Project Report- Group 4

I. Problem Definition

For a pre-generated world (hospital) without an inital map, a robot (robot M) is expected to map the hospital and guide a visitor (robot S) to announced locations. With the constraint that robot S is only equipped with a RGB camera plus communication with robot M and does not have SLAM capabilities, while robot M is solely responsible for mapping the world and guiding robot S around.

II. Task I

Our first job was to tackle autonomous mapping using SLAM on robot M. To accomplish this we used the '`explore_lite`' package `https://github.com/hrnr/m-explore.git`, which was used in tandem with the inbuilt Gmapping SLAM stack of turtlebot3. The '`explore_lite`' package is a greedy frontier-based exploration method whose job is solely to explore unseen frontiers by providing commands to the '`move_base`' node, it does not generate a cost map by itself, which makes it light weight and easy to setup. For the mapping part OpenSlam's Gmapping package is used (part of turtlebot3's SLAM package) to create occupancy grid from the robot's pose and laser scan data. The two packages work together to autonomously generate a map, a flowchart for the connection between packages is shown in fig 1.

For manually mapping, the Gmapping package was used with the telop launch file to drive the robot around. At the end of the autonomous mapping, not all parts of the world were reached so the robot had to be driven manually to complete the map, the map is shown in fig 2.

The second part of our task consisted of acquiring '`AprilTag`' locations around the world, the same packages as the ones used in the booster 1 assignment was used. Using a listener node, we were able to grab the '`AprilTagArrays`' from the '`/tag_detections`' topic. By using an x-axis transformation matrix, the April Tag coordinates were transformed from the robot's local frame (camera) to the world frame.

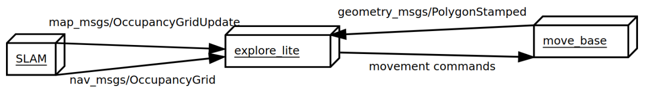

Fig. 1. Flowchart to show how explore lite is used to explore a unknown map.

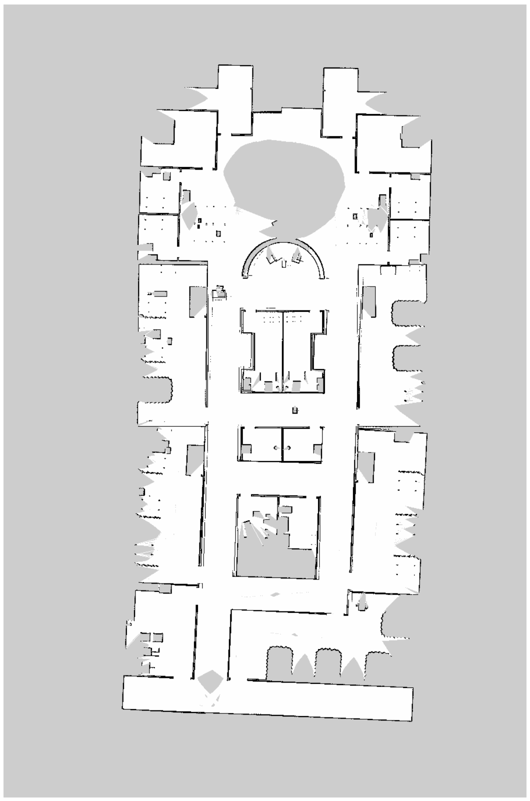

Fig. 2. Generated final map of the hospital.

III. Task II

The second task required us to create a new topic '`/target`' which holds three spots of interest (SOI). Instead of a publisher node, we simply published an array (using `Int16MultiArray`) using the command line '`rostopic pub)`'. Our subscriber node was embedded into a larger project. We utilized Fiorella Sibona's repo on github `https://github.com/FiorellaSibona/turtlebot3_nav` to intake multiple SOIs, split up coordinates and orientations accordingly, and send each set of SOI arrays as a goal to the robot. We subscribe to the '`/target`' topic and were able to acquire 3 separate SOI IDs. With a populated yaml file containing coordinates and orientations attached to SOI ID, we used the '`/target`' topic's published IDs, grabbed the appropriate data from the yaml file, and transferred the data into lists. These lists(used for coordinates/orientations) were also set as global variables so that the global listener/callback and the object could access them. The goal sequence code implements a basic state machine to keep the robot on it's current goal until completed. A flowchart to show the packages called on by the launch file and the interconnection between packages is shown in fig 3.

The flow of command is as follows:

Command line Publisher → '`\target`' → Python Listener → callback function (to populate global lists) → instance of MoveBaseSeq() that implements finite state machine → goal 1 → goal 1 reached → goal 2 → goal 2 reached

→ goal 3 → goal 3 reached → Robot Stops

From fig 3, we can see that the 'move_base_seq' node mainly includes two parts:

1) Global planner: It plans an overall path by reading the given set of targets. The global path of the robot is calculated by the navigation stack. This consists of Dijkstra's optimal path algorithm that implements the minimum cost path on the costmap.
2) Local planner: Having a valid path by avoiding surrounding obstacles, which is designed by 'base_local_planner' package. It will give desired pose and the command velocities to the robot to allow it to move around.

IV. Task III

For this the goal was to guide a visitor robot (robot S) to a target location. However the limitation imposed was that robot S does not have access to the any local or global map, or LiDAR/depth sensors, it is only equipped with a RGB camera and allowed to communicate with robot M through a communication channel '\comm'. This task is mainly an amalgamation of the previous two tasks, with the difference that robot S must be spawned to the hospital world at a known location.

Similar to task II, robot M now has 3 SOI, namely:

1) moving in front of robot S so its AprilTag is visible
2) the target location for the visitor, which robot M needs to guide robot S to.
3) starting location, which robot M must return to

With only the RGB camera equipped robot S can make use of an AprilTag placed on top of Robot M to estimate its proximity, thereby allowing it to closely follow robot M. Robot S then follows M closely as possibly without colliding into M, thereby allowing it to avoid obstacles. The assumption here is that the path generated by robot M's path planner is free of any obstacles. Robot M also tells S when to start following (i.e. when it's in front of S) and when to stop (i.e. when they have arrived at the target location) through the '\comm' topic

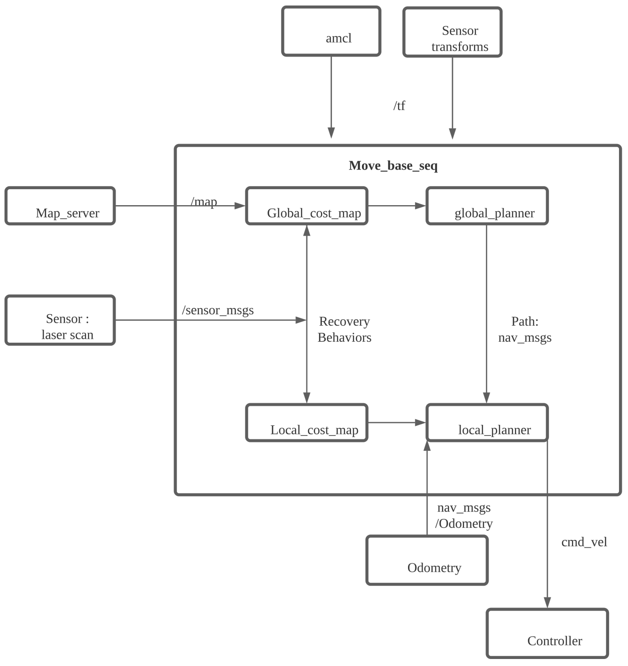

Fig. 3. Flowchart to show the packages used in task II and their interconnection.

V. Task IV

For the creativity task, we decided to work with OpenCV so that robot S could make use of it's RGB camera. OpenCV YOLO model was used along with robot S's RGB camera to implement real time object detection. YOLO is a convolution neural network based model that detects and classifies objects through a pretrained dataset https://github.com/pjreddie/darknet, when objects are identified a bounding box is placed around it with a predicted class name. A screen capture of detected objects are shown in fig 4

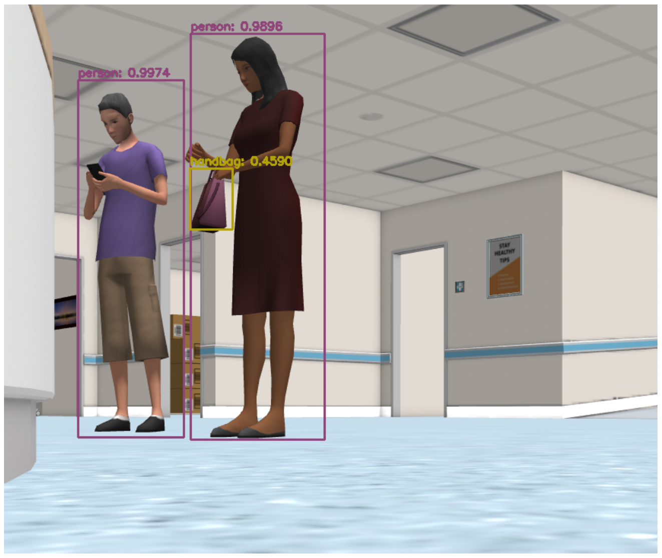

Fig. 4. A screen capture of robot S detecting people as well as a handbag carried by one of them.

Robot S is manually driven around, and we query its RGB camera topic to detect objects in the hospital world in real time. The reason we chose this task was to due to limitation that robot S is imposed with, having object detection capability could allow for further strategic planning for dynamic collision avoidance if objects detected are non-stationary objects such as humans or other vehicles.

VI. Contribution

Task I - Pravin generated the launch files to incorporate explore-lite with the SLAM-gmapping package, Marcos tuned the gmapping parameters and created tag detection to store location of april tags and generated the map.

Task II - Mohammedali and Yuewen worked on creating the /target topic for the robot to listen to, they created the code that allows the robot to navigate through the 3 SOI after they have been published to /target.

Task III - Marcos and Pravin spawned the robots M and S and wrote the code which allows S to track and follow robot M. They also created the /comm topic which allows communication between the robots so S known when to start and stop moving. The tag detection code from task I and the code to allow navigation to different SOIs from task II were incorporated here, therefore Mohammedali and Yuewen both contributed to successful implementation of part III as well.

Task IV - Marcos implemented object detection on his own and implemented it on the Gazebo simulation.

Overall all team members contributed equally in debugging and bugs that arose and implementing the code to each task.