ENPM809Y – Introductory Robot Programming

Final Project Report

Instructor:

Z. Kootbally

ENPM809Y

Authors:

Aditi Bhoir (119197257)

Sarin Ann Mathew (119390382)

Aditya Chaugule (119280539)

Table of Contents

List of Figures

Introduction

Robotics delves into the localization and object detection as an essential framework for performing simultaneous localization and mapping (SLAM) through a set of complex algorithms working synchronously.

The objective of the project is to deploy a TurtleBot robot using fiducial based localization in an environment. The robot moves to a reference goal then scans for a reference target given by an aruco marker. The aruco markers provide the final destination target positions by publishing to topic. The robot then moves towards the final destination.

The project explores the application method of localization and navigation of indoor mobile robots using landmarks as aruco markers to identify the final target position. The orientation and position

Project Outline

The goal of the project is to launch a TurtleBot in a gazebo environment to move the robot to a given destination, detect a fiducial marker and move to the final destination given by the fiducial marker.

Summary of steps:

1. Create an odom_updater broadcaster node connecting the **base_footprint** of robot to the **odom topic** in world frame
2. Update the package target reacher to implement
3. Move robot to the initial goal and rotate till the fiducial marker is detected and then traverse to the final destination target.

Approach

The Workspace – finals_ws was setup by cloning the provided GitHub repository – Reference Repository. Further, the entire project was sourced to GitHub by creating a new repository for the Project Group and the workspace was setup in Visual Studio Code environment.

Project Github Repository - https://github.com/Abhoir1/809y_project.git

The project is segmented into two objectives. For each objective, the deliverable was determined in regard to the implementation. Experimentation was performed on the algorithm implementation for each objective for an array of markers to validate the robustness of the algorithm. A detailed account on the approach for the project is documented henceforth.

Objective 1

Initially the **/robot1/base_footprint** topic was disconnected from the world node. A broadcaster node – odom_updater was implemented to initiate the robot odometry in reference to the gazebo world frame. This was achieved by creating a subscription to nav_msgs/Odometry. The implementation is mentioned as follows:

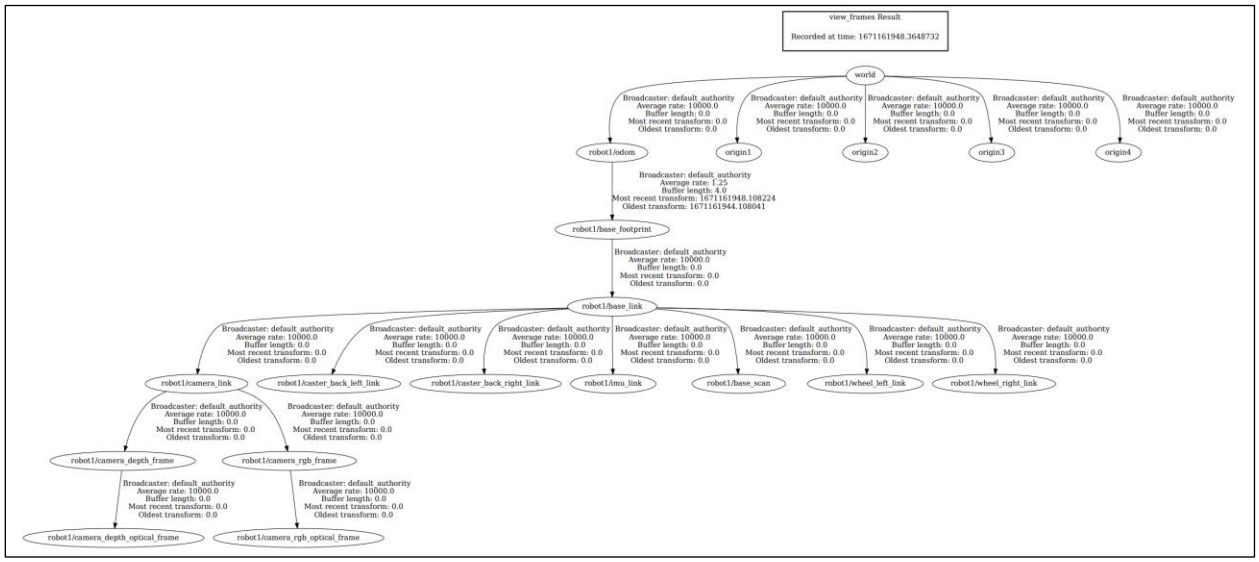

Figure 1: Broadcasting /robot1/base_footprint as a child of robot1/odom

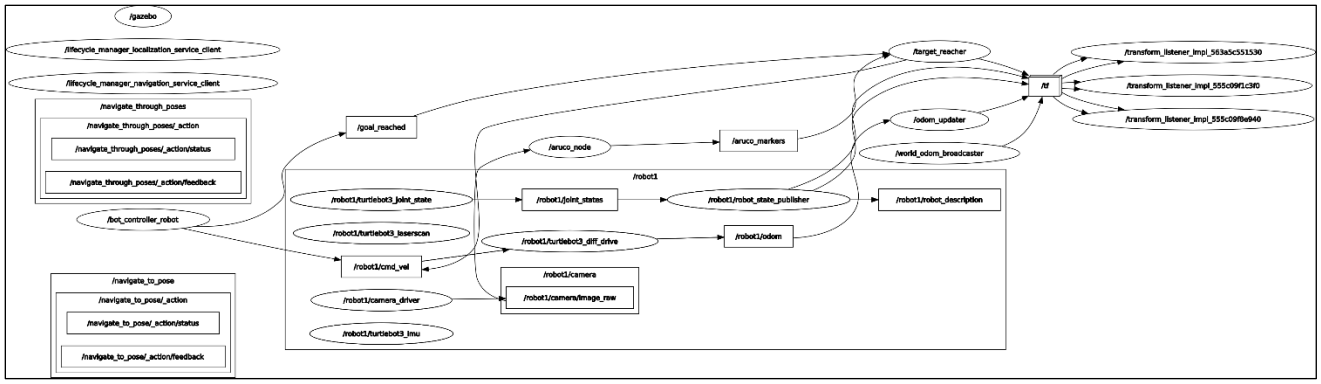

Figure 2: robot1/odom publishing to odom_updater

Include statements

- **<rclcpp>** is included to implement the rclcpp:Node class
- **<geometry_msgs>** provide messages for common geometric primitives such as points, vectors
- **<geometry_msgs /msg/transform_stamped.hpp>** to access the TransformStamped message and publish to the transformation tree to express transform between frame_id to child frame_id used by the tf2 package
- **<geometry_msgs/msgs/pose>** representation of position and orientation in free space
- **<tf2_ros/transform_broadcaster>** to publish coordinate frame transform information

Class Implementation

Initialization

- An odom updater class was initialized as a constructor to create an odom node
- The transform broadcaster is initialized as a unique pointer
- The odom subscriber callback is initialized as a subscription to **nav_msgs/Odometry** through **/robot1/odom** topic
- The private attributes to Transform Broadcaster is created
- The odom callback method is initialzied

Implementation

- An odom updater callback method is implemented by defining the broadcaster as transform from **/robot1/odom** to **/robot1/base_footprint**

Objective 2

This objective focusses on defining the attributes, methods and the class implementation for the robot to move towards a specific goal, locate the aruco marker and identify the message published on the **/aruco_markers**

Include Statements

- **<rclcpp >** is included to implement the rclcpp:Node class
- **<target_reacher.h>** is declared to defined methods into the cpp executable
- **<tf2/LinearMath/Quaternion.h>** implements quaternion to perform linear algebra rotations
- **<geometry_msgs/msg/pose.hpp>** representation of position and orientation in free space

Class Implementation

Initialization:

- The **Target Reacher** node is initialized as a Target Reacher constructor to create a shared pointer to bot_controller
- The **aruco_target** parameters are declared to identify the x and y positions of the frame id
- The **set_goal()** member function is referenced from the bot_controller
- A Goal reached subscriber timer callback is initialized using **std::bind** to **goal_reached_callback()**
- Aa publisher to **robot1/cmd_vel** is initialized through **geometry_msgs** of type Twist
- The static transform broadcaster is instantiated
- Private attributes as subscriptions to goal and **aruco_marker** and publisher to **geometry_msgs** are created
- The goal reached and final destination transform methods are initialized

Implementation:

- The method for reaching goal is developed. Condition for the goal and aruco_marker detection is checked and the robot can be moved with a set angular velocity of 0.2 by publishing to the cmd_vel topic
- The aruco_marker callback methods are defined to retrieve the final destination position from the specific frame id. Then, the **setgoal()** method is called from the bot_controller to move the robot.
- The methods to broadcast final frames and compute the goal in odom frame are defined using **geometry_msgs/TransformStamped** msg

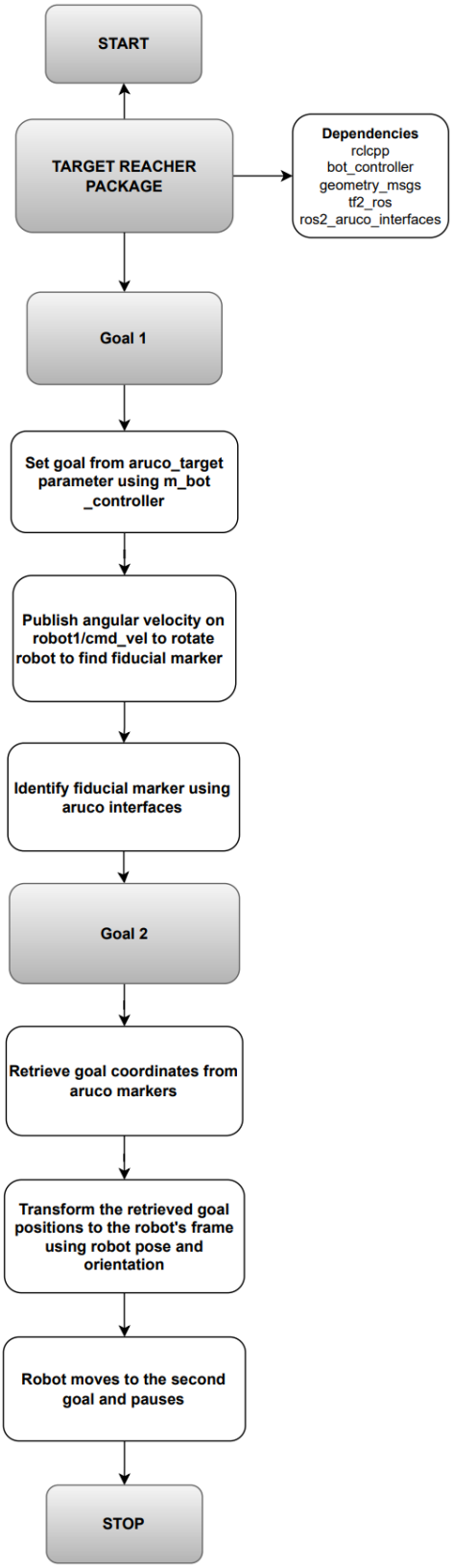

Figure 3: Flowchart for method implementation for robot movement from position to target

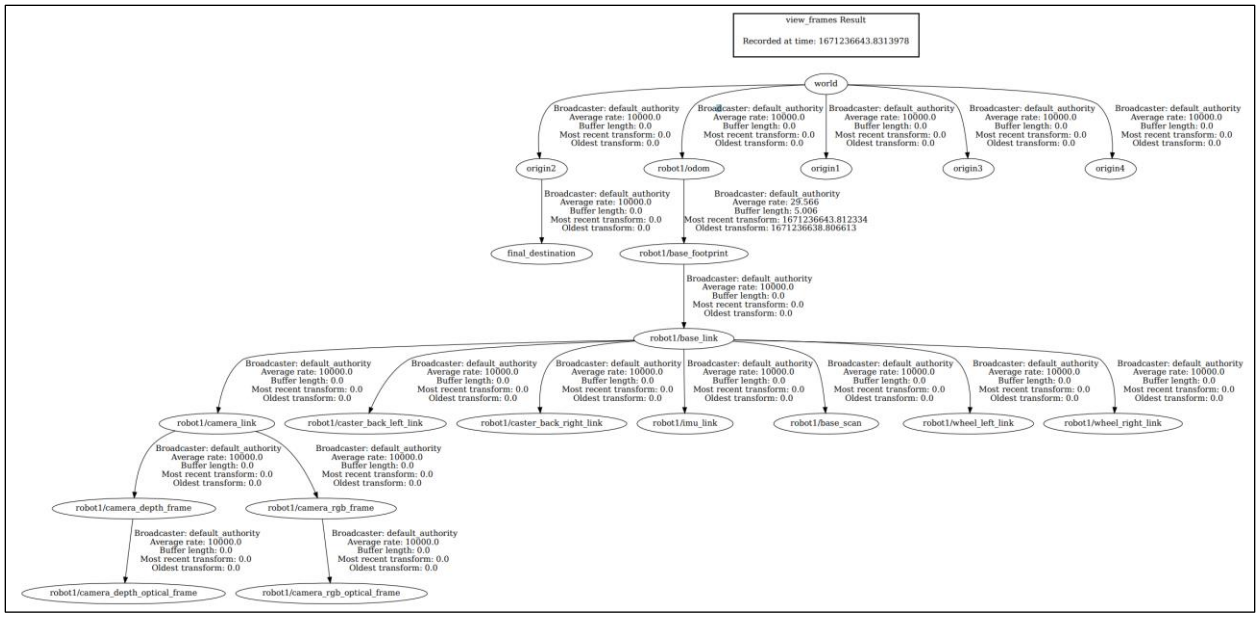

Figure 4: Final Destination Mapping

Gazebo Simulation

The implementation was tested using Rviz and Gazebo simulation environment. The TurtleBot model waffle was used as the robot for simulation. The environment was launched through **ros2 launch tb3_bringup final.launch.py** to observe the simulation operation. A demonstration of the working simulation environment is linked below.

Simulation Link – Click here

Challenges

The challenges faced during the project code implementation have been listed below:

- A few cmake list dependencies listing was missed out on initial package creation. Although addition of the dependencies manually gave a deep insight into the Cmake Lists
- The robot markers did not spawn in two systems. Hence the TurtleBot could not identify the aruco marker. Modular code development and implementation was carried out to ensure that that each individual contributes to the project code implementation and experimentation.
- Convoluted syntax of smart pointers along with namespaces make it harder to track the manipulated variable and methods.
- TurtleBot spawning issues in the gazebo environment. The TurtleBot although spawned in empty_world.launch.py but did not spawn in final.launch.py in one system.
- Package build inconsistencies and dependencies overwriting when ROS 1 and ROS 2 are enabled on the same system. Rclcpp is not referenced a few times was observed.

Contributions

Aditi Bhoir:

- Surveying pre-built code packages given in the GitHub respository
- Creating Broadcaster and Subscriber nodes
- Code Analysis

Sarin Ann Mathew:

- Method to find the fiducial marker
- Algorithm to move robot from goal to target
- Code Experimentation in gazebo environment

Aditya Chaugule:

- Documentation
- Report
- Code Experimentation in gazebo environment

Resources

1. Documentation - ROS Wiki
2. TurtleBot3 (robotis.com)
3. Writing a broadcaster (C++) — ROS 2 Documentation: Galactic documentation
4. https://dev.to/dance2die/push-git-cloned-repository-to-your-own-on-github-1ili