscriber m_subscriber_aruco. Defined a method cb1, which will extract the marker_ids filed from the published messages. Based on the detected marker_id, the final destination coordinates are extracted from the parameters defined in the final_param.yaml file.

7. **Compute the goal in robot1/odom frame:**

   Once retrieved the final destination coordinates, fd_aruco¡i¿_x and fd_aruco¡i¿_y where i is the frame id in which the retrieved coordinates are defined. The retrieved coordinates are not in frame robot1/odom, hence there needed a transformation to map these retrieved coordinates to the frame robot1/odom from frame i.

   As this transformation is not time-varying, we defined a static broadcaster to publish the TransformStamped message regarding the position and orientation of the final_destination_frame (frame i) in relation to the frame robot1/odom. Wrote a listener to get the final destination coordinates in the frame robot1/odom which will be the final destination for the turtlebot.

8. **Move the robot to the final destination:**

   From the coordinates of the final destination in the robot1/odom frame, the set_goal method is called and the tutlebot is moved to the final destination which is one of the 16 smaller dots in the Gazebo environment.

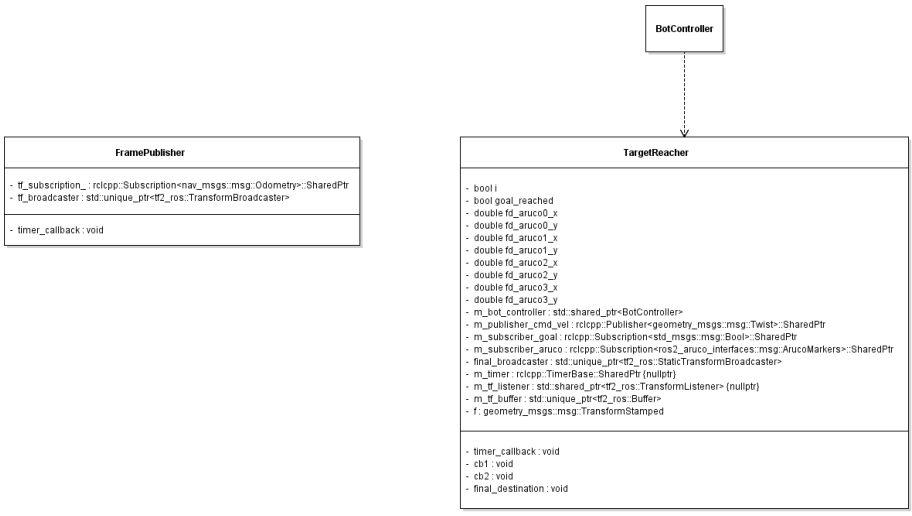

Figure 2: UML Class Diagram

3 Challenges

The challenges faced in this project are listed below -

1. **Struggled a lot with Syntax** :

   Someone who are just introduced to ROS2, proper syntax use was the frequent challenge throughout the project execution. Even though the approach and the C plus plus programming part was clear, we had to refer a lot of online resources and the class notes to write the ROS2 package.

2. **What variable name should be retrieved from the pose information?**

   While writing the frame publisher in the node odom-updater, issues were faced while retrieving the position and orientation information of the frames from the odom topic. Refer the figure 3, here one can see the complex structure in defining the parameters to retrieve. It took us a while to figure out a proper way to call the pose information from the odom topic.

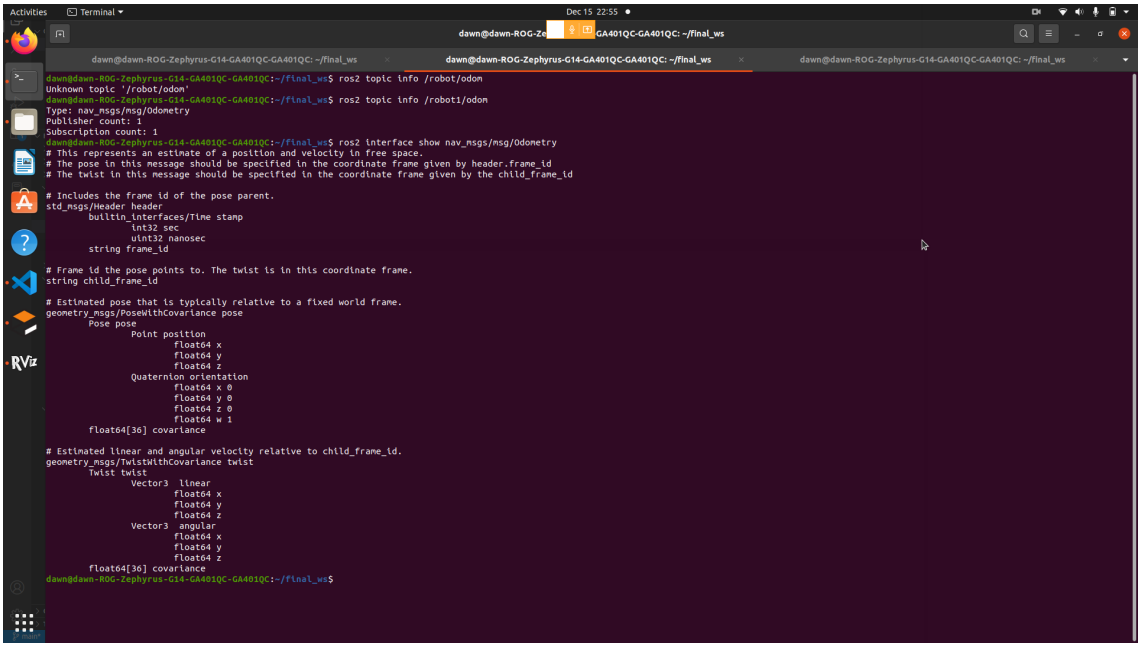

Figure 3: Topic problem faced

3. **Why turtlebot rotating around itself at the final destination?**

   The robot, after reaching the second goal (final destination coordinates) used to rotate again searching for an aruco marker. The problem occurred because the "goal_reached" topic was getting subscribed again once the goal was reached. To solve this issue, we made use of a boolean variable that didn't allow the subscriber to enter the if loop that published velocity commands for rotation.

4. **Unable to access the method "create_wall_time":**

   Spent a lot of time debugging the code and trying to get the reason why it was not accessing the method create_wall_time. Later realized that the namespace was commented, even though this does not look like a challenge, but it really was!

5. **Wrong message type!**

   While doing the broadcast_updater, the wrong message type was used for orientation information and used twist message type instead of the odometry message type. This took a lot of time to troubleshoot and with the help from the professor over mail communication the issue was resolved.

4 Final Results

From the Figure 4, it is verified that the turtlebot is successfully spawned in the gazebo environment at the defined initial position.

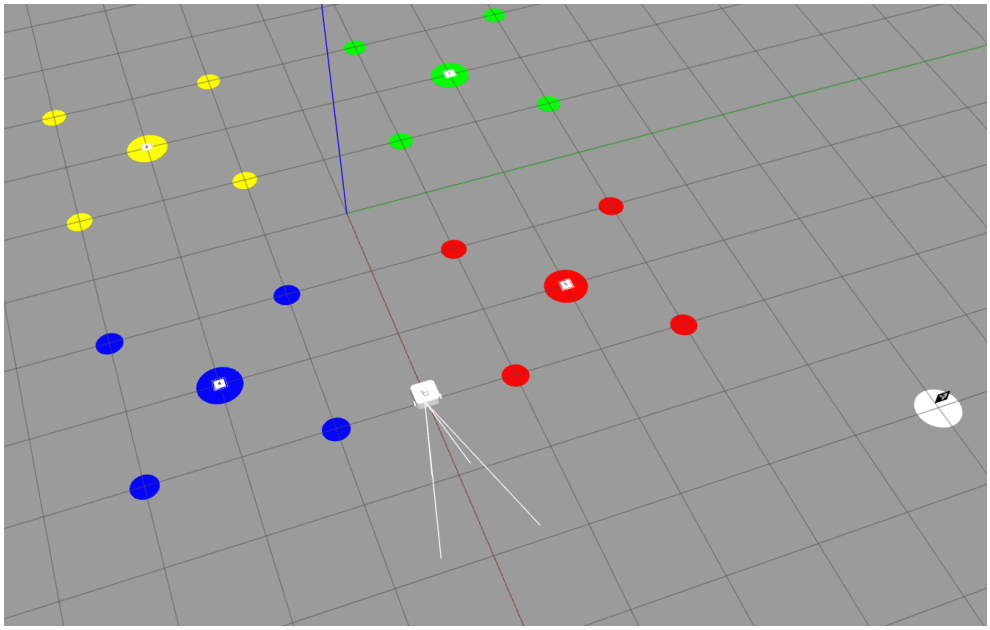

Figure 4: Turtlebot spawned in Gazebo environment at initial position

From here the turtlebot will move to the position goal —1, from here the fiducial marker will be visible. The robot starts rotating around itself at goal—1 unitl the fiducial marker is found. From the figure 5, it is seen that the turtlebot has found the fiducial marker and it can be clearly seen in the Rviz camera feed as well.

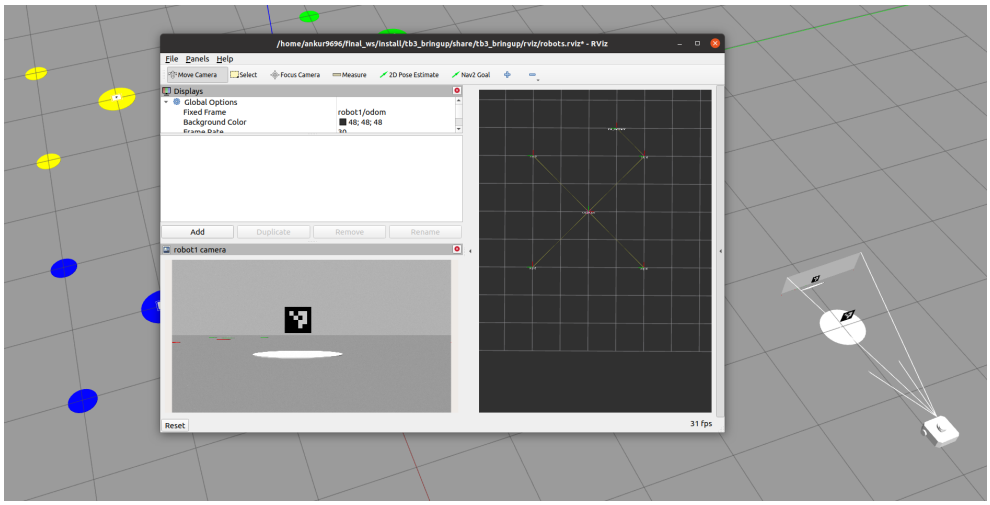

Figure 5: Turtlebot scanning fiducial marker

Once the fiducial marker is found, the turtlebot knows the final destination (goal#2) and starts approaching the goal#2. From the Figure 6, it can be seen that the turtlebot has reached the goal2 successfully hence validating our submitted package.

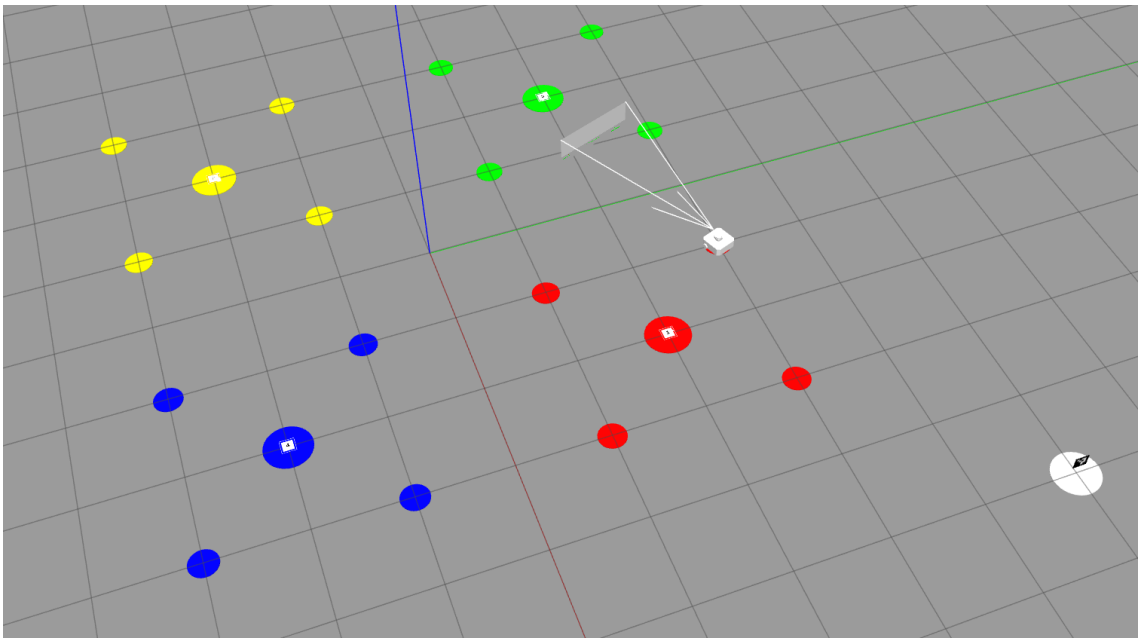

Figure 6: Turtlebot reached final destination

5 Pseudo Code

1. Pseudo Code for broadcast_updator:

```
Define a timer callback function that takes in a shared pointer to an Odometry message

    Create a TransformStamped message

    Set the stamp and parent and child frames of the TransformStamped message

    Set the translation values of the TransformStamped message

    Set the rotation values of the TransformStamped message

    Use the tf broadcaster to send the TransformStamped message
```

2. Pseudo Code for target_reacher:

```
Define a method to get the final destination coordinates in the robot1/odom frame

Create a TransformStamped message

Set the stamp and parent and child frames of the TransformStamped message
```

```
*****************************************************************************

    Set the tranlational position of the "final\_destination" frame based on the ID of the Aruco
        if id is 0 set the final destination of corresponing id
        else if id is 1 set the final destination of corresponing id
        else if id is 2 set the final destination of corresponing id
        else if is 3 set the final destination of corresponing id

    Set the rotation values of the TransformStamped message

    Use the tf broadcaster to send the TransformStamped message

    set the boolean to true which will be used to trigger broacasting



Define callback1 method to set the final destination based on the frame id

    Get the IDs of the detected Aruco markers
      If the first Aruco marker has ID 0, call final\_destination with argument 0
    else If the first Aruco marker has ID 1, call final\_destination with argument 1
    else If the first Aruco marker has ID 2, call final\_destination with argument 2
    else If the first Aruco marker has ID 3, call final\_destination with argument 3


Define a timer callback for checking if the goal is reached and making the turtlebot rotate

    if the data field of the message is true
        Create a Twist message

        If the goal has been reached, set the velocity to rotate in place

        else the goal has not been reached, set the velocity to zero

        Publish the velocity command

        Reset the goal_reached flag

        Create an empty ArucoMarkers message


Define callback 2 to listen to the transformation betwenn final destination and respective origi

    if the flag i is true

        Create a TransformStamped message

        try
        Look up the transformation between "robot1/odom" and "final\_destination" frames

        catch
        Catch any exception that may occur when looking up the transformation

        Set the goal position using the x and y components of the transformation

*****************************************************************************
```

6 Resources

- Git Clone
- ROS2 Documentation
- Writing the broadcaster
- Class Notes

7 Contributions

Contributions of each member			
Tasks	**Aaqib**	**Ankur**	**Krishna**
1	✓	✓	✓
2	✓	✓	✓
3	✓	✓	✓
4	✓	✓	✓
Report	✓	✓	✓

8 Course Feedback

The course is very well organised, and progresses in a manner that makes learning programming easy for beginner.