UNSW

Faculty of Engineering
School of Computer Science & Engineering

COMP3431
Robot Software Architectures

Project 1 Individual Report

Group: Bob Ros

Course Coordinator: Claude Sammut
Due Date: 16/09/2020

Dan Nguyen
z5206032

Contents

List of Figures

1 Tasks

The project tasks are summarised and divided like so (following the recommendation of the assignment specifications):

1. Mapping: Explore world, building a map and keep tracking of the trajectory.
2. Exploration: Autonomously navigate world frontiers.
3. Waypoint Navigation: Return to home once all AR markers are located.
4. Vision Processing: Locate and place AR markers on the map.

In order to solve these tasks, ROS packages are to be researched, selected and integrated into the final solution. This is discussed in the next section under relevant task subsections.

2 ROS Packages

2.1 Mapping

Packages explored were:

- GMapping
- Hector Slam
- Cartographer

For the course-provided maze world, package were compared to determine the most appropriate SLAM technique. Each SLAM package was simple to install using APT and easy to run (instructions in Appendix A). A trend existed in increasing map quality with recency of the 2D SLAM technique.

GMapping (Figure 1 below) was released in 2010 and generates the worst map due to its reliance on wheel odometry for localisation. This causes sections of the map to appear to have "drifted" when stitched to the global map, therefore reducing the map accuracy. No further research went into GMapping for its trajectory information as its map result was unsatisfactory.

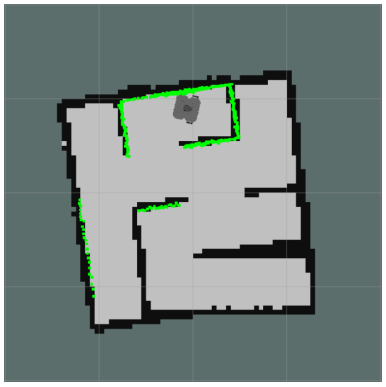

Figure 1: GMapping Map

Hector slam (Figure 2 below) was released in 2012 and generates a good map. This slam technique uses iterative closest point odometry with laser scan data producing more accurate map results. Trajectory required installation of the hector_geotiff package and launching a geotiff_mapper.launch node.

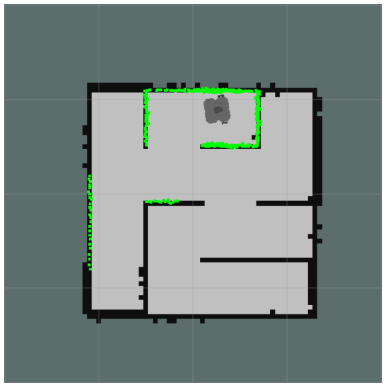

Figure 2: Hector Slam Map

Cartographer (Figure 3 below) was released in 2018 and generates the best quality map. Trajectory is included in the cartographer-ros package and is a display option in RVIZ and requires no additional installation or nodes.

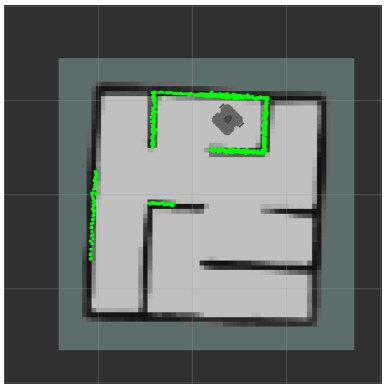

Figure 3: Cartographer Map

Ultimately, cartographer was selected due to its more accurate map result and default display trajectory option in RVIZ to solve the task of mapping. Ease of installation was negligible.

2.2 Exploration

Packages explored were:

- Frontier Exploration
- Explore-lite

From online rumours, frontier exploration was no longer supported for the Melodic distribution (this is confirmed by no available APT package). However, the frontier exploration project on GitHub had a melodic-devel branch which prompted personal interest in building the package. Building

and running frontier exploration showed problems related to its server-client feature i.e. frontier exploration was not communicating with any active topics (see Figure 4).

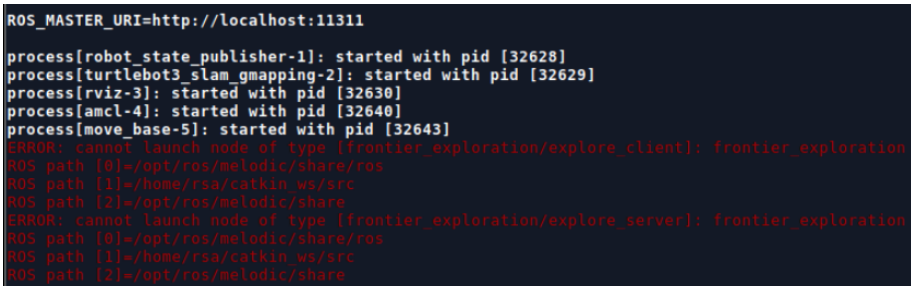

Figure 4: Frontier Exploration Error

Explore-lite is a derivative of frontier exploration designed as a light-weight SLAM. Running explore-lite revealed problems with the course-provided maze.launch world where the simulated turtlebot3 had difficulty in searching for a starting position (see Figure 5). This problem did not exist for large worlds, therefore it can be deduced that the turtlebot3 and wall boundaries were overlapped. As explore-lite did not show promise in simulation, we decided to use wallFollower.cpp to explore the map as the more reliable and quick-to-implement solution.

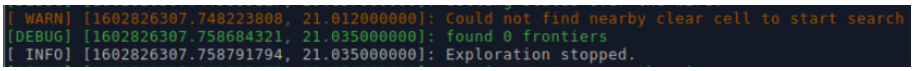

Figure 5: Explore-lite Warning

The provided wall following algorithm had a major problem of alternating between states at intersections (which causes the turtlebot3 to appear to have stopped or pendulate). This prompted research into existing wall following algorithms and attempts to improve the given algorithm (see Appendix C). A good wall following algorithm would have more subdivisions of its point clouds and many switch cases to select a heading, which was considered too verbose for this project. Therefore, the original wall follower code was used which had satisfactory performance to autonomously navigate the world.

2.3 Waypoint Navigation

No ROS packages were explored for navigation as wall following was decided as the exploration method. To navigate home, a listener was written in wallFollower.cpp to stop at home if the turtlebot3 was within the home boundary (see Appendix D). The initial implementation involved writing a submap listener which was published by cartographer. This worked well in simulation but caused conflicts in the transform tree on the turtlebot3 in the lab. The adhoc solution was to rewrite the listener to listen to the odometry topic. This was highly unreliable as pose.position had large positional drift exacerbated by turtlebot3 turns.

2.4 Vision Processing

Packages explored were:

- Aruco & OpenCV
- AR Track Alvar & ROS

Aruco and OpenCV was considered for its more powerful functionality for manually mapping laser scans with AR marker positioning therefore leading to more accurate localisation. This option worked but was difficult to integrate with the existing solution as well as its incompatibility with the marker dictionaries.

The package solution for the vision processing task was AR tracker alvar which has built in dictionaries to interface with the AR markers and was easily integrable into the existing solution. The AR marker solutions are mapped onto RVIZ as shown in Figure 6.

Figure 6: AR Mapping

3 Evaluation & Limitations

Bob Ros scored 9/10. The turtlebot3's performance was very good in traversing the map without collision, and spotting and marking all AR markers on the map.

0.5 marks was lost for a placement error in AR marker 4. The cause for this was due to the approach of the turtlebot3 to the marker at an extreme angle which caused the marker's misplacement. The marker placement error was minimised by reducing the distance the turtlebot3 was allowed to approach the marker.

Another 0.5 marks was lost due to the turtlebot3 failing to stop at home as it waited for the odometry position to fall within the home boundary. This was due to odometry's huge positional drift which may cause the odometry positional origin (home boundary) to drift outside the map boundaries. It should be noted that this drift was exacerbated with every turn.

Limitations on the current robotic software architecture are:

- Very large turn radius which may cause the turtlebot3 to run into walls or get lost.
- Distance and angle between turtlebot3 and marker which can cause marker misplacement.
- Left wall following exploration which inherently has an inability to explore certain map structures and possibility of missing AR markers on walls that are never faced by the turtlebot3.

4 Contribution

From the work outlined above in section 2: ROS Packages, my contribution covered mapping, exploration, and waypoint navigation:

- Researched SLAM packages, identified installation instructions, and selecting an appropriate SLAM.
- Investigate exploration packages, and proposing wallFollower.cpp as an alternative solution.
- Attempt to rewrite wallFollower.cpp algorithms, refined launch file parameters.
- Attempt to implement a "stop at home" feature in wallFollower.cpp.

Appendix A SLAM Instructions

The following instructions to install and run SLAM packages assumes ROS Melodic distribution.

GMapping

```
> roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

Hector Slam

```
> sudo apt-get install ros-melodic-hector-slam
> roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=hector
```

Cartographer:

```
> sudo apt-get install ros-melodic-cartographer ros-melodic-cartographer-ros
  ros-melodic-cartographer-ros-msgs ros-melodic-cartographer-rviz
> roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=cartographer
```

Running cartographer in the simulation requires editing the following file:

`~/catkin_ws/src/turtlebot3/turtlebot3_slam/config/turtlebot3_lds_2d.lua`

and changing

`tracking_frame = "imu_link",`

to

`tracking_frame = "base_footprint",`

Appendix B Exploration Instructions

The following instructions to install and run exploration packages assumes ROS Melodic distribution.

Frontier Exploration

```
> cd ~/catkin_ws/src
> git clone https://github.com/paulbovbel/frontier_exploration.git
> cd ~/catkin_ws
> catkin_make
```

```
> source ~/devel/setup.bash
> roslaunch turtlebot3_slam turtlebot3_frontier_exploration.launch
```

Explore-Lite

```
> cd ~/catkin_ws/src
> git clone https://github.com/hrnr/m-explore.git
> cd ~/catkin_ws
> catkin_make
> source ~/devel/setup.bash
> roslaunch explore_lite explore.launch
```

Appendix C Wall Follower Algorithm

Pseudo-code of an attempted wall follower algorithm.

```
Get the closest point on our right
Get the closest point on our left
Get the closest point in front

If left is free
    turn left
else if left is occupied
    turn right
else if left and straight and right are occupied
    turn 180
else
    go straight
```

Appendix D Homing Algorithm

Pseudo-code for the turtlebot3 to stop at home when at home.

```
If time elapsed is greater than 1 minute
    If current x position is within home x range
    And if current y position is within home y range
        Publish 0 velocity
```