CG2111A Engineering Principle and Practice
Semester 2 2021/2022

"Alex to the Rescue"
Final Report
Team: B03-3B

Name	Student #	Sub-Team	Role
Lee Zhi Xuan	[redacted]	Hardware	Member
Liong Wei Yong, Deen	[redacted]	Hardware	Member
Marvin Pranajaya	[redacted]	Hardware	Hardware Lead
Ong Chuan Kai	[redacted]	Software	Member
Zhang Wenze	[redacted]	Software	Software Lead

Table of contents

Section 1 Introduction

Over the past 10 years, the annual death rate from natural disasters has ranged from as low as 14,389 to a shockingly high figure of 314,5003 in 2010, according to the International Federation of Red Cross and Red Crescent Societies. [5] The first 72 hours in a disaster is extremely crucial and critical in saving hundreds to thousand of lives. This is where a Search and Rescue (SAR) robot like Alex comes to the rescue. Alex can assist in various ways in a SAR operation. It can fit into places that are dangerous and untraversable by humans and an environment where humans can operate in. Alex can be sent to places which are too dangerous for respondents, which is often needed in disaster areas.

This project aims to build a manually and remotely controlled robot vehicle, Alex, with search and rescue capabilities. Alex has to be able to map its surroundings while identifying foreign objects and effectively overcome them without knocking into it. By utilising a Light Detection and Ranging Sensor (LiDAR), specifically the SLAMTEC RPLIDAR, we can efficiently map out Alex's surroundings. These mapped data points are then transferred via WiFi, over to the remote controller on a laptop running Raspberry Pi (RPi). RPi receives the data and sends out commands to the Arduino Board, which are then sent to the peripheral components attached to Alex, namely the motors and ultrasonic sensors.

1.1 Functionality Summary

No.	Functionalities	Explanation
1	Remotely Controlled	Alex should be tele-operated using a separate device so that the process of searching for casualties can be done remotely. This allows the process to be done quicker and safer, without having to risk human lives.
2	Movement	Alex should be able to traverse terrain by performing simple movements such as moving forward, turning left, turning right and moving backwards, from commands sent by the user. This will be achieved through the use of motors, together with motors encoders attached, to provide accurate distance and angle movements for Alex.
3	Environment Mapping	Alex should detect its surroundings when navigated and send back the data to the operator. The environment that is detected should appear as a map for the operator to provide a sense of what terrain the rescuers would have to deal with. Alex should also be able to compute information such as the distance travelled to reach its current position so that rescuers would have a better idea of the location of the trapped survivors.
4	Proximity Detection	Given the unknown terrain, Alex needs to be able to overcome possible unevenness on the ground, and stop when needed (to prevent collision) in order to achieve mission success.

Section 2 Review of State of the Art

2.1 RAPOSA

RAPOSA is a tele-operated search and rescue robot that was designed to operate in unfamiliar terrains where human access is limited, such as “debris resulting from the collapse of built structures” RAPOSA can detect potential survivors using its set of specific sensors, which include “a thermal camera and two web cameras installed” on the front of the robot. Other hardware sensors include “gas, temperature and humidity sensors”, with additional components such as “webcams, light diodes, microphone and loudspeakers” that can assist in establishing communication with the potential survivors. The structural design and use of tracked wheels also allows the robot to travel when upside down. [2] RAPOSA can either use wireless communications or through tethering.
One of RAPOSA's strengths is the ability to dock and undock to the tether remotely with the help of the camera component, which allows the robot to relocate to a desired position safely. Another strength comes in the form of its structure. With the design of a front body, this allows the robot to be capable of accessing edges that are taller than the main body, which comes in handy when the terrain involves edges such as stairs. Although it can be used either with wireless communications or through tethering, both methods have their own limitations. Tethering involves the use of a cable, which could restrict the movements of the RAPOSA when entering a terrain that requires many turns. The wire can also be scratched or disconnected during the operation. Wireless communications may also struggle depending on the terrain. With the presence of metal and concrete obstacles, edges, wires and electromagnetic interference, these would weaken the wireless communication between the robot and the user. The need to constantly reconnect could waste precious time during the search and rescue operation. [3]

2.2 VGTV XTREME-ROBOT

This robot can assist search and rescue task forces by entering areas that are not safe to be searched by humans and gathering data back to rescuers. [1] The VGTV is remotely powered and has an inbuilt controlled video inspection system, bidirectional audio and remote sensing. There were also two 20W lights are attached to the robot for illumination and the robot can operate in a temperature range from 0°C to 50°C. The angled nature of the dual rear drive spokes, combined with the positioning of the spokes on the belt’s centre, also helps to ensure that the track will self-align if it becomes dislodged from the drive mechanism. VGTV sends video recording back to its user through cable tethering and a handheld control unit provides a joystick for speed and direction and separate controls for the shape change, variable lights, and camera tilt.
One major flaw is the camera itself. As stated above, VGTV is equipped with a manual camera, thus unable to auto focus and waste precious time managing the focus of the image before capturing. It also does not come with a zoom function, thus limiting its capabilities in areas where the bot could not traverse. In addition, VGTV’s had only a 100-foot-long tether cable. There were occasions when the bot was deployed to various voids and had to utilise its tether to the maximum length. This further limit the subsequent distance that the bot can cover and still be able to send the video back to its users. [4]

Section 3 System Architecture

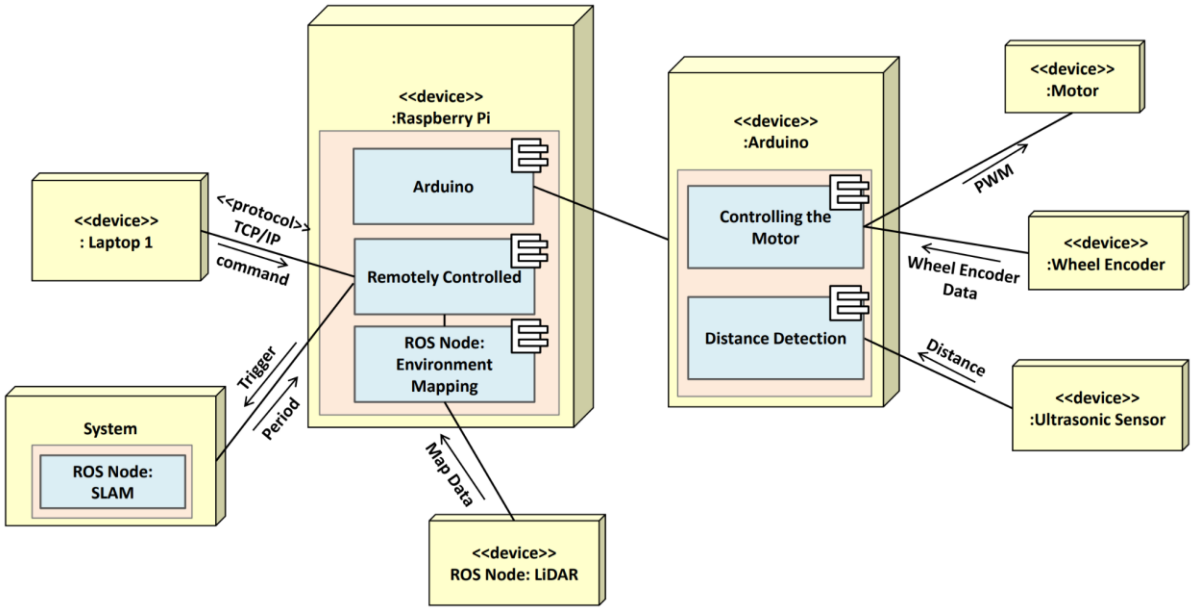

Figure 1

The yellow square boxes represent hardware, blue boxes represent software. The connection between the components can contain information / control passes between them.

3.1 Main architecture of Alex

Alex's overall system consists of 2 main devices: the Arduino and the Raspberry Pi (RPi). The Arduino is in control of the peripheral hardware components namely the motors, wheel encoders and ultrasonic sensors. These data are then directly relayed to the RPi via Universal Asynchronous Receiver Transmitter (UART) through their USB ports. The reason for the use of the UART protocol is due to the RPi's inability to support synchronous communication.

3.2 Arduino

The Arduino (1) controls the speed of the motor, (2) counts the number of times the wheel has rotated via the wheel encoder and (3) derives readings from the ultrasonic sensor.

3.2.1 Motor Controller

The speed of Alex is controlled through the use of Pulse Wave Modulation (PWM) signals. The user inputs a set of commands for Alex to go a certain direction at a given speed. The instructions will then be used to derive the desired PWM duty cycle before sending it to the 2 motors. Upon the completion of the given instructions, the wheel encoder records the information travelled and sends it to the Arduino Module.

3.2.2 Wheel Encoder

We use the wheel encoder to calculate the distance that the robot has turned or driven. It utilises the Hall Effect to detect changes in the magnetic field as the ring rotates. The number of ticks is then recorded and calculated accordingly with additional variables such as the circumference of the wheels and ticks per revolution to derive the turning angle and distance travelled.

3.2.3 Ultrasonic Sensor

When the user requests for the distances from the immediate front and side of Alex, the ultrasonic sensor emits a sound wave and registers the time for the waves to be received back. This allows the sensor to calculate the distance between Alex and its surroundings, namely the

front and right side in our project. This information is then delivered to the RPi and subsequently to the user to provide an accurate understanding of the surroundings.

3.3 Raspberry Pi

The RPi is being powered by the powerbank and serves as Alex's "brain". RPi uses the Secure Shell (SSH) protocol such that external devices can connect to it wirelessly. It also serves as a Transport Layer Security (TLS) server which handles communication between user and RPi. Moreover, it transmits and receives data (control and telemetry) to the Arduino Uno via Universal Asynchronous Receiver-Transmitter (UART) protocol. The RPi is a ROS slave whereby it handles the RPLIDAR ROS node and publishes a topic for the ROS Master to subscribe. The ROS Master (Laptop) will then perform SLAM (will be discussed in detail in Section 6). With this communication stack in RPi, it allows Alex to be controlled wirelessly and also transmit LiDAR data over WiFi.

3.3.1 Wireless control (Remotely Controlled)

RPi will initiate a TLS server and listen for an incoming connection from the laptop. After the Laptop request for a connection, RPi will check for its credential and approve the connection if valid. Subsequently, the laptop sends encrypted data to the RPi, which RPi will then serialise the commands into a command packet and send it to Arduino via UART.

3.3.2 Arduino

The Arduino module continuously reads data from the UART port. Upon receiving the command packet, it deserializes the command packet and subsequently handles it according to the predefined instructions laid out in the header file. The commands are then executed by the peripherals attached to Arduino. After each command, it will send an OK packet to indicate successful execution of the command. The user can also request for telemetry (distance travelled by each motor, ultrasonic reading, etc.), which will be sent through UART to RPi and then TLS over to the user's laptop.

3.3.3 LiDAR (Environment Mapping)

The LiDAR provides a 360 degree scan field with a rotating frequency of 10 Hz to provide environment maps. The map data is sent back to the RPi and communicated to the Hector SLAM ROS Node in the user's laptop for the mapping to be shown in RViz.

Section 4 Hardware Design

4.1 Overview

This section showcases the placements of the various hardware components. Extensive planning and discussion took place before we began constructing the robot. We aimed to keep the robot's CG as low as possible while ensuring that we do not hinder the robot's balance and manoeuvrability. Figure 2 shows the 3 layers (top, middle and bottom) and the arrangement of the components while figure 3 includes a description for each component.

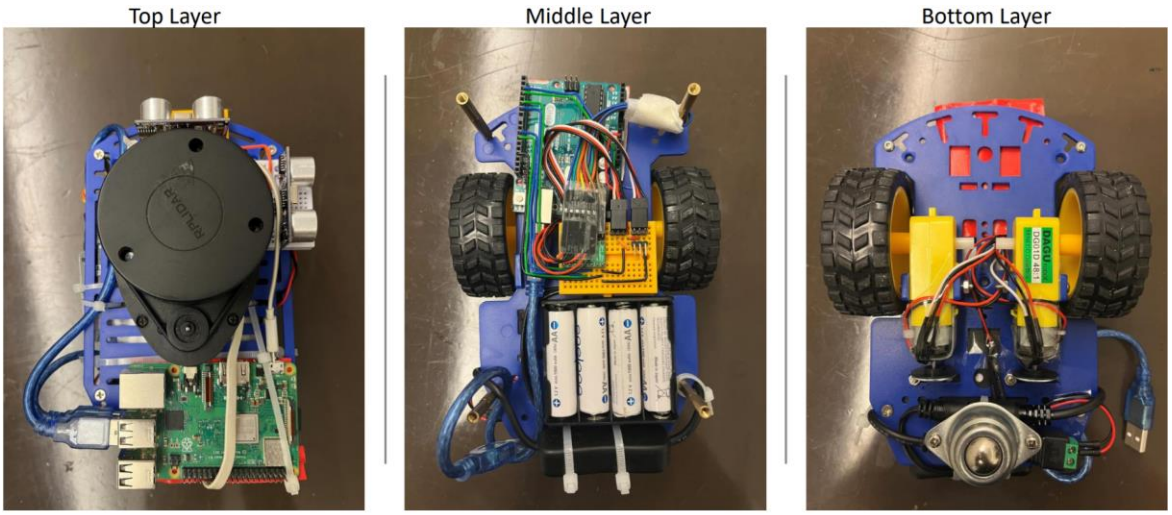

Figure 2

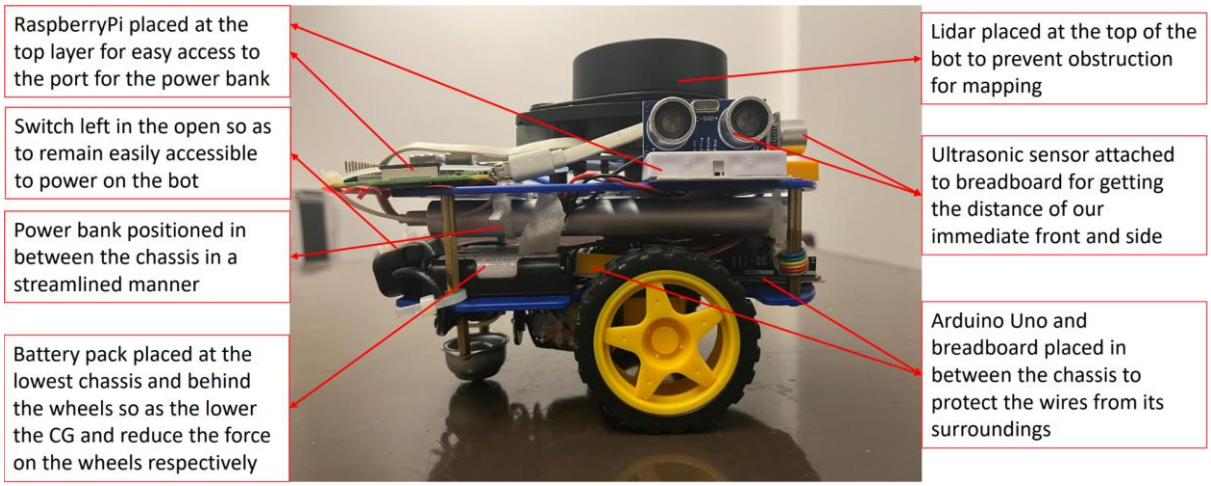

Figure 3

4.2 Placement of Primary Components

The basic components include the LiDAR, Arduino Uno, Raspberry Pi, power bank, battery pack, motors, wheels and wheel encoder.

The top layer consists of the LiDAR and Raspberry Pi. The LiDAR is mounted at the top to ensure that none of Alex's components will be an obstruction and interfere with the mapping process. This will allow for a reliable and accurate map reading by the LiDAR. The RPi is also positioned at the top to ensure enough airflow to prevent the RPi from overheating

The middle layer consists of our Arduino Uno, the battery pack, the power bank and a Breadboard Hub. The battery pack is positioned towards the back of Alex so as to reduce the

force on the wheels and also lower the chance of toppling over while Alex goes over the hump. The breadboard hub is strategically placed so as to manage incoming and outgoing wires between most components and the Arduino. This is done to prevent wire clustering around the Arduino GPIO, which will increase debugging difficulty. We used solid core wire to ensure proper connection from the hub to the Arduino. The middle layer offers protection as it minimises wires exposed to the surroundings which will reduce the chances of any accidental disconnection of the wires. The power bank is then placed above the Uno and battery back but connected to the underside of the upper chassis in a streamlined manner. This is to ensure that we achieve an evenly distributed weight throughout Alex .

Lastly, the motors, wheels, wheel encoders and Omni wheels are mounted on the underside of the lower chassis with the wires just long enough for proper operation. This is to ensure no wires are dangling below Alex.

4.3 Placement of additional components

The top layer involves 2 additional ultrasonic sensors. Similar to the LiDAR, we positioned it at the top layer as the middle layer is reserved for wire management. This prevents any wire to unintentionally obstruct the sensor while getting the distance of our immediate front and right side. The front reading gives us an estimate on the distance Alex can travel before encountering an obstacle, therefore allowing us to command Alex to travel large distances with confidence. In turn, the time and command packets needed by Alex to travel the same distance can be reduced significantly.

Meanwhile, the side readings serve as an additional guide while operating Alex especially in tight spaces. All in all, the ultrasonic sensor serves as a redundancy in addition to our LiDAR if it ever fails.

Section 5 Firmware Design

5.1 High Level Algorithm

1. Initialisation of Arduino
    a. Setup of Arduino, initialisation of interrupts and ports.
    b. Waiting and establishing communication with RPi at a baud rate of 9600

2. Receiving Commands
    a. Arduino polls for new instructions from RPi
    b. Upon receiving the instructions, the data is de-serialised and read into the packet form
    c. If the transfer of data was unsuccessful, a bad response packet would be sent back to indicate the error.
    d. If transfer of data was done successfully, an OK packet would be sent in response. Instructions in the packet will be sent to the peripheral component and specific actions will be carried out by Alex.

3. Executing Commands
    a. Movement
        - Using the predefined constants defined in the header files, it will execute the movement command accordingly with the parameters (speed/angle) sent from the RPi.
    b. Ultrasonic Sensor
        - Ultrasonic sensor emits a sound wave and registers the time for the waves to be received back. The duration taken for the sound wave to return is used to calculate the distance away from the object by multiplying time taken with the

speed of sound divided by 2. The data will then be transmitted back to RPi via UART.

4. Repeat from step 2 until connection to the RPi is disconnected or Alex is turned off.

5.2 Communication Protocol

A. Arduino UNO and RPi

Alex uses a custom data packet to communicate between different devices. This is necessary because Arduino UNO is a 16-bit microcontroller while RPi is a 32-bit ARM device, thus they must agree on the communication structure beforehand. Alex utilises the UART protocol with a baud rate of 9600.

The custom packet is called TPacket, and it is sent in the following structure:

Variables	Size	Description
char packetType	1	Indicates the type of data packet sent. It can be a response, command, error, normal or hello message.
char command	1	Indicates the type of command. There are 7 possible commands.
char dummy	2	Pad the data which resolves the differences in compiler between Arduino and RPi.
char data[32]	32	Contains the information on packet types, commands and status constants.
uint32_t params[16]	64	4 sets of 16 bytes to store information for telemetry.

B. RPi and User Laptop

Both devices are communicating through SSH which will then initialise the TLS server and client connection for subsequent use. They use the same header file that contains the TPacket data type and constants used in Arduino.

The wireless network is encrypted through a Certificate Authority (CA) that we created in one of the lab sessions. To initialise a connection, the RPi and the Laptop need to have the valid CA file that is shared to each other. The server (RPi) must have a verified server certificate and key. The client (Laptop) also must have a verified client certificate and key. Once the connection is established and verified, subsequent communication will be done through Secure Socket Layer (SSL) connection.

Section 6 Software Design

6.1 High Level Algorithm

1. Initialization of TLS connection
    a. User will SSH into RPi
    b. Initialise TLS Server on RPi, establishes UART connection with Arduino and listens for connection from client.
    c. Laptop initialises TLS Client and searches for a connection
    d. RPi and Laptop connects via a TLS connection
    e. RPi sends a "Hello" packet to the Arduino
    f. Arduino will reply with a "Hello" packet

2. Initialization of ROS nodes and receive LiDAR data
    a. Set Laptop as ROS Master and set RPi as ROS Slave.
    b. On Laptop, run `roscore` to initialise it as Master. Then run the Hector SLAM ros node.
    c. On RPi, run the RPLIDAR ROS node to start publishing data to the Hector SLAM node in Laptop.
    d. The map will appear in the Laptop RViZ window.

3. Send and receive user command
    a. User types command into terminal of Laptop
    b. Laptop serialises the command into a command packet and sends it via the TLS connection to the RPi
    c. RPi reads the packet via sslRead. As long as there is network data received, the RPi will handle the network data
    d. RPi checks network data received against the list of valid commands
    e. If network data contains a valid command, RPi serialises the network data and sends it via UART to the Arduino

4. Carrying out user command
    a. Arduino continuously reads data from the UART port
    b. When Arduino receives serialised command packet from RPi, it deserializes the command packet and handles the command packet
    c. If it is a valid command, it sends an "OK" packet to the RPi to signal that it is ready to execute the command
    d. Arduino will then execute the command based on the predefined command constants in the header file.
    e. If telemetry data is requested, Arduino will calculate the ultrasonic sensor distance, and send it back along with other data to the RPi.

5. Repeat Step 2 until navigation is over
    a. While server is running, the RPi will continue to listen to any commands sent from the Laptop
    b. When user sends a new command, Step 2 will be run again

6.2 Additional software-related capabilities

Other than the mandatory functionalities required in Alex, we have also implemented other capabilities into Alex to aid us with our project objective.

6.2.1 Robot Operating System (ROS)

ROS software packages allow independent applications (nodes) to transmit (publish) and receive (subscribe) information through a data "tunnel" called topic. This allows different softwares to communicate easily and with little user intervention on how the data is handled. In our application, the RPLIDAR node publishes its data points onto the topic `/scan`, which is then subscribed by the Hector-SLAM node to do calculations and show the mapping in RViz.

6.2.2 RViz Configuration

1. To increase the robustness of our SLAM system, we subscribed topic `/scan` in our RViz configuration so that the real-time scanning results of LiDAR (red dots in Figure 4 below) can be seen in the map. In this case, even when the Hector SLAM algorithm failed to identify and draw an obstacle, we were still able to recognize it and controlled our robot to dodge the obstacle.
2. We also drew a grid with 27cm by 27cm cells (green grid in Figure 4 below) in RViz. We added a slight offset to the grid before we started the robot, making it aligned with the real maze so that we could draw the maze more accurately.

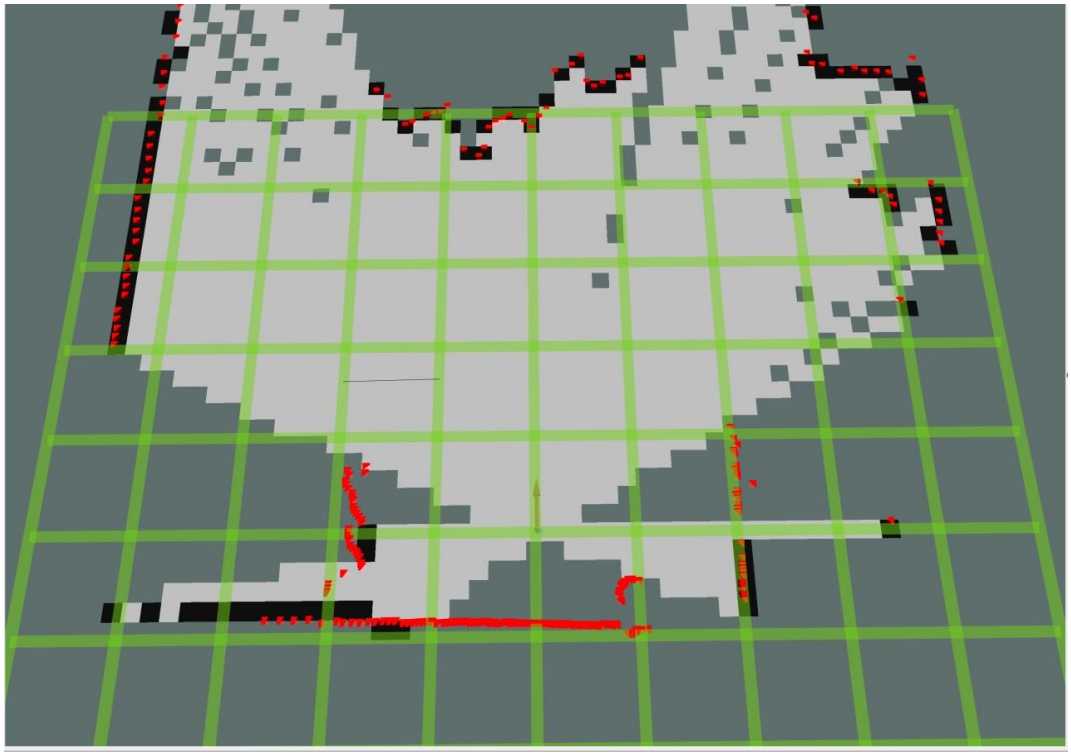

Figure 4

6.2.3 Hector SLAM

Hector SLAM is an implementation of SLAM that does not require odometry data and can be used on platforms that exhibit roll/pitch motion [6]. By matching the current laser data with successfully associated data, the algorithm is able to identify the robot's position and orientation in the existing map. Additionally, Hector SLAM can compute a new fit line which represents an obstacle if there are too many related data points that cannot match any of the current fit line. However, Hector SLAM does not have self-correction capability and needs

relatively high data update frequency and thus the robot can only map correctly when it moves at a lower speed.

6.2.4 Incremental Movement Upgrade

After multiple testings, we have realised that we need to implement a solution to allow for more precise yet simple movements. By implementing fixed movement keys – represented by keys W, A, S, D for forward, leftwards, backwards and rightwards respectively – we were able to allow for more precise movement of Alex as these commands will move Alex by a fixed precise distance at a pre-set speed. The commands are still sent via SSL, and it is achieved by allowing the client and server to accept this keyword and preset it to run with a certain amount of distance/angle and speed. This allowed us to send commands quicker, which results in a faster and more accurate response to the surroundings detected.

Section 7 Lessons Learnt - Conclusion

7.1 Overview

Our objective in this project is to build a remotely operated search and rescue robot, Alex, to aid us in mapping out an area for search and rescue operations. As such, we needed to design and implement suitable functions in Alex to achieve its intended capabilities of remotely controlled movement and environment mapping.

7.2 Two greatest mistakes made

1. Following the Alex Assembly Guide in Week 8 Studio 1, we used pins 5, 6, 10 and 11 (connected to the AIN1, AIN2, BIN1 and BIN2 of the DRV-8833 Dual Motor Driver Carrier) to control the motors. However, doing so resulted in us using all 3 timers available in the Arduino. Hence, when we decided to implement bare-metal to the code, we wanted to utilise one timer to implement the delay function and was unable to do so as none of the timers were available. This became a significant problem for us and ultimately, we had to write our bare-metal code to toggle the timer from PWM mode to CTC mode every time we wanted to use the delay function (and back to PWM mode afterwards). Doing so caused our code to be more complicated and prone to bugs as a result. Therefore, the mistake of not considering our usage of timers earlier resulted in us facing issues in the later stages of the project.
2. In the earlier stages, we did not realise that the position of the battery pack would have a great impact on the weight distribution, specifically on the omni wheel's ball bearing. Majority of Alex's weight was pressing down on the omni wheel hence preventing the ball bearing from spinning smoothly and we had to increase the power in the motors to manoeuvre Alex. It was after multiple attempts to mitigate Alex's inability to move smoothly (such repeatedly applying exorbitant amounts of lubricant on the omni wheel) before we decided to change the position of the powerbank. This solved the weight distribution completely and thus this grave mistake of ours cost us a large amount of time.

7.3 Two most important lessons learned

1. Importance of good planning
   As a result of our mistakes made during the project, we have learned that we need to develop and finalise the design of Alex earlier to ensure such that we can do more iterative design when met with technical issues. Good planning would have allowed us to consider the design of Alex in greater detail and prevent us from having to redo its design repeatedly.
2. Proper integration of hardware and software capabilities

The hardware and software used in the implementation of Alex have differing limitations. As such, when designing robots, we have to consider those limitations to integrate the hardware and software capabilities seamlessly so that the robot will work as intended. For example, Hector SLAM is unable to perform reliably if the movement of Alex by the motors are above a certain speed.

7.4 Conclusion

Completing this project has provided us with a great entry point into the realm of software-hardware integration in robotics. Firstly, we were exposed to hardware by building the different components of Alex and making sure they can work independently. Secondly, we handled the implementation of the software required to carry out the intended capabilities. Lastly, we had to integrate the hardware and software components in Alex so that they can work in tandem to achieve our project objective of creating a search and rescue robot. With these skills obtained from CG2111A, we are thus able to step into the realm of robotics with sturdy feet and continue to make progress in our journey towards becoming a computer engineer in the later modules.

References

[1] Boyle, A. (2005, September 8). High-tech goes into action in disaster zone. NBCNews.com. Retrieved March 29, 2022, from https://www.nbcnews.com/id/wbna9240563

[2] Marques, C., Cristóvão, J., Alvito, P., Lima, P., Frazão, J., Ribeiro, I., & Ventura, R. (2007). A search and rescue robot with Tele-operated Tether Docking System. Industrial Robot: An International Journal, 34(4), 332–338. https://doi.org/10.1108/01439910710749663

[3] Marques, C., Cristovao, J., Lima, P., Frazao, J., Ribeiro, I., & Ventura, R. (2006). Raposa: Semi-autonomous robot for rescue operations. 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. https://doi.org/10.1109/iros.2006.281836

[4] Micire, M. J. (2007). Evolution and field performance of a rescue robot. Journal of Field Robotics, 25(1-2), 17–30. https://doi.org/10.1002/rob.20218

[5] Walker, J. (2019, November 21). Search and rescue robots - current applications on land, sea, and Air. Emerj Artificial Intelligence Research. Retrieved April 10, 2022, from https://emerj.com/ai-sector-overviews/search-and-rescue-robots-current-applications/

[6] Wiki. ros.org. (n.d.). Retrieved April 21, 2022, from http://wiki.ros.org/hector_mapping