RMIT University

Final Report

In fulfilment of OENG1168 Engineering Capstone Project 4B

Autonomous Robotic Mapping and Navigation

Authors:
Chris Benhard Armanda (s3626565)
Housam Barakat (s3401160)
Michael Veloso (s3485068)

Supervisor:
Dr. Akram Hourani

Abstract

Autonomous navigation and mapping is starting to be more well known as it can be used in a variety of applications. From being deployed to search and rescue missions, exploration, disaster zones and picking and storage as well. In this report, the above mentioned applications will be presented and the related software to allow the robots to possess such skills. The algorithms Hector_SLAM and Gmapping will be compared, in terms of localization and mapping capabilities in an urban environment. All software used has been used internationally, for many purposes. As of the current date, Autonomous Mapping and Navigation has been researched further for the main purpose of search and rescue.

Contents

List of Figures

Chapter 1

Introduction

The concept of Autonomous Mapping and Navigation is becoming increasingly popular in the recent years, thanks to smart cars manufacturers, the most well known being Tesla. The concept however is no longer exclusively useful only to build smart cars, but also have been introduced to robots to do seemingly menial tasks to increase work efficiency and at the same time to decrease costs and manpower. Amazon, the online retail giant, is known to have several robots equipped with autonomous mapping and navigation in their warehouses to keep accurate inventory without the need of a employing large number of warehouse workers.

Although robot with self-localization and mapping (SLAM) offers many benefits, they are usually expensive for personal use. Such robots are also usually fit for single purpose. This project aims to offer a cheaper option of SLAM robots that is both fully configurable and open-sourced. TurtleBot3 has been decided to be procured for the purpose of this project, since its documentation is comprehensive and supported by the growing community, helping to build variations of the robot to execute more than one purpose. The robot was preloaded and programmed with Robot Operating System (ROS), a pseudo-operating system residing side-by-side with a functioning Linux Ubuntu system.

Chapter 2

Statement of Problem

By aiming for fully autonomous mapping and navigation there are multiple applications that we can reach out to. Most notable fields are:

- **Exploration**

  Which can now be done autonomously, by sending in a configured robot this will allow it to fully traverse the terrain without being hindered by any obstacle while simultaneously mapping the surroundings and transmitting the data to a remote laptop. One of the huge advantages of using autonomous robots in such a dangerous field, is the lowering of the mortality rate. These robots can be deployed in an urban search and rescue mission, deployed in disaster zones and other such related areas. Robots are expendable, can be fully configured and prepared for most scenarios, making it much more easier and safer. Humans on the other hand are not expendable, it takes time to train, money to prepare and procedures that they need to remember for every situation. Even when a fully trained human is been given the proper gear and instructions which will leave them ready to face most scenarios, there will always still be a mortality rate present due to the nature of their profession. To effectively lower the mortality rate in this particular field, autonomous robotics would be the ideal tool.

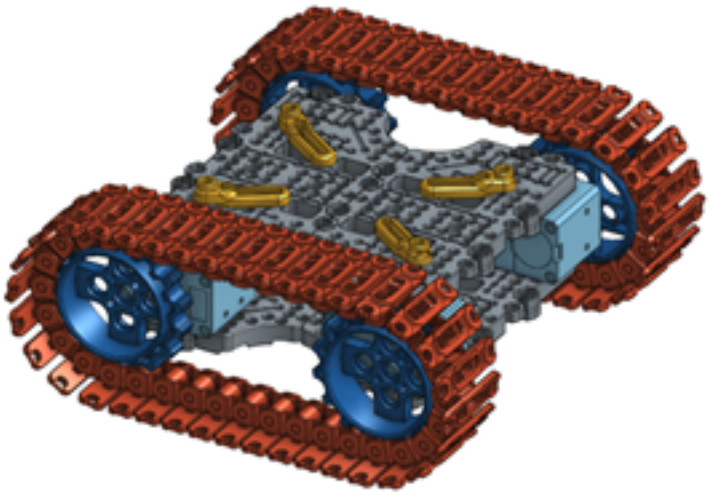


Figure 2.1: Unique all-terrain robotic solution

- **Picking and Storage**

  Automated picking of components and storing them in predetermined locations is another area we can branch out to as well. Utilising a robotic arm with sensors allows the robot

to detect the type of component,size and then map a path towards the storage location. Having a automated storage also ensures that the system being used can keep track of every delivery therefore reducing any downtime. Automating a picking and storage system can create big leaps in terms of efficiency and lean manufacturing.

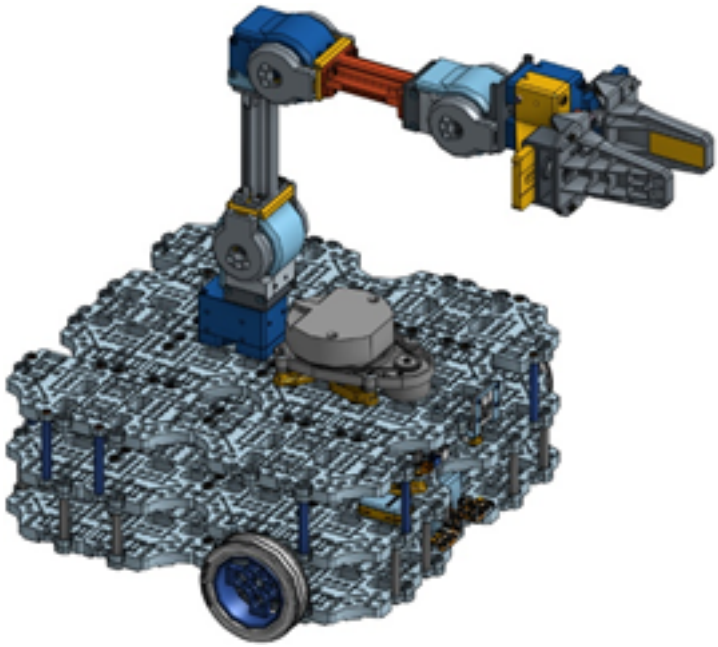


Figure 2.2: Unique picking and storing solution

- **Emergency Relief**

  Fitting the TurtleBot3 with a fire resistant exterior, will allow the robot to go into buildings with low structural integrity or any environments deemed to dangerous for humans and map the location. Fully configured with the proper hardware will allow the robot to locate sources of extreme heat and be able to tell the different heat temperatures so civilians in distress can be identified. By having the assistance of a autonomous robot Firefighters will be able to locate civilians much faster and more efficiently, theoretically removing the amount of time spent in a dangerous environment and the amount of casualties usually associated with the job. (current work in progress)

Chapter 3

Literature Review

SLAM Robots in the past have been involved in high-profile operations, including the 9/11 disaster (1). Such robots have proved beneficial during SAR operations, as disaster areas may still be lethally dangerous for human due to collapsing building structure or poisonous air, and psychologically demanding. Hence robots are seen as an alternative method to reach such areas thought unsafe for professional SAR teams to do so (2). It is generally accepted that a certain degree of self autonomy should be assigned to robots in scenarios where time and speed are crucial. However, there is no consensus whether a robot should be given full autonomy or should it be supervised by a human controller who is able to override the robot's decisions. although Robot Operating System (ROS) has provided methods for both options, deciding which to choose is still considered a gap in the literature (3).

In building fire scenarios, it is expected that a robot will have to traverse among debris. Algorithms called Parallel Tracking and Mapping (PTAM) and Inertial Measurement Unit (IMU) have been successfully employed in a prototype robot, allowing it to control its movement's inclination to move on an unknown rough terrain (4). Another challenge the robot will face is reduced visibility due to smoke and fog from the fire. A breakthrough to overcome this problem has been done, by a combination of a long range finder (LRF) sensor and utilising one of ROS's available SLAM algorithm called gmapping (5).

In order to assist the firefighters to identify the fire's hotspots and identifying trapped victims, it is imperative that the robot has a camera, and transmitting the image wirelessly into a computer through an ad-hoc peer-to-peer connection. An encrypted transmission of images have been previously done for traffic surveillance purposes (6). Images recorded and transmitted while in a SAR operation should be very secure and can only be viewed by the firefighters due to the sensitive nature of the images which would be taken.

In situations where it is deemed unsafe for firefighters to reach trapped victims, a robot can help evacuating them. A study by Boukas et al. have found that robot with SLAM capabilities can find a path to guide people to a building's exit (7). However, Boukas' study concerns about guiding a large crowd to alternative exit to prevent stampeding at one exit, which would be more useful in settings such as stadiums. There are no emergency situation introduced to Boukas' proposed robot, where time is indispensable.

Victim identification can be further assisted by the use of Wireless Sensor Network (WSN). It has been proposed to attach a Passive Infrared (PIR) sensors to a robot, so as to detect human movement (8). WSNs can also be utilised to find the primary hotspots of the fire by locating any source of gas leaks. An algorithm for self-navigating robot to rely on WSN to find its own way has been developed. The study by Siyao Fu et al. examined a robot's behaviour

when navigating on its own by WSN, however their study involves WSN being setup on the building's wall (9). Instead of relying on externally setup WSN, the proposed robot should have its own WSN to locate the sourceof fire. A breakthrough has been made to equip robots with microbolometer (an infrared-emitting sensor to detect heat) to detect the presence of carbon dioxide (CO_2) (10).

Another problem with SLAM robots are how the maps produced. Currently, there have been no definite solution to decide whether to store collected data permanently or discard them (11). This issue is important to be considered, as the surroundings of a robot can change drastically over short period of time in a fire setting. The type of map representation to be produced should also be decided. There are six common map representations, namely:

1. Grid
2. Feature
3. Topological
4. Semantic
5. Appearance
6. Hybrid maps (12).

For the robot to produce a quality map, an efficient odometry is required. Other mapping techniques such as utilising Global Positioning System (GPS) cannot be relied on areas where GPS signal may be denied (13). Gmapping, as mentioned above, as one of the SLAM techniques provided in ROS, utilises odometry information to update maps; however, odometry should not be seen as authoritative solution, as data gathered odometry can be erroneous due to lack of precision, robot slip, and friction (14).

Odometry errors can be corrected, as a study done by Czarnetzki et al. has shown, by using an optical sensor (15). Another workaround to avoid odometry errors entirely is to use a different ROS mapping technique instead. Hector_SLAM can be proposed, as it relies on scan matching alone without odometry (16).

Before a robot can self localize it needs to be equipped with sensors that are read by the algorithm used and determines the robot position at the time. For our project our autonomous navigation robot is equipped with a LiDAR (Light Detection and Ranging). A LiDAR is a remote sensing method that uses light in a the form of a pulsed laser to measure ranges to the earth. A LiDAR system is capable but not restricted to, geological mapping/imaging to monitor erosion or other changes, monitoring growth of plants and trees, doing surveying work for construction projects, making accurate volumetric estimates of landfills.

LiDAR is based on measuring the time it takes for a laser pulse to return to the sensor, highly reflective surfaces pose issues. Most materials have rough surfaces on a microscopic level, and scatter light in all directions. If a surface is very reflective the light is reflected coherently away from the sensor, and the mapping can be incomplete for that area. Heavy fog and rain and other environmental effects can pose issues for the LiDAR system, this issue can be alleviated by using higher power lasers. LiDAR systems also have a relatively slow refresh rate of the snipping LiDAR 10hz (10 times per second) is approximately the fastest that the LiDAR system can rotate, but it takes more than a LiDAR to get a robot to fully work autonomously, there are five main checkpoints the robot must tick to achieve full autonomy.

1. **Self-Localization**

2. **Navigation** the key to having a mobile robot that it is able to be navigated so micro-controllers and motors to dive the wheels are essential in all robots.

3. **Autonomy** the desired result of all robots that are mobile is to be able to navigate autonomously without the help of a pilot having minimum to no human interactions

4. **Sensors**, playing a very important role in making any robot, is to be able to navigate autonomously while also avoiding any obstacles in the way, that is achieved with different kinds of sensors but the most frequently used sensor is the ultrasonic sensor. An ultrasonic sensor is a device that can measure the distance to an object by using sound waves. Other researchers used a laser range finder to get the required results

5. **Algorithms** for robotic mapping are essential to obtain the most accurate map possible.

Chapter 4

Methodology

4.1 TurtleBot3 Assembly

Once all of the necessary components arrived from Robotis, there were certain tasks that needed to be completed before we could begin.The first task to do was to inspect the components visually to check for any defects and functionally testing to ensure that all components were working as intended. This process took roughly ten minutes to complete, which then allowed for the next stage: assembly. Assembling the Turtlebot3 was time consuming due to the nature of the components presented and the human interaction required for the part to be mounted correctly.

	Part name	Number of Components used
Chassis parts	Waffle Plates	8
	Plate Support M3 (35mm)	4
	Plate Support M3 (45mm)	10
	Wheel	2
	Tire	2
	Ball caster	1
Motors	Dynamixel (XL-430-W250-T)	2
Boards	OpenCR 1.0	1
	Raspberry Pi 3 Model B+	1
	USB2LDS	1
Sensors	LDS (HLS-LFCD2)	1
Memory	MicroSD Card	1
	Raspberry Pi Camera	1
Power	Li-Po Battery Extension Cable	2
	USB Cable	1
	SMPS (12V 5A)	1
	A/C Cord	1
	Li-Po Battery (11.1V 1800mAh)	1
	Li-Po Battery Charger	1
Tools	Philips Screwdriver	1
	Rivet Tool	1
Miscellaneous	PH_M2x4mm_K	8
	PH_T2x6mm_K	4
	PH_M2.5x8mm_K	16
	PH_T2.6x12mm_K	16
	PH_M3x8mm_K	4
	NUT_M2.5	20
	NUT_M3	16
	Rivet_1	14
	Rivet_2	2
	Spacer	4
	Bracket	5
	Adapter Plate	1
	Raspberry Pi Camera Ribbon Wire	1

Table 4.1: Components used to assemble TurtleBot3

The base level (known as level 1) housed the two DYNAMIXEL motors, two sprockets for the wheels and the Li-Po Battery 11.1V, 1800mAh. In this level there were no real challenges presented, manually aligning the base of the DYNAMIXEL with the waffle so the rivets can hold it in place was a minor issue that was resolved quite easily.

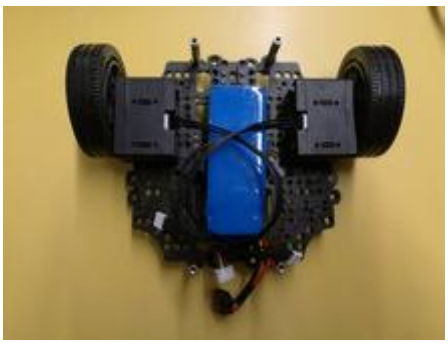


Figure 4.1: Level 1 assembly and arrangement

The level 2 waffle was placed directly on top of the motors and was supported by four rods and four screws. Once the waffle was secure, the OpenCR (32-bit ARM ® Cortex ® - m7) circuit board needed to be attached via four PCB support connectors. Wiring the OpenCR so that it can control both DYNAMIXEL motors as well as receive power presented a minor issue, the cables were in tight tension when both ends are in use. After manually adjusting and bending the cable to reduce tension on specific parts of it, we were able to properly establish a connection between the level 1 components and the OpenCR on level 2.

Figure 4.2: Level 2 assembly and arrangement

The third level was for the Raspberry Pi and USB2LDS boards, the Raspberry Pi will be used for wireless connectivity and data transference. The USB2LDS will connect to the LiDAR and allow it to transmit the raw data to make a Top down view of its surroundings. A similar issue was faced in terms of tension in the cables, the cables provided by ROBOTIS were short and needed to be manually bent in order to relieve a small fraction of the tension they are under. The USB2LDS cable is relatively thin and requires more attention when bending it, due to the established design of the TurtleBot3 there is not much that we can do about it, without a new design of the cable. We also implemented another camera on this level to allow manual control. In the scenario that you are on the master remote PC at the office and you need to send in a robot for manual surveillance and scouting of a building that is on fire, you will them have the ability to see what the robot does, visually inspect the surroundings and point out key interests.

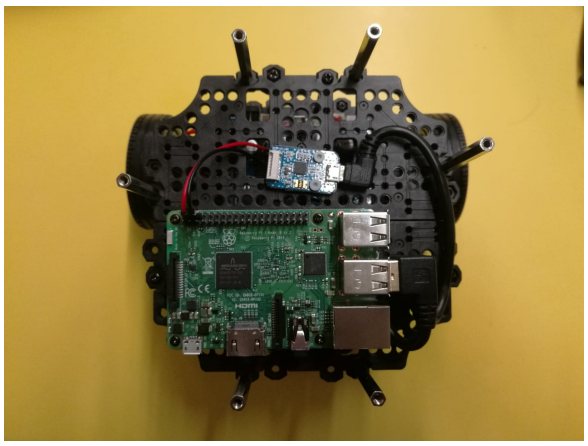


Figure 4.3: Level 3 assembly and arrangement

The fourth and final level (upper most level) of the Turtlebot3 has the 360° LiDAR for SLAM and Navigation. By having the LiDAR on the upper most level helps ensure that it will be able to see everything without any obstructions to its vision from its body. The LiDAR is connected via four PCB support connectors This method also brings forth an issue, being on the highest level means that it can not detect anything below itself as it establishes itself as the base level and records what it sees. To resolve this issue an additional camera was added to the third level, which will allow the TurtleBot3 to detect any obstacles that might be in front of it and below the LiDAR. The LiDAR was the final component of the assembly, the overall process was quite linear and did not consume a lot of time. Secondary testing was also done on the components as well, we wanted two testing phases to ensure that the components weren't damaged after assembly as well.

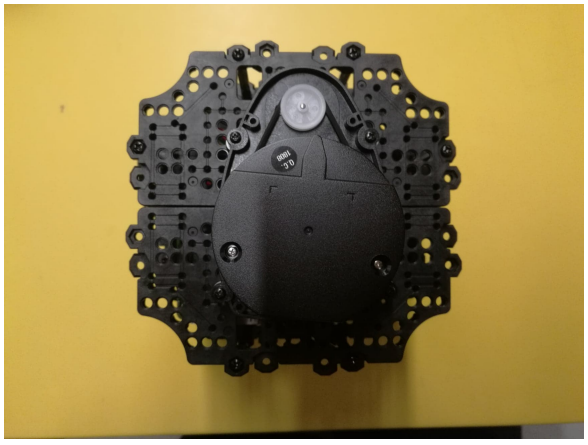


Figure 4.4: Level 4 assembly and arrangement

4.2 Robot Operating System (ROS)

ROS is an open-sourced framework to write software for robotics. ROS includes tools, libraries and packages to program a robot and analyse its behaviour. In this project, ROS acts as intermediary between a user's personal computer and the TurtleBot3's Raspberry Pi as the latter's computing unit. ROS can only be installed in Linux Operating System; hence the primary reason why the user's PC should be a Linux machine. Since Raspberry Pi's default operating system is based on Linux Ubuntu, ROS is an excellent framework to use for TurtleBot3.

ROS provides numerous stacks necessary to program the robot. The stacks include files to run TurtleBot3's sensors, servo motors, mapping algorithms, and simulations. Since ROS is community-maintained open-source framework, researchers and engineers may compile their own stacks to be used by other TurtleBot3 users. For example, one of the mapping algorithm being used in this project, Hector_SLAM, is developed by a group of engineers from the Technical University of Darmstadt, Germany. ROS can be installed.

In this project, the latest stable release of ROS, codenamed Kinetic Kame was used, installed onto a PC with a Linux Ubuntu MATE 18.2 operating system.

4.3 Installation and Configuration on Linux Machine

To install ROS on a typical Ubuntu machine, the command

```
$ sudo apt-get install ros-kinetic-desktop-full
```

is to be issued to download and install the complete ROS stack. After ROS is installed, the command

```
$ source /opt/ros/kinetic/setup.bash
```

should be issued in the terminal to run the script file unique to the Kinetic distribution of ROS. Next, the command

```
$ mkdir -p ~/catkin_ws/src
```

is issued. The catkin_ws directory will be where all ROS stacks, applications, and files will be located.

As ROS is compatible with any robots that are programmable with Linux, it is important to configure the installed ROS to the files distributed by TurtleBot3's manufacturer, Robotis. To do so, Robotis' code repository from GitHub is to be cloned onto the catkin_ws/src directory made earlier. The commands

```
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
```

and

```
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
```

are issued. To install the two repositories onto the ROS in the machine, the command

```
$ cd ~/catkin_ws && catkin_make
```

is issued. The command catkin_make compiles the code repositories and ensures their hierarchy and libraries are maintained in the machine as they are in the GitHub.

The Linux machine will control the TurtleBot3 remotely through Wi-Fi. As such, the ROS should know the IP address of the machine. The setup shell file bashrc should be opened, and the following two lines should be added at the end of the file.

```
export ROS_MASTER_URI= http://192.168.10.100:11311
export ROS_HOSTNAME= 192.168.10.100
```

The computer was assigned the IP address 192.168.10.100, hence the command. Whenever the IP address's lease expires, or the TurtleBot3 is operated in a different network, the bashrc shell file should be modified to follow the correct IP address.

4.4 Installation and Configuration on TurtleBot3

Raspberry Pi 3 is the main computing unit of the TurtleBot3. For this project, the Raspberry Pi's microSD card is preloaded with a burned image of Ubuntu Mate, using a normal PC and software such as Etcher. The Raspberry Pi should also be connected to the same Wi-Fi network with the Linux machine from the earlier section. With the Pi connected to the Internet, the same commands to install ROS should be issued from the Pi's terminal window, with a few changes.

After ROS is installed onto the Raspberry Pi, a driver file for the TurtleBot3 LIDAR should be downloaded by issuing the command

```
$ git clone https://github.com/ROBOTIS-GIT/hls_lfcd_lds_driver.git
```

on the catkin_ws directory. Another change is that the Pi does not need ROS packages such as SLAM, teleoperation, and simulation, as all of these packages would be run on the Linux Machine. To remove them, the commands

```
$ cd ~/catkin_ws/src/turtlebot3
$ sudo rm -r turtlebot3_description/ turtlebot3_teleop/ turtlebot3_navigation/
turtlebot3_slam/ turtlebot3_example/
```

is issued.

Similarly, the IP address of the Pi should also be added into its bashrc shell file. During the testing, the Pi was assigned the IP address 192.168.10.101. As such, the bashrc's last two lines should be as follows.

```
export ROS_MASTER_URI=http://192.168.10.100:11311
export ROS_HOSTNAME=192.168.10.101
```

It is important to note that only the ROS_HOSTNAME reflects the IP address of the Pi, but not the ROS_MASTER_URI. This is because when ROS is activated at both the Linux computer and the Pi, the former will act as a Master computer, whereas the Pi will be the Slave. All commands to operate the TurtleBot3 will normally be issued from the Master computer.

4.5 Raspberry Pi Camera

Camera is not included in the TurtleBot3 package. However, a camera would be useful as a modification to the robot. Since the robot is meant to be operated remotely, the user can navigate the robot safely. To do so, a Raspberry PI Camera v2 is installed as an external peripheral on the Raspberry Pi's camera port.

By default, Raspberry Pi does not allow a camera to take photography or stream videos, although the camera is already physically connected. To reconfigure the Pi, the command

```
$ sudo raspi-config
```

is issued in the Pi's terminal window. The camera's can then be modified as an external input device for the Raspberry Pi. The Pi then must be restarted to complete the process. Upon rebooting, the Pi should then be able to take pictures. To test whether the camera functions, the command

```
$ raspistill -v -o test.jpg
```

is issued, to take a photograph and save the picture in the PI's home directory as "test.jpg".

Should there be no problems in testing the camera, the Raspberry Pi is ready to livestream a video into the remote Master's browser. A Python script is needed, which is available in the appendix of this report. The Python script file is then be saved as rpi_camera_surveillance_system.py. The script will feed the video to the Pi's TCP port 8000. As such, to view the camera's live video, the user controlling with the Master can open an Internet browser and type the address 192.168.10.101:8000. A sample of the camera's live stream on the Master's browser is shown in the Figure 4.5 below.

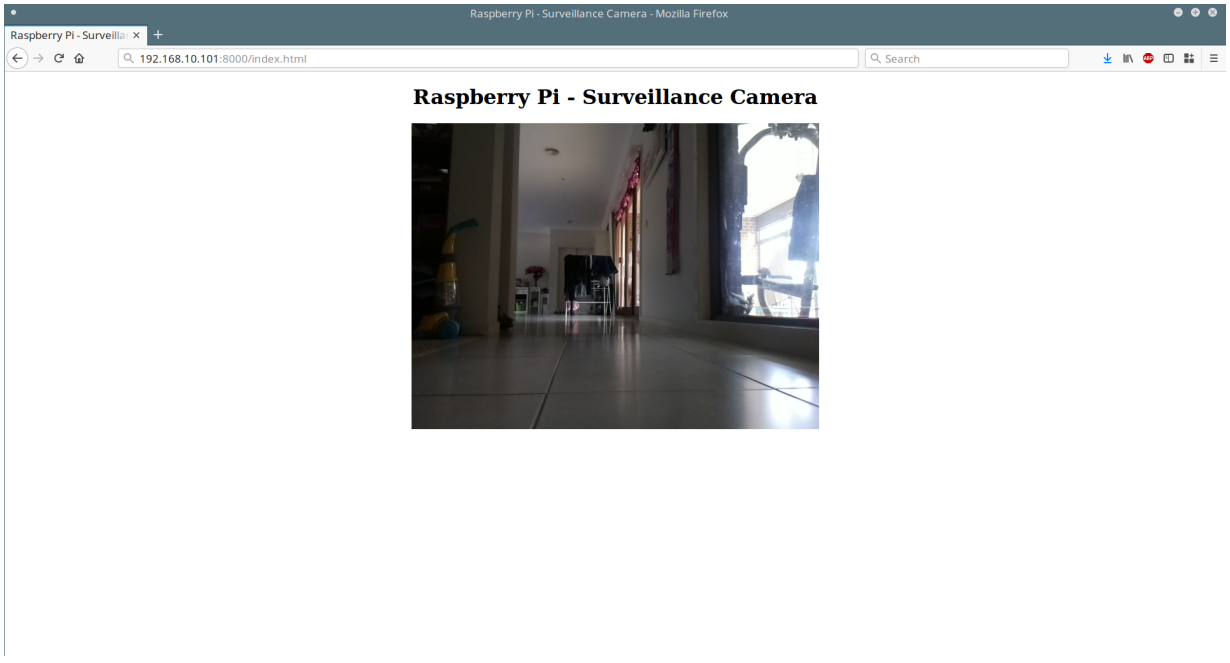


Figure 4.5: Live video stream from the Raspberry Pi Camera

4.6 RViz

ROS Visualization is a 3D visualizer for displaying sensor data and state information from ROS. RViz can also display live representations of sensor values coming over ROS Topics including

Camera, depth cloud, effort, fluid pressure, grid, grid cells, illuminance, image, interactive makers, laser scan, map, maker, maker array, odometry, path, point cloud, pose, range, robot model, tf, temperature.

All these qualities that RViz provide to the user makes it much easier to see and understand what the robot is seeing and thinking in the world around it hence making debugging much more easier to the user. The combination of ROS and RViz makes using simultaneous navigation and mapping (SLAM) easier to achieve.

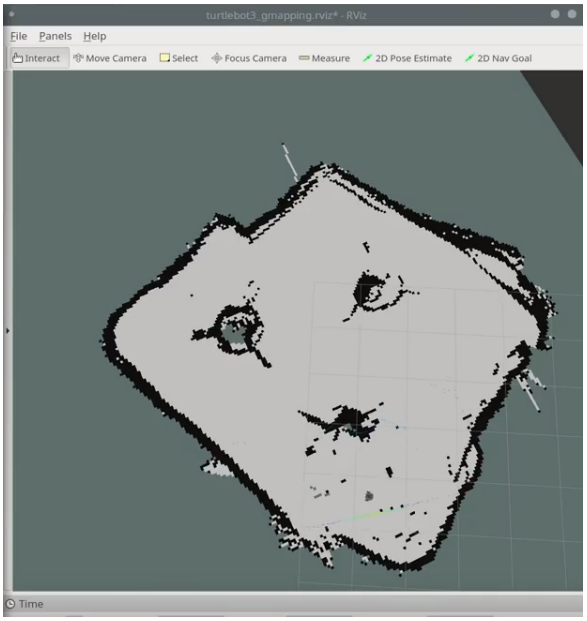


Figure 4.6: Rendering of map with RViz

The Figure 4.6 above shows the complete mapping of building 55 level 02 room 04 that is produced in RViz in this is why RViz is a very handy application to use since it provides the user with information of what the robot is seeing and what type of map is being produced while it moves around the room.

Chapter 5

Discussion of Results and Findings

The TurtleBot3 was tested mainly in RMIT's Manufacturing Hub. The laboratory room was chosen due to its relatively small area but filled with chairs and desks, therefore suitable to examine the robot's ability to avoid obstacles and the mapping algorithms' accuracy.

5.1 Operating TurtleBot3

In the Master computer's terminal window, the command

```
$ roscore
```

is issued. This will activate the ROS package within the Master. At the same time, a Secure Shell (SSH) session is established to access the Slave's terminal. In the Slave terminal, the ROS is similarly activated in the TurtleBot3, with the command

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

The two terminal windows will be left as it is during operating the TurtleBot3.

To control TurtleBot3's movement, a new terminal window is opened in the Master. The command to be launched is

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

This command allows the TurtleBot3 to be controlled using the Master's keyboard. In the keyboard, pressing 'W' will drive the robot forward and increment its speed by approximately 10 cm/s. Pressing 'X' reverses the direction by 10 cm/s. Pressing 'A' or 'D' will turn the robot's heading towards the left or right respectively by 0.01 rad/s.

5.2 Drawing Maps

While keeping all previous terminals open, another terminal should be opened to launch the SLAM algorithm. The command to be issued in the new Master's terminal is

```
$ roslaunch turtlebot3_slam turtlebot3_slam.launch.
```

The terminal will display a stream of the robot's pose, and starts an RViz instance showing the robot and the obstacles around it. To build the map to a larger area, the teleop_key.launch

terminal used to control the robot's movement is reopened. RViz will update the map accordingly.

By default, the turtlebot3_slam will build a map with gmapping algorithm. To draw a map with other algorithms, for example with Hector_SLAM, the command to be issued should be

```
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=hector
```

5.3 Comparison of SLAM Methods

Gmapping algorithm assumes that the robot's initial pose in an imaginary Cartesian plane to be located on the origin, that is (0, 0, 0). The first 2 coordinates are the (x,y) coordinates, with the angle where the robot is heading is represented as the third number. The gmapping algorithm then will determine the robot's nearby obstacles such as wall and furnitures, and draw them accordingly based on the information provided by the scanning of the Lidar. As such, the gmapping's iteration of the robot's pose can be illustrated in the table below.

X_1	X_2	X_3	...	...	X_n
Y_1	Y_2	Y_3	...	...	Y_n
θ_1	θ_2	θ_3	...	...	θ_n

Table 5.1: Poses information streamed by gmapping

The second method tested in this project, Hector_SLAM, does not take into account the Lidar's scanning to determine the robot's pose. Instead, Hector_SLAM reads into the robot's odometry. To do so, the algorithm reads into the Dynamixel's encoding to determine the robot's position. In the scope of this project, the robot is mainly tested within RMIT University's Mechatronics Lab, Building 55 Level 2 Room 4. The actual map of the room during the testing was sketched with a computer aided design software, as shown by the Figure 5.1 below.

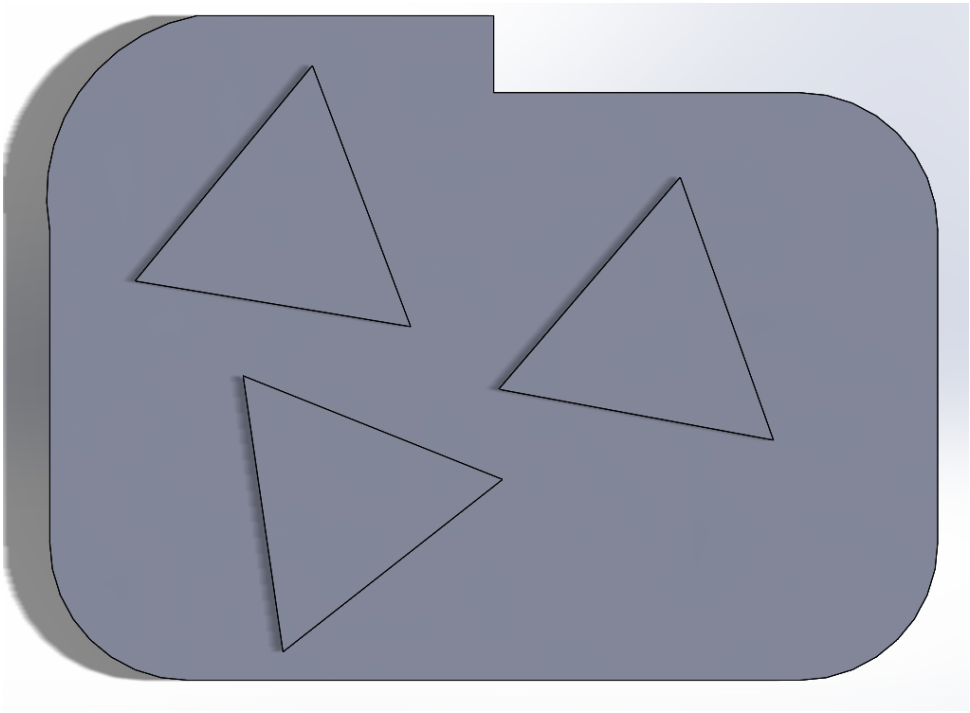


Figure 5.1: Sketched map of the testing laboratory for comparison

The room consists of 3 triangular desks supported by circular bases. Since TurtleBot3's Lidar only captures obstacles at the same height where the Lidar is positioned on the robot, only the desks' bases were captured in RViz. The map rendered with gmapping algorithm is shown in the Figure 5.2 below.

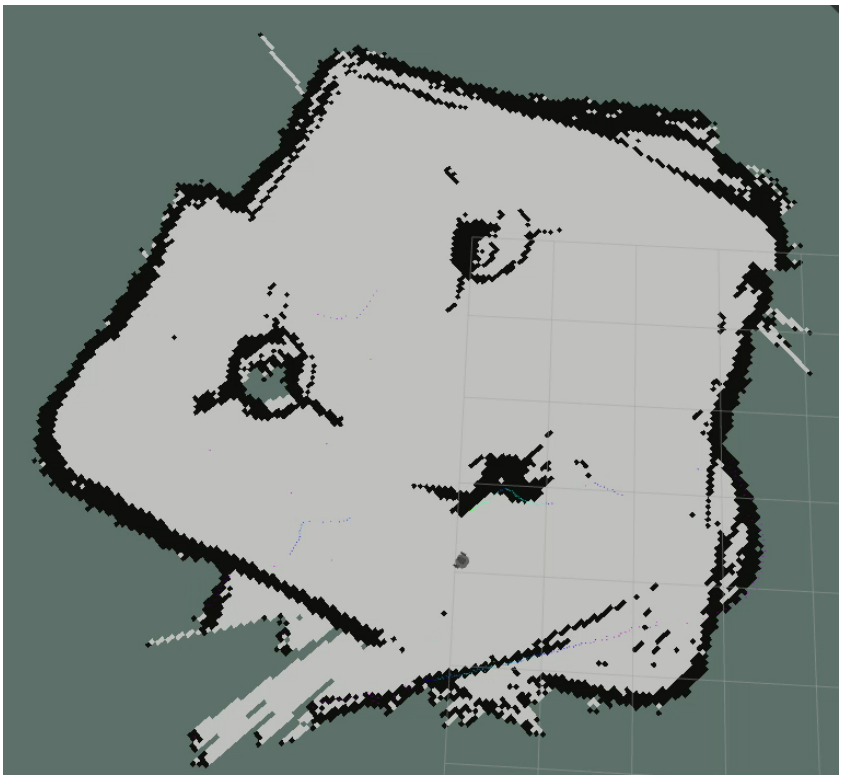


Figure 5.2: Map rendering with Gmapping algorithm

In comparison, the map rendered by Hector_SLAM algorithm can be seen in the Figure 5.3 below.

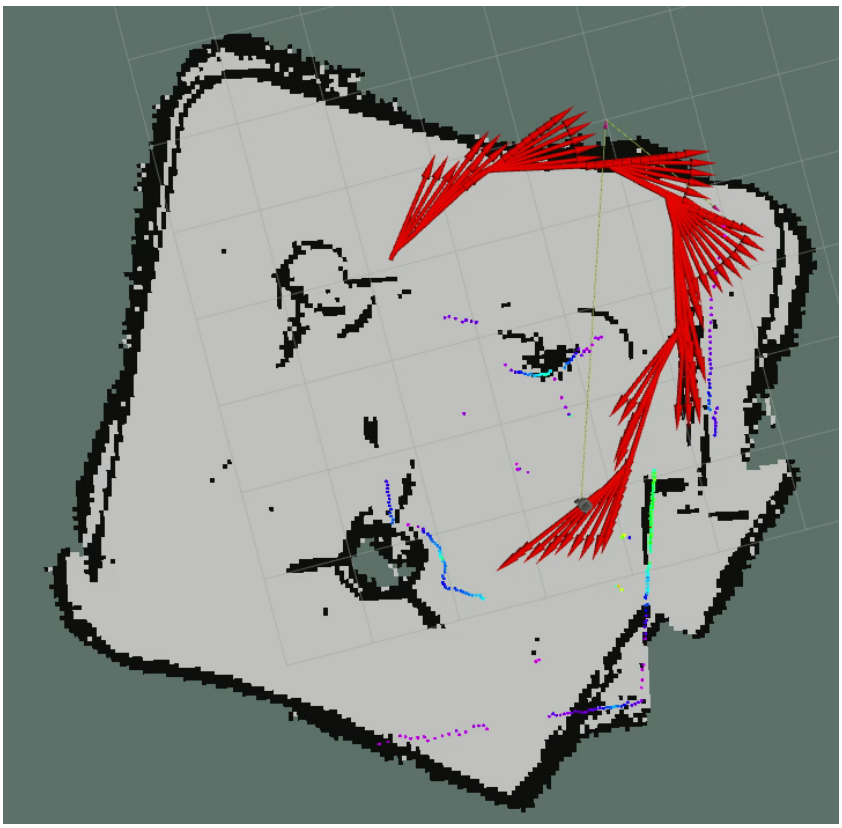


Figure 5.3: Map rendering with Hector_SLAM algorithm

In both maps, the black lines are the obstacles as seen by the Lidar. The light gray shows empty spaces where there are no identified obstacles. The area of the room was approximately 50 square meter, and gmapping algorithm requires 13 minutes to draw the map, whereas Hector_SLAM needed 10 minutes and 15 seconds to do so. During both processes, a screen recorder is activated to better show how the map was built by the two algorithms. The video has been uploaded onto YouTube, which links have been included in the Appendix 2 of this report.

From the gathered data, it is inconclusive to determine which algorithm is objectively better in all settings and scenarios. Gmapping algorithm can be disorienting for the robot when the surrounding obstacles are not inanimate, for example walking passersby or moving obstacles. Since Hector_SLAM relies on encoded data from the Dynamixel servo motor, the algorithm is not suitable when the robot is operated on rough, uneven surfaces such as carpeted ground. The robot's slippage causes Hector_SLAM to draw the map inaccurately.

5.4 Comparison of Similar Robots

In this particular industry there are other competitors with similar or different variations to the modular solutions that what we offer, they promote robots that operate by moving through the air rather than ground movement.

Figure 5.4: ERLE-COPTER drone kit from Erle Robotics

This particular method is quite conventional as there are certain advantages to this modular solution. Aerial robots can access locations and navigate them quite easily. Due to their flying capabilities which allows them to fly lower and in more directions, therefore resulting in more freedom and less movement restrictions in comparison to ground robots. They can be deployed from any location and being mapping immediately. Being airborne means that any obstacles stemming upwards from the ground up will be mapped and visualised in a 3 dimensional map where the Z-coordinate is used.

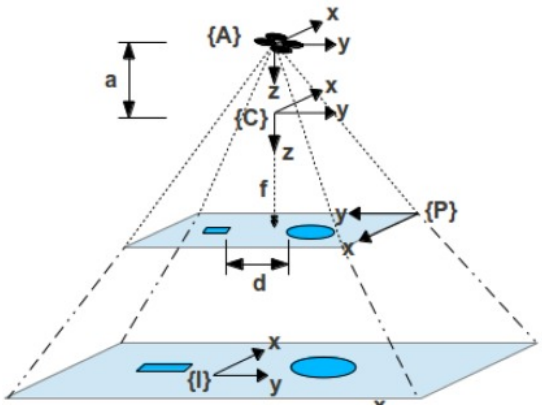


Figure 5.5: Coordinate frames from "An Aerial–Ground Robotic System for Navigation and Obstacle Mapping in Large Outdoor Areas"

Figure 5.5 above demonstrates an aerial robot mapping the surrounding area below it and taking into account the obstacles marked in blue. The first layer, the image plane (I) has the distance between the two obstacles (d) and the X,Y, position from them. The second layer, labelled the Pixel plane receives the data, reduces the size of the first layer and has the X,Y,Z position from the camera frame (C), which then ties in directly with the Aerial frame (A) and maps the position in relation to the previous levels data. As advantageous as this

is there are external factors which affect the performance of the drone and its mapping and navigational capabilities. Variables taken into account can range from changes in weather, wind direction/speed, battery power, hardware sizes and strains, possible malfunctions, precipitation, humidity, snow etc. Variables that don't affect ground robots include changes in weather as the ground robots can be configured for all types of weather, wind conditions as well do not have that much of an effect as well. Variables such as battery power, hardware sizes and strains possible malfunctions are variable that do affect the ground robot as well but not to a a large extent as it does with aerial robots. Precipitation wouldn't be able to affect a robot operating on the ground as the robot will be using caterpillar treads to move through the terrain.

Figure 5.6: ERLE-ROVER from Erle Robotics

When comparing other robots on the market with our modular ones, various factors need to be taken into account, size would be one of them. A larger grounded robot will need a larger battery, larger motors and frame. A larger frame doesn't equate to a better robot, the mobility of the solution would be at risk. A larger robot would have trouble autonomously navigating a rough terrain area in comparison to a smaller more optimized version of the robot would be more than capable of doing the task. Our goal is to solely focus on movement on the ground to an acceptable premise as there is not a lot of external factors to focus on.

In relation to size, the design must also be criticized. A small robot may be optimal but if the design is poor then the robot will suffer as a result. As seen in Figure 5.7 below the ERLE ROVER has been designed for outdoor mapping and navigation, this particular design is not optimised for traversing rough terrain due to the frame and tires. The frame is quite light which can result to the robot flipping if encountering an awkward angle on the terrain floor and choosing to still move forward.

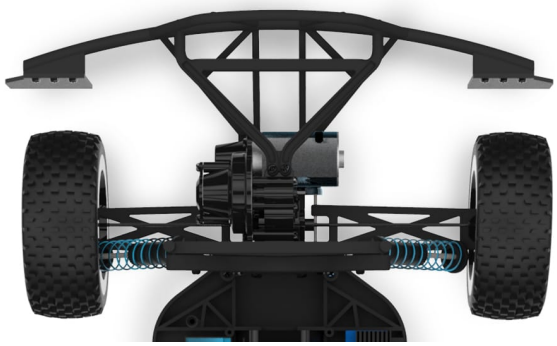


Figure 5.7: Zoomed in image of the single motor

Using one motor for two wheels results in low torque, a more optimised solution to this problem would be for two motors, one motor would control two wheels using a caterpillar tread. The type of wheel is another issue to be addressed, having four individual wheels grouped into two means that each set would have to overcome a tough incline. The size of the body can become stuck/damaged if the robot encounters and sudden change in ground level as the rear wheels would still push the robot forward while the body is grazing the ground floor. If any of the wheels fail then the mobility will experience a big loss, changing the rounder wheels for caterpillar treads removes this issue, as illustrated by Figure 5.8 below.

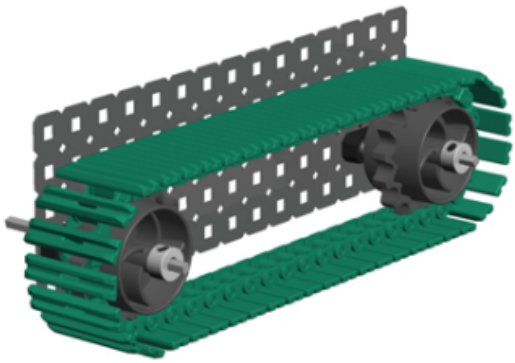


Figure 5.8: Basic track profile

Caterpillar treads ensure that the robot will be able to cope with sudden changes in ground level without damaging the main body, as well as being able to traverse over obstacles to map the surrounding areas in comparison to round wheels.

Chapter 6

Risk Assessments

The risks identified in the project are listed as the following.

1. **Traffic Hazard**
   TurtleBot3's movements can be controlled remotely, hence unaware passersby may trip on the moving robot. Users operating the robot unaware of the environment/surroundings of the robot may operate the robot to outdoor surfaces (i.e. soil/body of water). To alleviate the risk, the robot shall be Operated only within a restricted space and its movement will be monitored in person. This risk is also the reason why Raspberry Pi Camera is installed to the front side of the robot. Finally, nearby passersby will be alerted when the robot is operated.

2. **Electrical Hazard**
   The robot may not be connected to the power outlet properly during charging. This risk can be reduced by only charging the TurtleBot3's battery in the prepared power mains in the laboratory or during the exhibition.

3. **Fire Hazard**
   Due to the utilisation of an 11.1V Lithium-Polymer (Li-Po) battery as the TurtleBot3's power supply, proper handling process must be observed carefully. Li-Po batteries can explode and burn all of the cells when charged incorrectly. Since the battery is essential and cannot be substituted, the risk can only be reduced by alleviating the impact should any accident happen. To do so, a Li-Po handling bag will be used to transport the battery, as well as preparing a bucket of sand as the battery's disposal in the event of explosion. The operators of the robot should also be aware of where the dry chemical fire extinguishers are located when operating the TurtleBot3.

Chapter 7

Timeline Project Plan

	Task	Start Date	Finish Date
1	Installing Linux Ubuntu Mate on Raspberry Pi	30/07/2018	30/07/2018
2	Assembly on TurtleBot3	31/07/2018	31/07/2018
3	Drafting Project Plan	30/07/2018	1/08/2018
4	Finalizing Gantt Chart	1/08/2018	6/08/2018
5	Submission of Gantt Chart (Assignment 1)	6/08/2018	6/08/2018
6	Programming OpenCR Board	1/08/2018	13/08/2018
7	First test of main system	13/08/2018	27/08/2018
8	Mounting Raspberry Pi Camera	27/08/2018	27/08/2018
9	Comparing and Implementing SLAM algorithms	16/08/2018	30/08/2018
10	Drafting final report	23/08/2018	8/10/2018
11	Preparing EnGenius exhibition supporting materials	24/09/2018	16/10/2018
12	Final tests and preparation for EnGenius	9/10/2018	16/10/2018
13	EnGenius Exhibition and Presentation	17/10/2018	17/10/2018

Table 7.1: Project scheduling

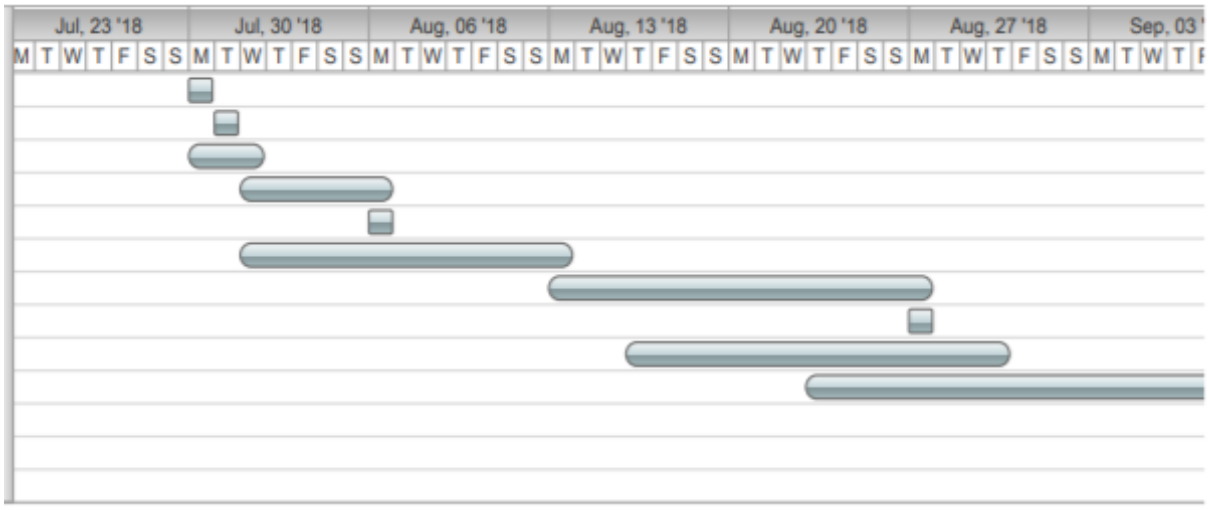


Figure 7.1: Gantt chart for project schedule

Chapter 8

Conclusion

In conclusion learning the way to operate ROS and find different algorithms to get the most accurate map possible mapped out with Rviz we came to a conclusion that Hector_SLAM is the preferred method and algorithm because the maps in comparison to Gmapping was more accurate and we have two links to the videos of both Hector_SLAM and Gmapping that shows the creation of the map live. We have also concluded that the Robot can move autonomously but it needs a sense of direction meaning it need to be given the first coordinate for it to move too and from there it can move around autonomously and map the environment around it.

Chapter 9

Future Works

It is possible to create more accurate maps of an indoor setting with multiple TurtleBots 3's working together. ROS allows one Master computer to control multiple TurtleBots 3's remotely, hence cutting time on building the map.

Effectively using multiple bodies with ROS as the master controller, navigating and mapping buildings with low structural integrity can be done much more quickly and efficiently. Therefore theoretically lowering the mortality rate caused by fires for both civilians and fire fighters. In the exploration field, sending multiple robots to different locations in the surrounding area to map it, will drastically reduce the mapping time. Each robot can scan and transmit the raw data which ROS will then start to stitch together to form a larger map showcasing the X,Y and Z coordinates and the heights of any other obstacles. Ground units as well can utilize this method to send multiple robots out in an unknown environment and map it individually then transmit the data to the master to be put all together.

Offering unique modular solutions means that there are infinite possibilities of hardware and designs combinations for application use. We can add a heat detector to aid firefighters to locate sources of extreme heat, as well as being able to locate civilians due to the varying heat temperatures of the body and its surroundings.

Bibliography

[1] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *Journal of Intelligent & Robotic Systems*, vol. 72, pp. 147–165, Nov 2013.

[2] R. R. Murphy, "Human-robot interaction in rescue robotics," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 34, pp. 138–153, May 2004.

[3] M. Mortimer, B. Horan, and M. Seyedmahmoudian, "Building a relationship between robot characteristics and teleoperation user interfaces," *Sensors*, vol. 17, no. 3, 2017.

[4] D. Belter and P. Skrzypczynski, "Precise self-localization of a walking robot on rough terrain using parallel tracking and mapping," *Industrial Robot: the international journal of robotics research and application*, vol. 40, no. 3, pp. 229–237, 2013.

[5] J. M. Santos, M. S. Couceiro, D. Portugal, and R. P. Rocha, "A sensor fusion layer to cope with reduced visibility in SLAM," *Journal of Intelligent & Robotic Systems*, vol. 80, pp. 401–422, Dec 2015.

[6] E. Kougianos, S. P. Mohanty, G. Coelho, U. Albalawi, and P. Sundaravadivel, "Design of a high-performance system for secure image communication in the internet of things," *IEEE Access*, vol. 4, pp. 1222–1242, 2016.

[7] E. Boukas, I. Kostavelis, A. Gasteratos, and G. C. Sirakoulis, "Robot guided crowd evacuation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, pp. 739–751, April 2015.

[8] G. Tuna, V. C. Gungor, and K. Gulez, "An autonomous wireless sensor network deployment system using mobile robots for human existence detection in case of disasters," *Ad Hoc Networks*, vol. 13, pp. 54 – 68, 2014.

[9] S. Fu, Z.-G. Zhou, and G. Yang, "An indoor navigation system for autonomous mobile robot using wireless sensor network," in *2009 International Conference on Networking, Sensing and Control*, pp. 227–232, March 2009.

[10] R. Barber, M. A. Rodriguez-Conejo, J. Melendez, and S. Garrido, "Design of an infrared imaging system for robotic inspection of gas leaks in industrial environments," *International Journal of Advanced Robotic Systems*, vol. 12, no. 3, p. 23, 2015.

[11] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, pp. 1309–1332, Dec 2016.

[12] S. Saeedi, M. Trentini, M. Seto, and H. Li, "Multiple-robot simultaneous localization and mapping: A review," *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016.

[13] M. Wu, S. Lam, and T. Srikanthan, "A framework for fast and robust visual odometry," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, pp. 3433–3448, Dec 2017.

[14] R. L. Guimarães, A. S. de Oliveira, J. A. Fabro, T. Becker, and V. A. Brenner, *ROS Navigation: Concepts and Tutorial*, pp. 121–160. Cham: Springer International Publishing, 2016.

[15] S. Czarnetzki, M. Hegele, and S. Kerner, "Odometry correction for humanoid robots using optical sensors," in *RoboCup 2010: Robot Soccer World Cup XIV* (J. Ruiz-del Solar, E. Chown, and P. G. Plöger, eds.), (Berlin, Heidelberg), pp. 48–59, Springer Berlin Heidelberg, 2011.

[16] K. Kamarudin, S. M. Mamduh, A. Y. M. Shakaff, and A. Zakaria, "Performance analysis of the microsoft kinect sensor for 2d simultaneous localization and mapping (slam) techniques," *Sensors*, vol. 14, no. 12, pp. 23365–23387, 2014.

Appendix A

Rpi_camera_surveillance_system.py

```
import io
import picamera
import logging
import socketserver
from threading import Condition
from http import server

PAGE="""\
<html>
<head>
<title>Raspberry Pi - Surveillance Camera</title>
</head>
<body>
<center><h1>Raspberry Pi - Surveillance Camera</h1></center>
<center><img src="stream.mjpg" width="640" height="480"></center>
</body>
</html>
"""

class StreamingOutput(object):
    def __init__(self):
        self.frame = None
        self.buffer = io.BytesIO()
        self.condition = Condition()

    def write(self, buf):
        if buf.startswith(b'\xff\xd8'):
            # New frame, copy the existing buffer's content and notify all
            # clients it's available
            self.buffer.truncate()
            with self.condition:
                self.frame = self.buffer.getvalue()
                self.condition.notify_all()
            self.buffer.seek(0)
        return self.buffer.write(buf)

class StreamingHandler(server.BaseHTTPRequestHandler):
```

```
    def do_GET(self):
        if self.path == '/':
            self.send_response(301)
            self.send_header('Location', '/index.html')
            self.end_headers()
        elif self.path == '/index.html':
            content = PAGE.encode('utf-8')
            self.send_response(200)
            self.send_header('Content-Type', 'text/html')
            self.send_header('Content-Length', len(content))
            self.end_headers()
            self.wfile.write(content)
        elif self.path == '/stream.mjpg':
            self.send_response(200)
            self.send_header('Age', 0)
            self.send_header('Cache-Control', 'no-cache, private')
            self.send_header('Pragma', 'no-cache')
            self.send_header('Content-Type', 'multipart/x-mixed-replace; boundary=FRAME'
            self.end_headers()
            try:
                while True:
                    with output.condition:
                        output.condition.wait()
                        frame = output.frame
                    self.wfile.write(b'--FRAME\r\n')
                    self.send_header('Content-Type', 'image/jpeg')
                    self.send_header('Content-Length', len(frame))
                    self.end_headers()
                    self.wfile.write(frame)
                    self.wfile.write(b'\r\n')
            except Exception as e:
                logging.warning(
                    'Removed streaming client %s: %s',
                    self.client_address, str(e))
        else:
            self.send_error(404)
            self.end_headers()

class StreamingServer(socketserver.ThreadingMixIn, server.HTTPServer):
    allow_reuse_address = True
    daemon_threads = True

with picamera.PiCamera(resolution='640x480', framerate=24) as camera:
    output = StreamingOutput()
    #Uncomment the next line to change your Pi's Camera rotation (in degrees)
    #camera.rotation = 90
    camera.start_recording(output, format='mjpeg')
    try:
        address = ('', 8000)
        server = StreamingServer(address, StreamingHandler)
```

```
    server.serve_forever()
finally:
    camera.stop_recording()
```

Appendix B

Map Building

Gmapping: https://www.youtube.com/watch?v=I1hssrv1OCc&feature=youtu.be

Hector_slam: https://youtu.be/V1pfV9e9gFA

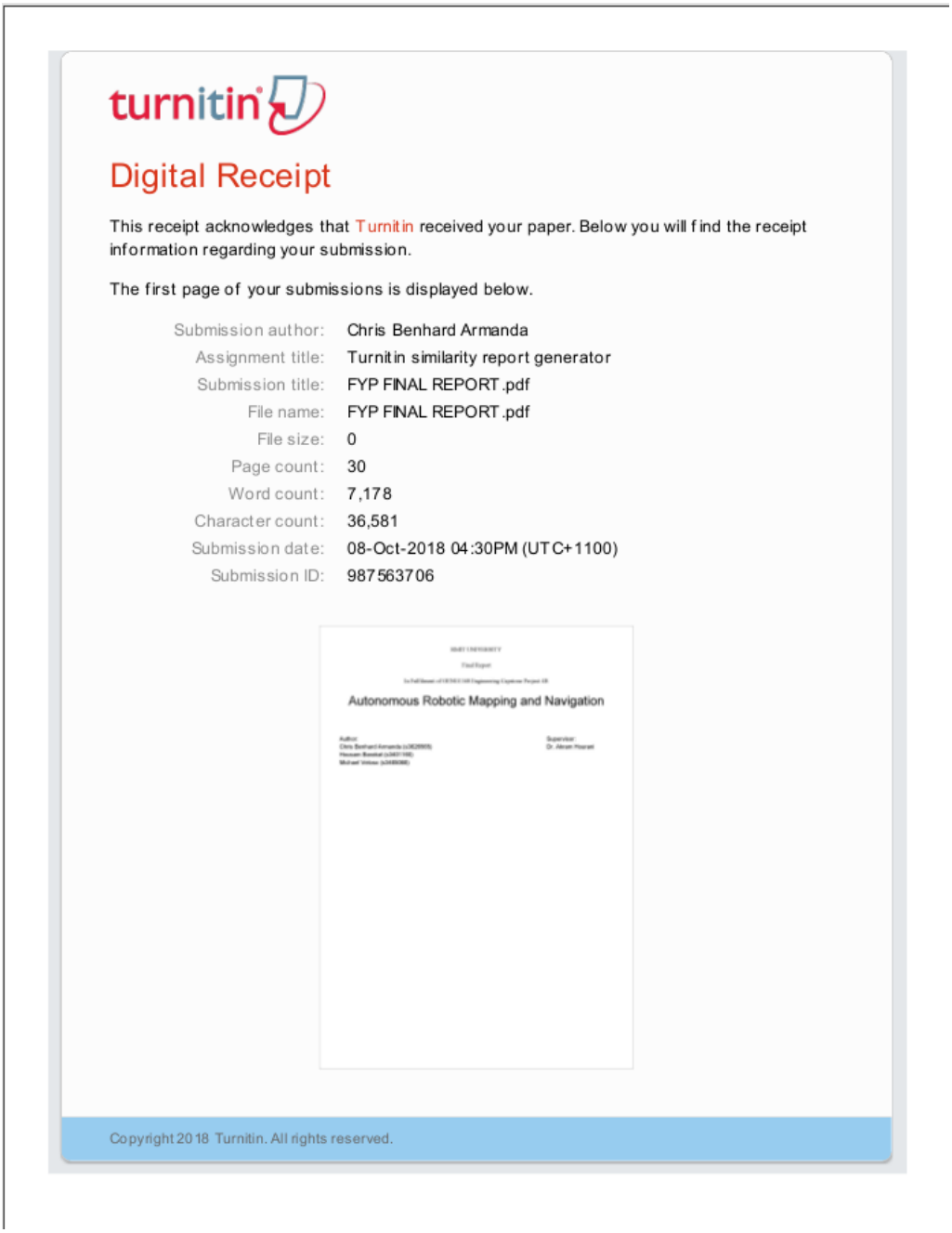
Appendix C

Group Work Member Contribution Table

Report Section	**Person Responsible and Percentages**
Abstract	Michael Veloso (100%)
Introduction	Chris Benhard Armanda (100%)
Statement of Problem	Michael Veloso (100%)
Literature Review	Chris Benhard Armanda (50%) Housam Barakat (50%)
Methodology	
TurtleBot3 Assembly	Michael Veloso (100%)
Robot Operating System	Chris Benhard Armanda (100%)
Installation and Configuration on Linux Machine	Chris Benhard Armanda (100%)
Installation and Configuration on TurtleBot3	Chris Benhard Armanda (100%)
Raspberry Pi Camera	Chris Benhard Armanda (100%)
Rviz	Housam Barakat (100%)
Discussion of Results and Findings	
Operating TurtleBot3	Chris Benhard Armanda (100%)
Drawing Maps	Chris Benhard Armanda (100%)
Comparison of SLAM Methods	Chris Benhard Armanda (100%)
Risk Assessments	Chris Benhard Armanda (100%)
Timeline Project Plan	Housam Barakat (100%)
Conclusion	Housam Barakat (100%)
Future Works	Michael Veloso (100%)

Appendix D

Turnitin Similarity Report



turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Chris Benhard Armanda
Assignment title:	Turnitin similarity report generator
Submission title:	FYP FINAL REPORT.pdf
File name:	FYP FINAL REPORT.pdf
File size:	0
Page count:	30
Word count:	7,178
Character count:	36,581
Submission date:	08-Oct-2018 04:30PM (UTC+1100)
Submission ID:	987563706

Autonomous Robotic Mapping and Navigation

Copyright 2018 Turnitin. All rights reserved.