Fall 2022

Final Project

Introductory Robot Programming

December 17, 2022

Student:
Darshit Miteshkumar Desai(118551722)
Vamshi Kalavagunta(119126332)
Vinay Krishna Bukka(118176680)

Instructors:
Zeid Kootbally

Group:
21

Course code:
ENPM809Y

Contents

List of Figures

1 Introduction

The project aims to perform automated tasks in the Industry to make a robot move along different points on a shop floor. Fiducial markers are used on different objects in the industry so that the robot can detect and transport the object to the desired location retrieved when the marker is scanned. This project makes use of Simulation environment Gazebo and ROS2 Galactic to achieve the task of making the robot to move between two goal points. The robot model used in this project is Turtlebot3. The robot is integrated with proportional controller to reach its desired goal each time when the goal is set by proportionally changing speed and orientation based on feedback. The Gazebo environment contains four sets of final goals differentiated by different colours and one fixed goal. The robot has to reach the fixed goal and scan for Fiducial Marker called ArUco to retrieve the final goal's location. A depiction of above description is shown in Figure (1) below.

The objective of the project is to reach the first goal position which is near to the ArUco marker location and rotate the robot in place to scan the marker to fetch the location of the final goal. Four different ArUco markers with predefined positions of final goal posts near four origins. After getting the final goal post position, the turtlebot should reach the final destination. The following report discusses about the approach in achieving the task, challenges faced and resources utilised to solve the problem both effectively and efficiently.

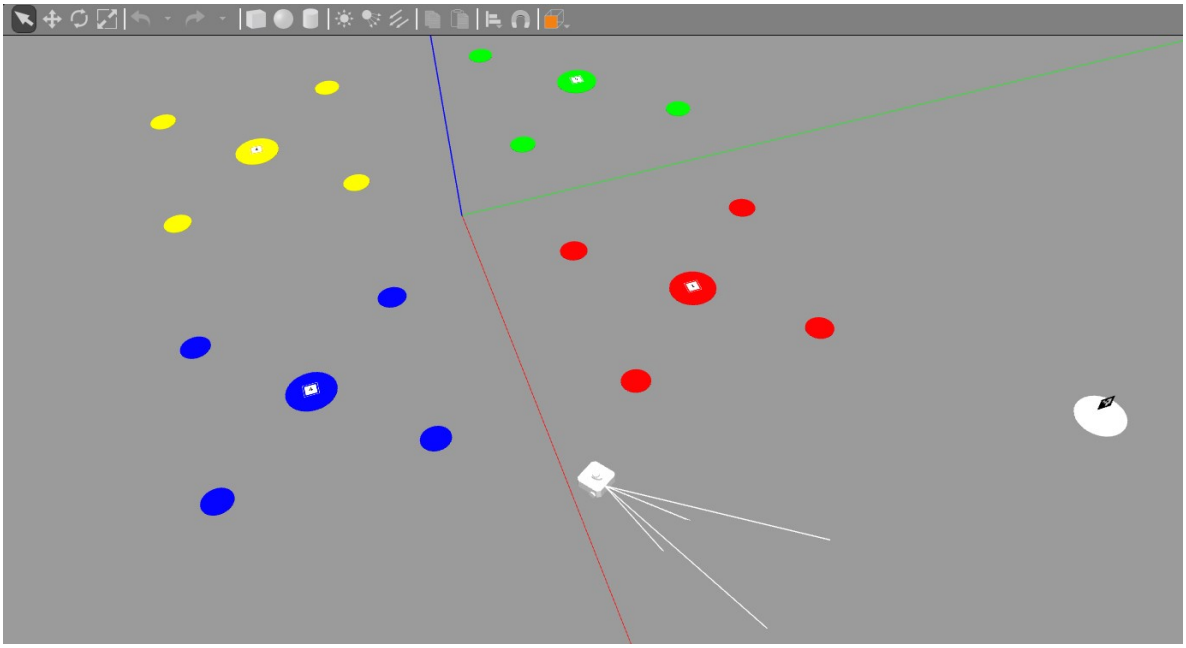

Figure 1: Gazebo world with ArUco marker and Origin frames, Robot enroute to fixed goal

2 Approach

The problem presented is planned to complete in a general software development approach. First a detailed study of the requirements and provided package is done. A High level design is roughly made dividing the problem into number of tasks and to follow procedural programming approach for each task and then optimise the optimise the task following object oriented approach. The general flow-diagram of the process and algorithm designed is shown in Figure (2). After the design, the approach followed in completing each individual task is explained below

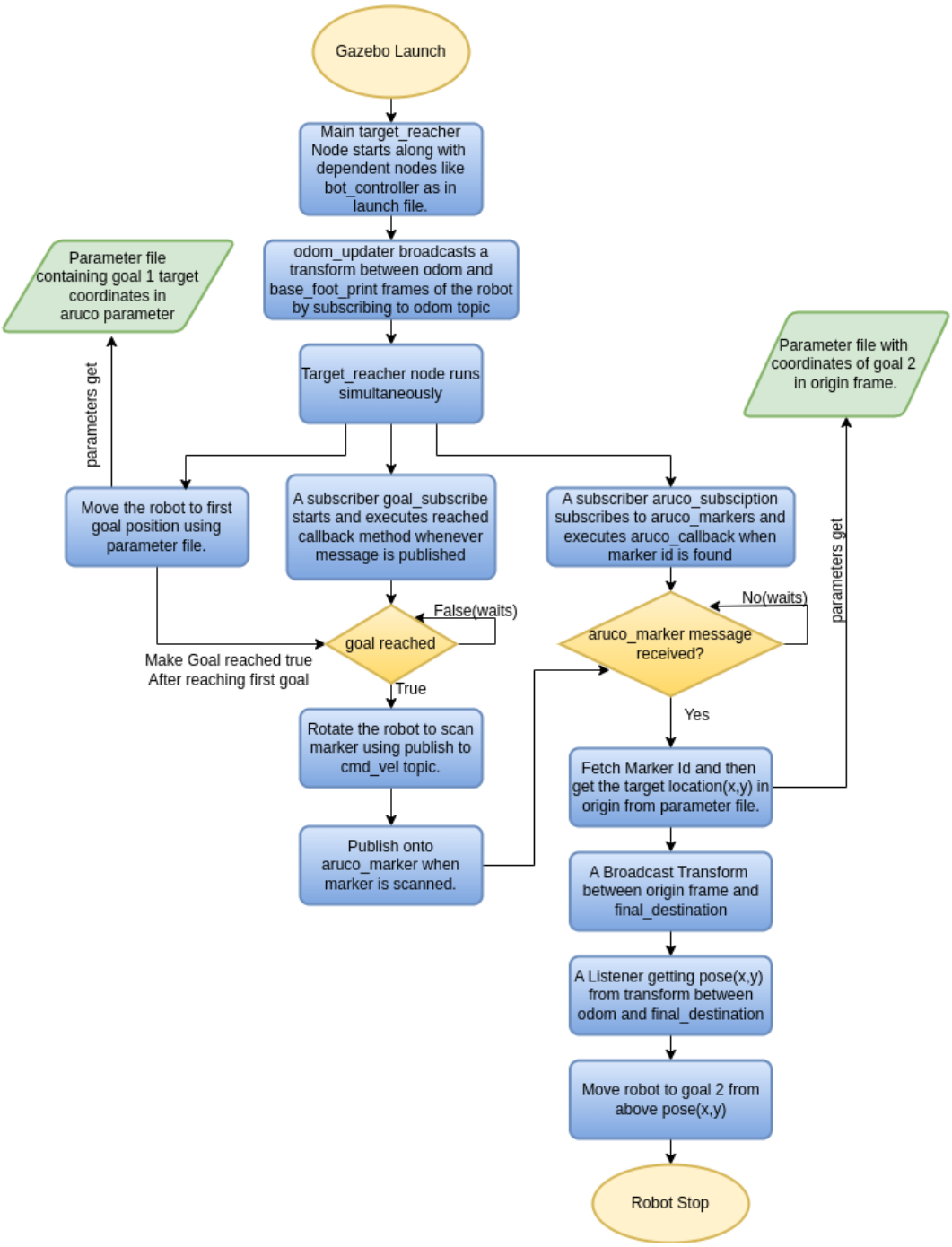

Figure 2: Flow Diagram of whole process

2.1 Broadcaster Creation

After launching the application, on observing the frames tree it clearly shows that all the frames are not connected properly. It has two disconnected trees. The first task to be performed is to connect this disconnected trees. To connect the trees, a package called odom updater is created with dependencies such as rclcpp,geometry msgs,nav msgs and tf2 ros. In the package odom updater, a node is created to broadcast the transformation between robot1/base_footprint and robot1/odom frames. To broadcast a message into this transformation the current pose of the frames is required. Therefore, after referring to [1], a subscriber is created to the topic robot1/odom to retrieve the pose of the robot in odom. From the message received on subscriber, a transformed message containing position and orientation is created and then that message information is broadcasted between two frames leading to a link creation between the two frames.

The UML class diagram for frame publisher with odom updater is shown in figure (3). Note that the methods and variables implemented for this class has been encapsulated and abstracted. This topology is followed to ensure that no methods are called directly from the main function. Only the constructor of the class is accessible making sure that correct initialization of the variables take place during the call to constructor.

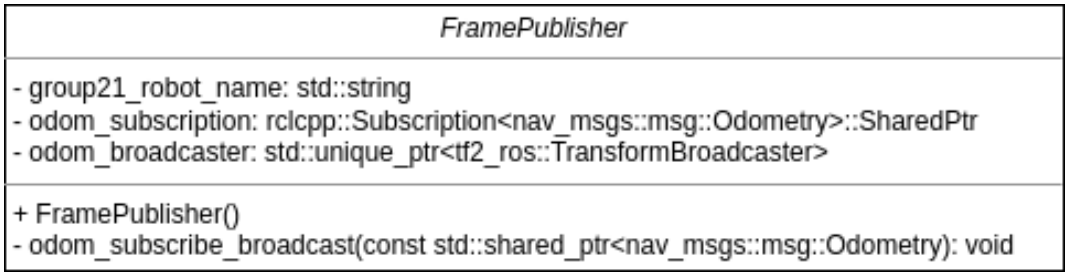

Figure 3: UML class diagram for FramePublisher

2.2 Moving to ArUco Target Location

The next task is to make the robot move from origin to target location which is the first goal position. The target reacher package is used to achieve this task. After understanding the structure and data in final parameters yaml file from which the aruco target location(x,y) is to be fetched, these parameters are declared in target reacher node to be used further. The turtlebot is then moved to the ArUco marker location from the target location coordinates fetched from the parameter server. This location is passed onto set_goal which is a method of bot_controller and made the robot move. Once the goal is reached, a message called goal reached is published. The turtlebot reaching the goal 1 is shown in figure (5)in the gazebo environment.

A UML class diagram design for target reacher is shown in 4. A similar approach as Section 2.1 for class design of the following class is followed to ensure encapsulation and abstraction of class member functions and attributes.

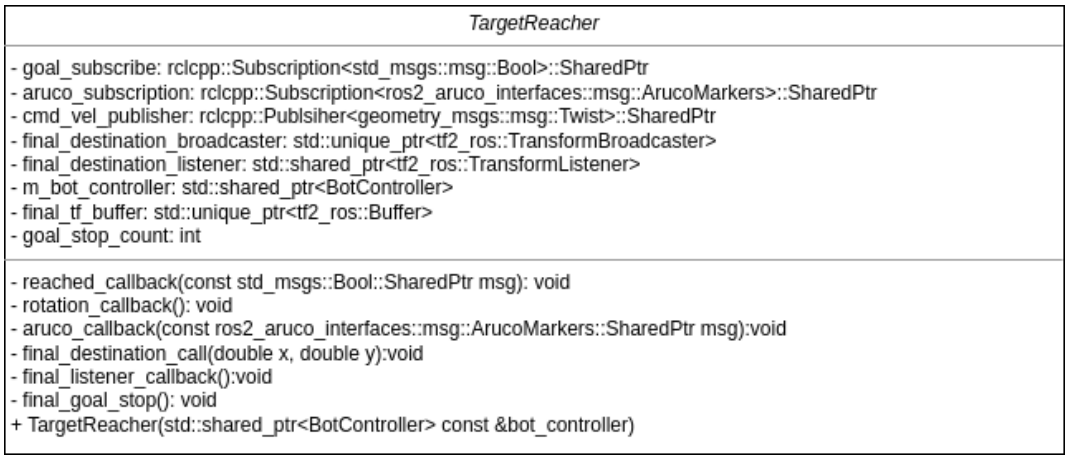

Figure 4: UML class diagram for TargetReacher class

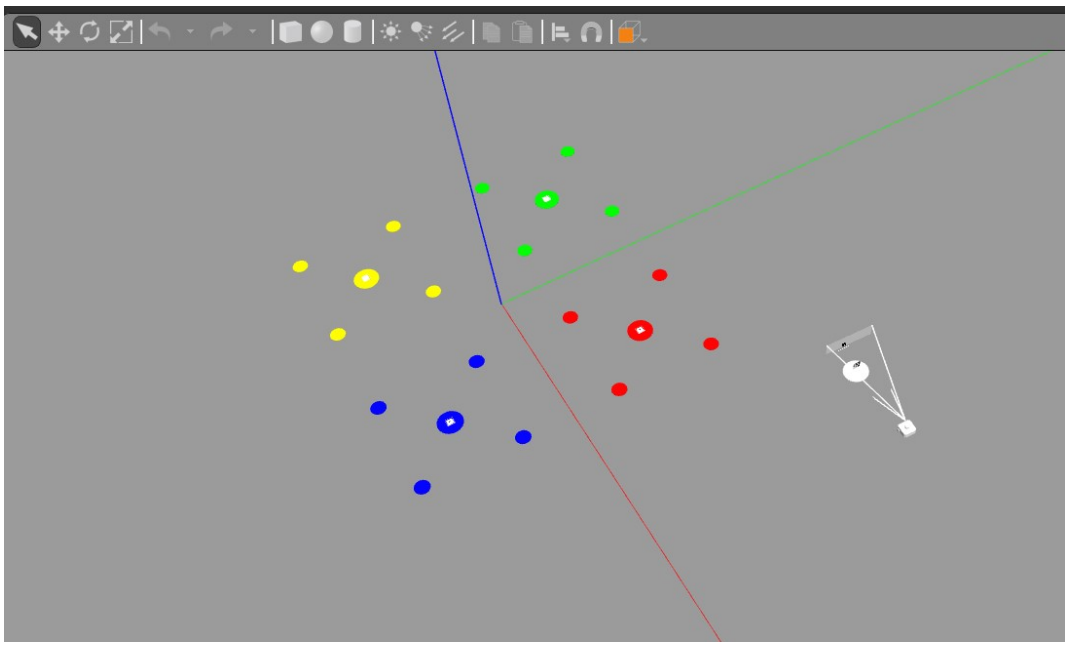

Figure 5: Turtlebot reaches goal 1, angular velocity command given to scan the ArUco marker

2.3 Locate the ArUco marker

To locate the AuRco marker the robot has to rotate to scan for the marker. To do this a confirmation is first needed that robot has reached the target location which is obtained from /goal_reached topic. So, once the robot reaches target location, a success message is published onto /goal_reached topic. After receiving the message, the robot starts rotating by publishing the command velocity on /robot1/cmd vel. Here angular velocity 0.2 is given and the linear velocity is kept zero. Once the marker is scanned, a message containing marker information and position information is published on topic /aruco_markers.

2.4 Parameters retrieval from the marker scanned

A subscriber to /aruco_markers is created to get the marker id of the scanned marker. Thereafter, to retrieve the parameters for the aruco marker message published, all the parameters from parameter server are declared. Dynamic strings are initialised to form coordinates for final goal which are retrieved from parameter server in origin frame based on marker id. The final coordinates in origin frame are (final_destination.aruco_i.x,final_destination.aruco_i.y) and "i" being the marker id.

2.5 Move to Final Goal

The retrieved coordinates from the previous step is are in origin frame of final destination but not in odom frame. The approached followed here is to first establish a connection between origin and the final_destination frames.The frame id retrieved from parameter server(final_destination.frame_id) has the value of particular origin among the 4 final origins. A broadcast is done between final_destination and origin frame by broadcasting the final coordinates retrieved from parameter server belonging to origin frame. As, the transform has been established, the robot now needs the final coordinates of the origin with respect to odom frame. To achieve this a listener to odom topic has been created. From the final pose(x,y) received from odom topic, the robot is commanded to reach the final pose using the set_goal method of bot_controller. The robot finally reaches one of the four goal posts around the mentioned origin and stops there,In figure (6), the turtlebot reaches final goal position .

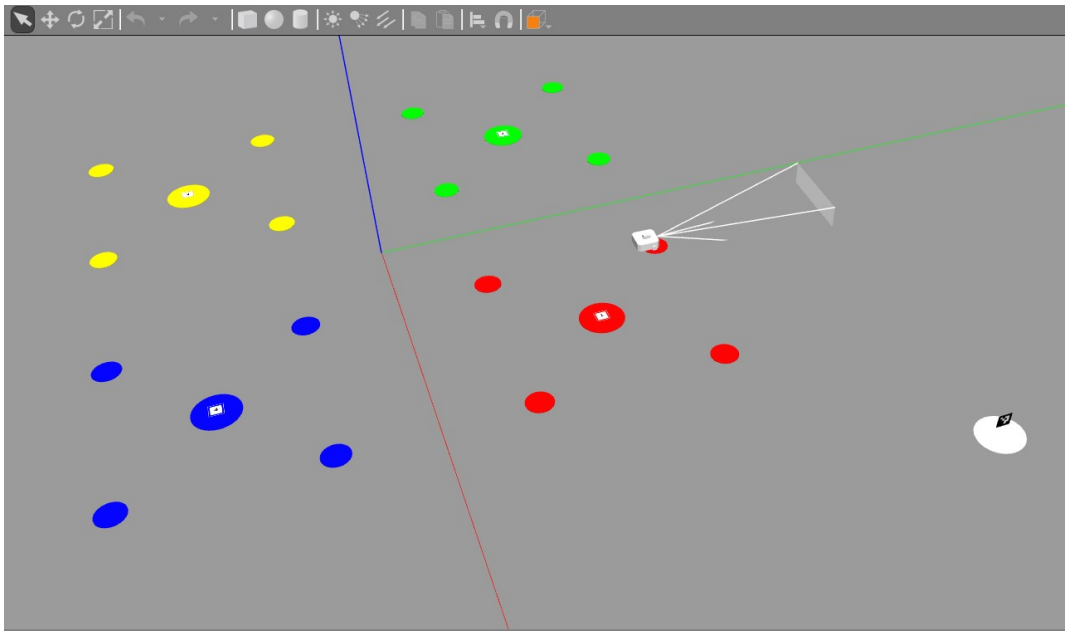

Figure 6: Turtlebot final reaches final goal Case: ArUco Marker 0 origin 1.

The final connected tree of the whole package is shown in figure (7). Note that here the team changed the origin from yaml file to validate the code.

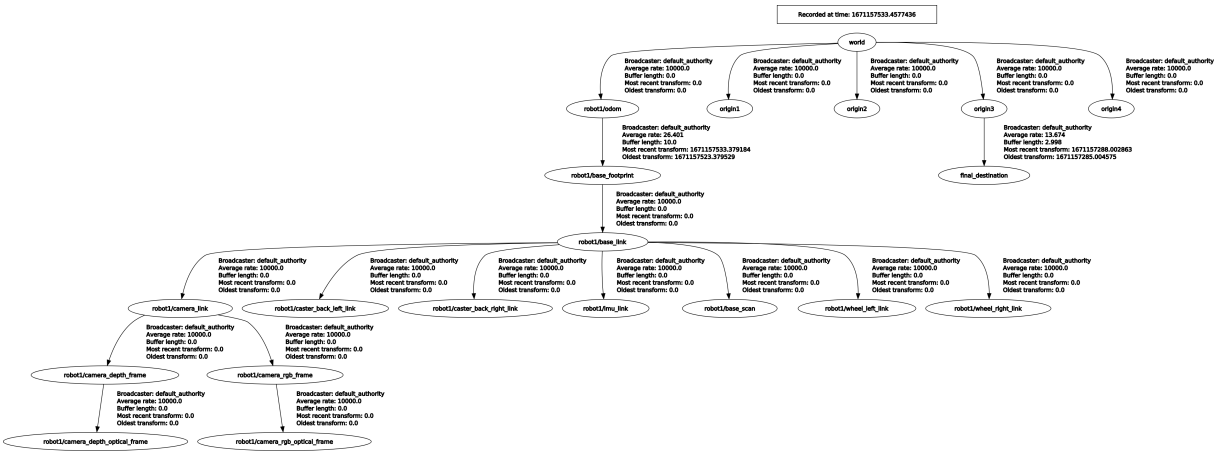

Figure 7: TF tree with origin location set to origin 3 in yaml file

3 Results

As mentioned in the problem statement, the tasks of reaching two goal points by broadcasting transformations between frames is achieved. The robot can also scan different ArUco markers and get the marker information. To check the efficiency and accuracy, the robot has been tested for all 4 origins and different goal posts around each origin. The robot has been tested for all 16 possibilities of the target locations. (4 Origins and 4 ArUcO One of the result of robot reaching the two goals can be referred from figures (5)and (6)

4 Challenges

1. **Odom updater package development:** The team faced hurdles during the development of odom_updater package. Since the examples available on the ROS wiki were generalized for turtlesim, the team had to refer documentation odometry messages to accurately publish the transforms from odom to base footprint.

2. **Integration of retrieved ids from ArUco marker:** The second difficulty faced was regarding the fetching of the goal parameters for Goal 2. The team had developed a method to get the ArUco marker id and concatenate in the form required for fetching the parameters from parameters.yaml file. This however was not correctly executing since the concatenated string retrieved from paramters.yaml file was not declared for the specific instance of the class. This was identified with the help of Prof. Kootbally after which the retrieval of the target coordinates happened properly.

3. **Stop the robot from rotating after reaching the target:** This was observed in the final stage of the project. The team identified that since the goal_reached subscription is bound with the method which checks whether the goal has reached or not, the robot will again keep on rotating after reaching the goal. The team followed two approaches for that:

- The first approach was to declare a private variable for angular velocity which would first be initialized to the value of 0.2 and than set to 0 after the code executes the set goal function for reaching goal 2.
- The second approach was to declare a private flag variable which was later used in a control statement inside the rotation_callback method which continuously monitors the flag variable. This flag variable is set to 0 after execution of the set goal method.

The team decided with the second approach based on the notion that directly assigning an angular velocity is prone to bugs.

5 Project contribution

The task distribution was initially done during the planning phase of the project. This was done based on the core competencies of the individuals and it was made sure that all team members have equal exposure to the projects details. A high level overview of it is mentioned in the individual contributions as shown in (8)

List of Tasks	Darshit	Vamshi	Vinay
Making of a high level design detailing about different tasks to be achieved and algorithm approach to be followed.	✔	✔	✔
Broadcaster creation	✔		
Moving to ArUco target location	✔		
Scanning of ArUco marker by rotating the robot	✔		
Locate the ArUco Marker	✔		✔
Parameters retrieved from the marker scanned			✔
Move to Final goal			✔
Stop the robot rotation after reaching the final location	✔	✔	✔
Report Making and template modification of report		✔	
Creation of flow charts and UML diagrams for report			✔
Gathering figures of the robot reaching two different goals for report	✔	✔	
Doxygen Documentation of both odom_updater and target_reacher packagd	✔		✔
Testing the robot's efficiency by making it reach all target locations	✔	✔	

Figure 8: Table for individual contributions

6 Resources

1. https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Broadcaster-Cpp.html
2. "Lecture Notes - Introduction to Robot Programming" , Professor Zeid Kootbally

7 Course Feedback

- The overall course contents of C++ is explained very nicely and the lecture notes alone are enough to revisit the concepts of C++. If there are lectures or lecture notes given on the data structures part of C++ especially the ones related with competitive coding, it would be really useful too. (like unordered maps, hash maps etc).

- The team felt that the ROS part of the course sufficiently covered the contents required for accomplishing tasks of Final Project. Although some of the topics like launch files needs to be revisited after the project to understand it in detail.