CG1112 Engineering Principles and Practices
Semester 2 2019/2020

"Alex to the Rescue"
Final Report
Team: 4-1-2

Name	Student #	Main Role
Wira Azmoon Ahmad		Leader
Theodore Pinto		Quality Checker
Toh Yong Xiang Brandon		Coordinator
Sugandha Tuteja		Sub-Team Leader
Ganapathy Sanath Balaji		Sub-Team Leader

Project Video Link: https://youtu.be/A9tuBDnDaaU

Purpose

The main purpose of this report is to summarize the different functionalities of Alex, and the different parts involved in building it. This report talks about the need for Alex, a brief review of state-of-the-art technology to take inspiration, the system architecture, hardware, software and firmware design, power management, and the mistakes and lessons that were learnt. The final aim of Alex is to search and rescue, and the ability to map unknown environments and traverse through without collisions is just the first step in Alex achieving this goal.

Section 1 Introduction

Disasters often occur without notice, and there have been a spate of them in the 21st century, leading to the loss of many lives, such as the 9/11 disaster in New York [1] and the 2011 earthquake in Japan [2]. These disasters and the loss of lives highlight the importance of exploring the field of technology in order to aid in search and rescue in the aftermath of such catastrophes.

Alex is a robotic vehicle that can map out its surrounding terrain within approximately 75cm from its current location and can easily move around. An operator can view the surroundings of the robot as data points on a graph via an application called gnuplot [3] and move the robot remotely from a laptop, via console commands, after establishing communications with it. With continued work and further developments, it would be possible for Alex to become a search and rescue robot.

Section 2 Review of State of the Art

The following section describes two tele-operating search and rescue robotic platforms, WALK-MAN, and RAPOSA. It gives a simple description of their system, focusing on its functionalities and hardware and software components, as well as their strengths and weaknesses.

1) WALK-MAN

a) WALK-MAN is a humanoid robot developed by a team at the Italian Institute of Technology (IIT) [4] for search and rescue operations. The robot can be controlled by a pilot remotely and interacts with its surroundings using its body parts to accomplish tasks. It also provides a live video feed to the pilot and has the capability of measuring distances.

Its parts are modelled after human body parts and contain cameras to replicate human eyes and electrical motors to help it move (unlike hydraulic systems used for similar robots). ROS (Robot Operating System) and YARP (Yet Another Robot Platform) are used mainly to develop the software [5]. ROS functions as the robot's operating system whereas YARP is used to control the robot.

b) Being a humanoid robot, it can easily interact with objects familiar to humans, especially for urban rescue operations. This is one of the WALKMAN's biggest strengths, giving it an edge over other rescue robots.

However, it cannot function in tight workspaces as other compact rescue robots can due to its large size.

2) RAPOSA

a) RAPOSA is a robot for Search and Rescue operations and was designed to work in harsh conditions such as navigating remains from collapsed structures [6]. The first robot was designed by ISR/IST and developed by spinoff IdMind. Its original function was search operations only by transferring data from sensors to remote operators. It included 3 conventional cameras, one thermal camera, several explosive and toxic gas sensors, temperature and humidity sensors, inclinometers, artificial lights, microphone and speakers. It was designed to easily fit sewer pipes in cities and climb standard stairs.

b) The strengths of this robot include its ability to operate without any malfunctions even after flipping upside down [7]. This is due to the robot being able to determine its orientation and not having a defined top or bottom.

After several tests were done on the robot, some problems were encountered concerning wireless communications. These included the position of antennas on the robot and interference from other wireless networks, causing unreliable communications.

Section 3 System Architecture

Alex is made up of 5 components, namely the Raspberry Pi (Pi), RPLiDAR, Arduino Uno (Arduino), Motors and Wheel encoders, and is controlled using a Personal Computer (PC). The user can give instructions to the Pi via the PC and the Pi would transfer these instructions to the respective devices.

If a movement command is given to the Pi, the instruction would be transferred to Arduino which in turns moves the motor accordingly. If a command to generate a GNU plot of the environment is given, the instruction would use information gathered in a data file to do so. This file, located on the Pi, is updated by the LiDAR in real time. The PC would then be able to view the plot of the environment data via the Pi.
A schematic diagram of the interactions between the devices is shown in Figure 1 below.

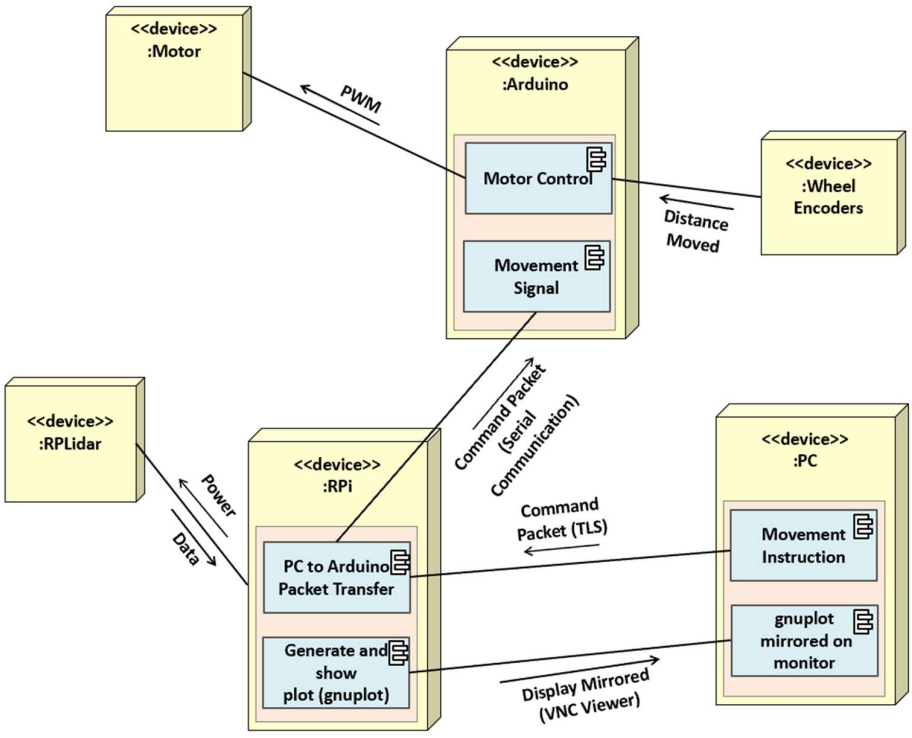


Figure 1: Interactions between the various devices of Alex and the PC

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

Section 4 Hardware Design

Different hardware components are present in Alex, which work hand in hand with each other. Figures 2,3,4 and 5 below show the various parts of the robot. The details of each component are explained in Table 1 on the next page.

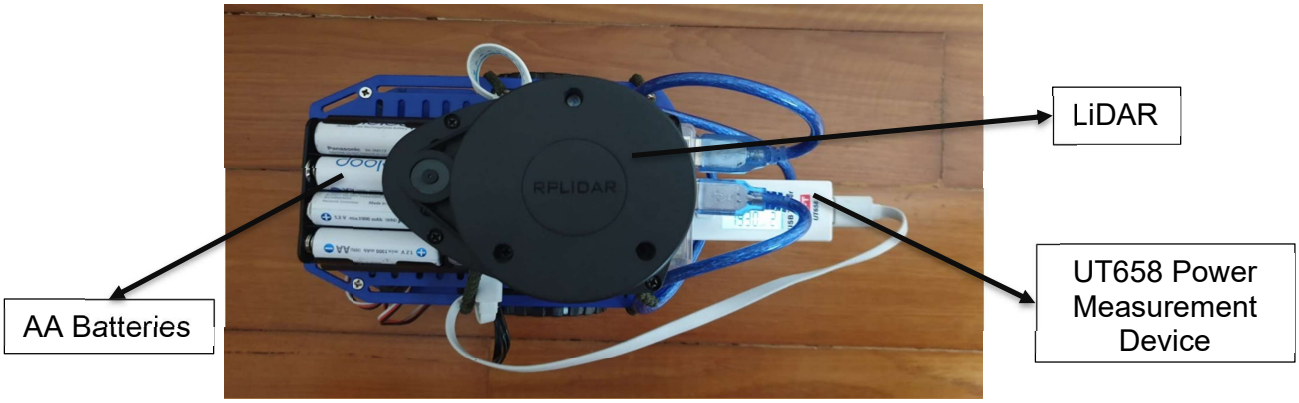


Figure 2: Upper Deck of Alex

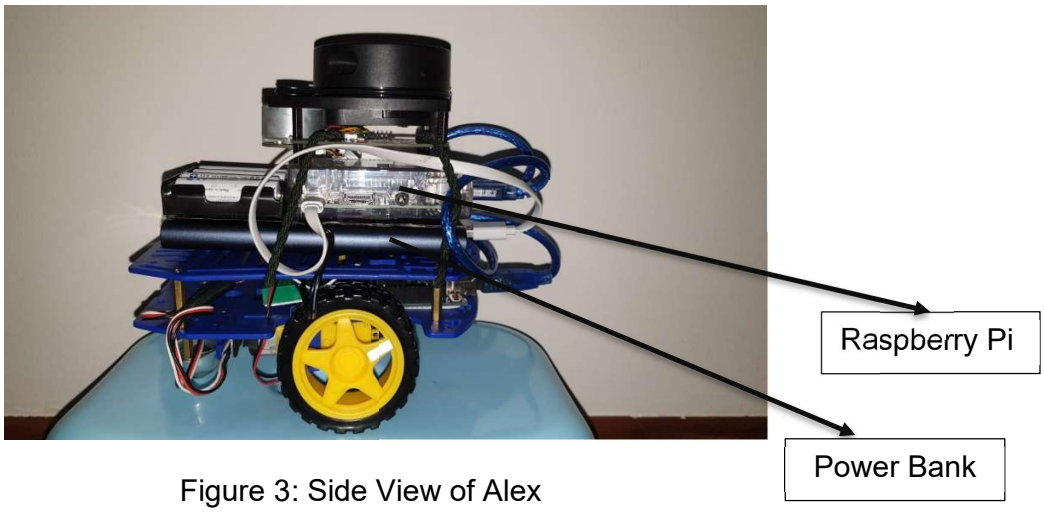


Figure 3: Side View of Alex

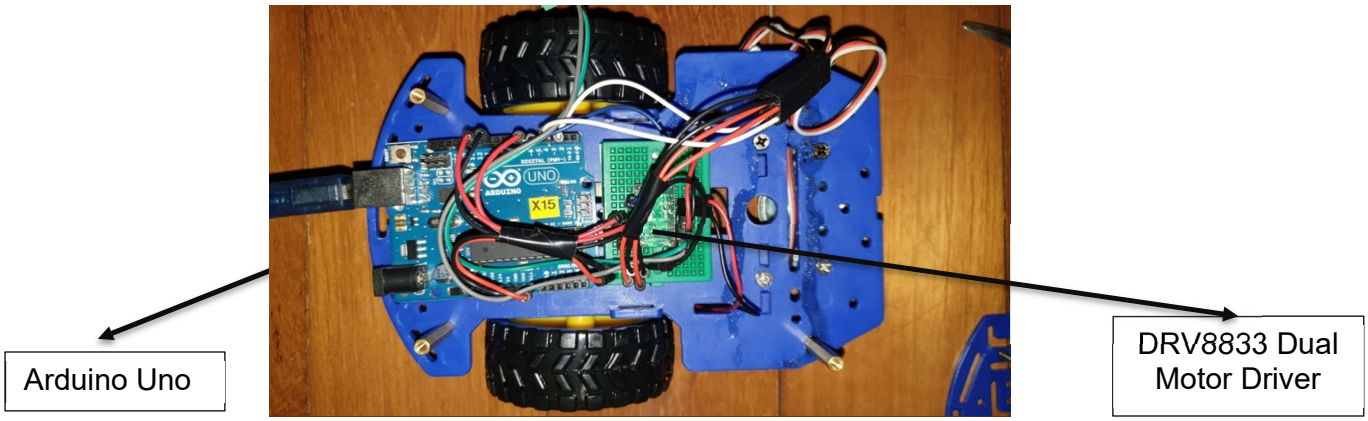


Figure 4: Lower Deck of Alex

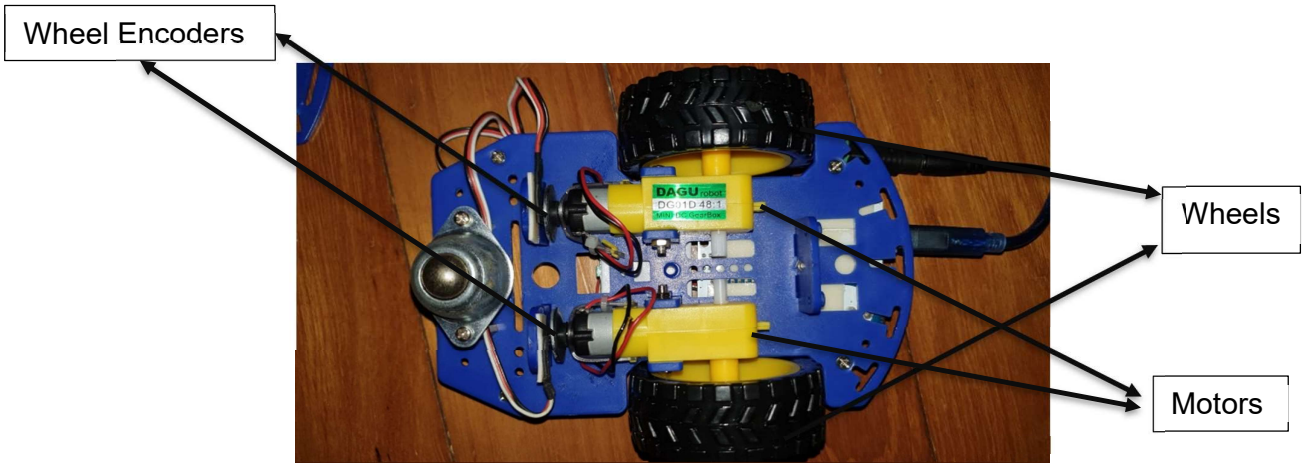


Figure 5: Bottom View of Alex

Name	Function	Placement
Raspberry Pi	1) Sends information over to the Arduino to control the motors 2) Transmits power to the LiDAR and the Arduino Uno 3) Displays the data obtained by the LiDAR via gnuplot	Upper deck of Alex, secured above the power bank with Velcro
LiDAR	Maps the surrounding environment and transmits this data to the Raspberry Pi	Upper deck of Alex, on top of the Raspberry Pi to ensure that there are no wires blocking the sensor
4 AA Batteries	Provides power to the DRV8833 Dual Motor Driver	Upper deck of Alex, secured with Velcro for easy accessibility
Power Bank	Provides power to the Raspberry Pi and the Power Measurement Device	Upper deck of Alex
UT658 Power Measurement Device	Measures the amount of power being used by the Raspberry Pi	Plugged into the USB Port of the Raspberry Pi
Arduino Uno	Receives instructions from Raspberry Pi and controls the motors when required	Lower deck of Alex, secured with foam double sided tape
DRV8833 Dual Motor Driver	Provides power to drive both motors	Lower deck of Alex, so that the wires connected to it are less vulnerable to disconnections and secured using foam – based double sided tape
2 Wheel Encoders	Measures the amount of time taken for the motor to complete 1 full revolution	Connected to the front of each motor
2 Motors	Receives instruction from Arduino and rotates accordingly	Placed as wide as possible to increase stability of Alex when it turns
2 Wheels	Allows the motor to move based on the rotation of the motor it is attached to	Connected to the motors

Table 1: Details of the individual parts of the robot

Section 5 Firmware Design

There is a high-level algorithm used on the Arduino and a communication protocol being followed to communicate between the Pi and Arduino. Figure 6 shows the algorithm for when the Arduino receives a command packet from the Pi. Section 6 will further explain when this function is called in the overall algorithm.

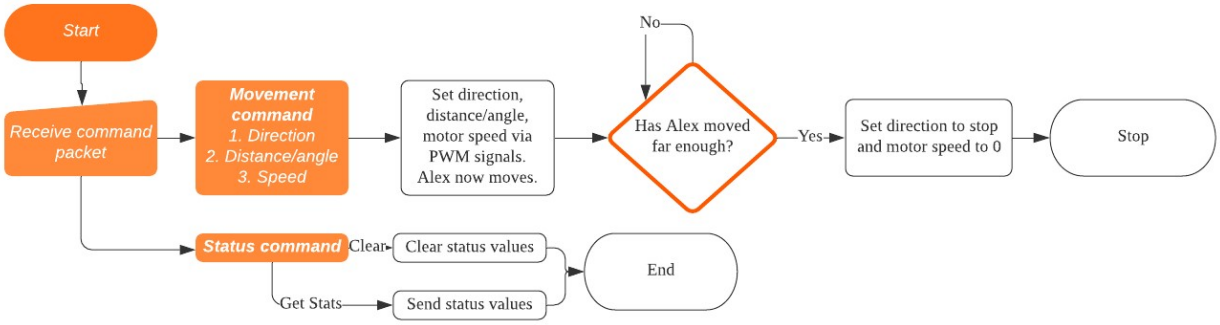


Figure 6: Arduino receiving Command Packet

Interrupts are used to calculate the distance moved or angle rotated using the help of the wheel encoders to guide the Arduino in precisely controlling the motors. When the distance or angle is set, a number of ticks is calculated by the Arduino. An interrupt occurs when the wheel moves a certain distance, and each interrupt provides a tick which is added to a tick counter. This allows the Arduino to stop the motors appropriately when enough ticks have been reached.

The main purpose of the Arduino is to execute the commands sent over from the PC to the Pi. The Pi and Arduino are physically connected via USB, and all communication between them will pass through this connection. Their bit-level communication has a baud rate of 57600, with data length of **8** bits, **N**o parity bits, and **1** stop bit – 8N1. Table 2 shows the structure of a packet sent and received by Arduino and Pi – labelled TPacket.

Packet variable	Description
char packetType	Stores type of data packet holds (see packet ID)
char command	Stores command (if any)
char dummy[2]	Padding to ensure reliable packet size
char data[32]	Stores string data (if any)
uint32_t params[16]	Stores parameters as 32-bit integers (if any)

Table 2: Packet structure

Packet types received and sent have their own ID. The packet types correspond to the packetType variable in the main TPacket. This allows an appropriate action to be taken for each type of packet received. The following table summarizes the different Packet Types and their IDs.

Packet Type	ID assigned
PACKET_TYPE_COMMAND	0
PACKET_TYPE_RESPONSE	1
PACKET_TYPE_ERROR	2
PACKET_TYPE_MESSAGE	3
PACKET_TYPE_HELLO	4

Table 3: Packet types and their assigned ID

Furthermore, Arduino only sends data to Pi when prompted, such as when the Pi asks for stats or sends a command. Since serial devices can only deal with streams of bytes, the packets mentioned earlier must be serialized into one. This is done by copying the structure into an array of char data type, which is sent over the serial port. The receiving end will then have to deserialize the data accordingly. To initialize serial communication for environment mapping, the Pi will send a hello packet to the Arduino and receive one back in return. Subsequent tasks are then detailed in the next Section.

Section 6 Software Design

Environment mapping is an important task carried out by Alex which helps it to navigate through unknown terrains. Figure 7 below shows a flowchart describing how Alex carries out this task.

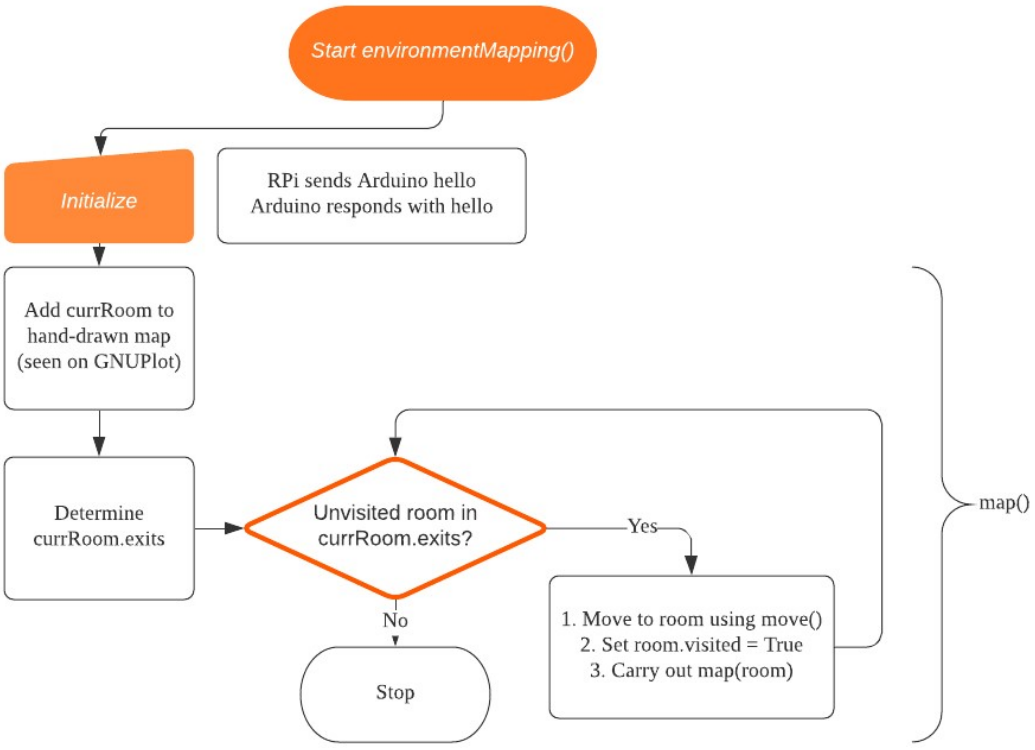


Figure 7: environmentMapping() function

- ❏ `currRoom` – current room Alex is in
- ❏ `currRoom.exits` – unvisited areas from `currRoom`
- ❏ `room.visited` – Boolean as to whether `room` has been visited

High Level steps:

1. Initialization
2. Determine exits
3. Receive user command
4. Carry out command
5. Repeat step 2 to 4 until navigation is over

As seen in Figure 7 above, the high-level environment-mapping algorithm used by Alex has a single loop, with a portion of the algorithm being recursively called until the algorithm terminates, when all rooms have been visited. While the actual map may not be feasibly split into perfectly shaped rooms, the user can use their judgement in deciding whether there are any unvisited areas in the map.

The movement command is entered by the user in the terminal of Pi, whereby it uses the communication protocol and high-level algorithm explained in Section 5 to send that data to Arduino and subsequently move Alex. The Pi will receive an acknowledgement when the Arduino receives the message. An example of such a command is shown in Figure 8 below.

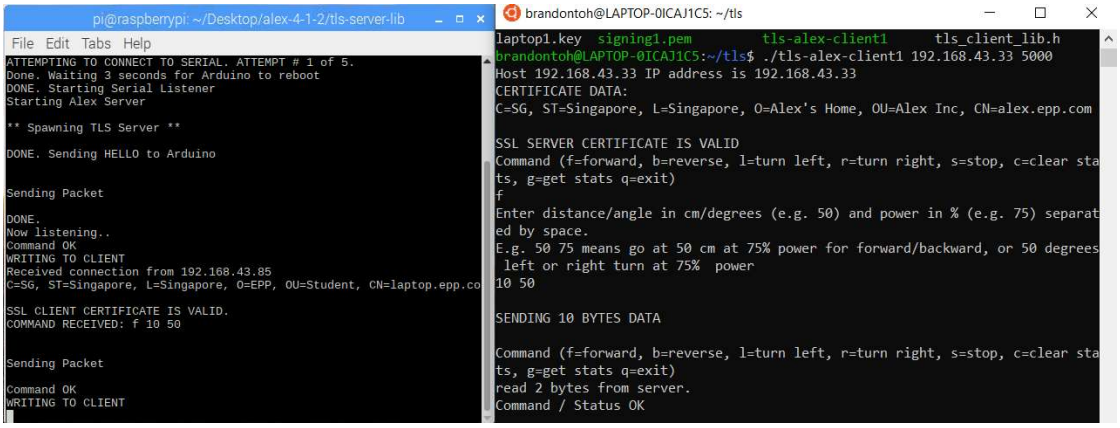


Figure 8: Terminal showing movement command

Moreover, gnuplot is used to determine the area surrounding Alex. The LiDAR scans the 360-degree surroundings around Alex and plots a graph as shown in Figure 9 below. To further assist the user, concentric circles are drawn on the map which show the distance from the obstacles around the LiDAR. The user can note down the surroundings and move Alex appropriately.

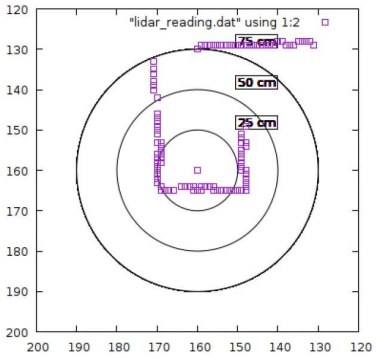


Figure 9: gnuplot Application

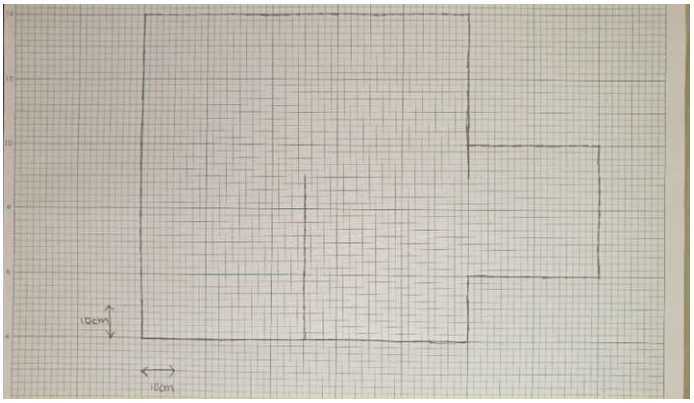


Figure 10: Hand-drawn map using gnuplot

As seen in Figure 10 above, the map is drawn out with the use of gnuplot to obtain the final environment mapping using the algorithm mentioned earlier. To further assist the user in teleoperating Alex, the Raspberry Pi can send a “Get Stats” command to the Arduino to obtain a status report as shown in Figure 11 below. This helps the user to determine how far the Arduino has travelled and how much each wheel has rotated, so that the user may adjust accordingly.

```
g
Command (f=forward, b=reverse, l=turn left, r=turn right, s=stop, c=clear stats, g=get stats q=exit
)

 ------- ALEX STATUS REPORT -------

Left Forward Ticks:             197
Right Forward Ticks:            210
Left Reverse Ticks:             0
Right Reverse Ticks:            0
Left Forward Ticks Turns:       0
Right Forward Ticks Turns:      0
Left Reverse Ticks Turns:       0
Right Reverse Ticks Turns:      0
Forward Distance:               12
Reverse Distance:               0

--------------------------------------
```
Figure 11: Terminal showing the status report of Alex

TLS Server

To enable communication between the user and Alex directly, a server on the Pi and a client on the user's laptop is created. However, several steps must be taken to ensure secure communication. The steps are categorized into two parts - the generating and signing of keys. Both the client and server would need to have signed certificates signifying that the connection between them is authorized, using their private keys to get the certificates. In a real scenario, there would be a Certificate Authority (CA) doing so, but for Alex, the user represents this authority.

With the help of the keys, certificates and functions (createServer(), createClient()), a secure connection is established by having RPi constantly listening to a server port for a new connection and a laptop connecting to the same port to read and write to the server. Figure 8 above showcases the TLS client-server setup.

Section 7 Power Management

Notably, Alex needs to manage its power efficiently so that it can last as long as possible in any mission. Thus, the following section details the power management measures taken by Alex. In order to save power, timers and peripherals not utilized were switched off on the Arduino with the help of the ATmega328P datasheet [8], using the Power Reduction Register (PRR). When Alex is not moving and no commands are detected, Arduino will go into idle mode, saving power, turning back to normal mode when a command is received. The functions WDT_off(), setupPowerSaving(), putArduinoToIdle() in Alex.ino contain the code that executes these power saving measures.

The Watchdog Timer (WDT), Timer0, Timer1, Timer2 and the clock to the module were switched off by modifying the following registers: MCU Status Register (MCUSR), Watchdog Timer Control Register (WDTCSR) and PRR.

WDT is a timer counting the cycles of a separate on-chip 128kHz oscillator that gives an interrupt when the counter reaches a given time-out value. The timer is reset (`MCUSR &= 0b00000111`) and the old pre scaler setting is kept (`WDTCSR |= 0b00011000`) before turning off the timer (`WDTCSR &= 0x00`). Details of how bits in MCUSR and WDTCSR are allocated is shown in Figure 12 & 13 below respectively.

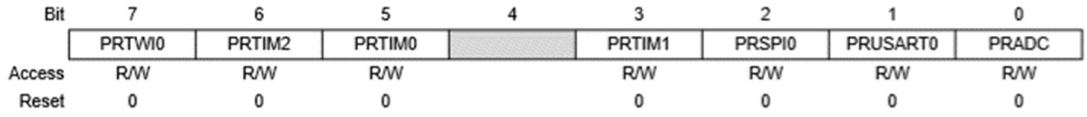
Bit	7	6	5	4	3	2	1	0
					WDRF	BORF	EXTRF	PORF
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Figure 12: Bits allocation in MCUSR [8]

Bit	7	6	5	4	3	2	1	0
	WDIF	WDIE	WDP[3]	WDCE	WDE	WDP[2:0]		
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Figure 13: Bits allocation in WDTCSR [8]

Timer0, Timer1, Timer2 and the clock to the module allow us to run different operations at different rates. These, along with the ADC module can be switched off by writing `PRR |= 0b11101001`. Details of how each bit in PRR is allocated is shown in Figure 14 below.



Bit	7	6	5	4	3	2	1	0
	PRTWI0	PRTIM2	PRTIM0		PRTIM1	PRSPI0	PRUSART0	PRADC
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

Figure 14: Bits allocation in PRR [8]

The idle mode on the Arduino is done by setting logic 1 to SE in SMCR to enable sleep mode and using “sleep_cpu()” function to put the MCU into sleep, waking it up again by setting logic 0 to SE in SMCR.

In addition to the timers, clock and Arduino mode, peripheral ports were also switched off. By entering appropriate commands in the terminal and editing configuration files, the HDMI port, Ethernet port, LED and Bluetooth [9] were also successfully switched off. This is further elaborated below in Table 4.

Peripheral	**Action done**
HDMI Port	In the terminal, use command `/usr/bin/tvservice -o`
Ethernet port	In the terminal, use command `/echo '1-1' | sudo tee /sys/bus/usb/drivers/usb/unbind`
LED Bluetooth	In the file `/boot/config.txt` , the following lines were added: o `dtoverlay=pi3-disable-bt` o `dtparam=act_led_trigger=none` o `dtparam=act_led_activelow=on`

Table 4: Peripherals switched off for power management

Section 8 Conclusion

There are many lessons that we learnt as a group and some mistakes were made which together helped us grow. This section summarizes the two greatest lessons we learnt and our two biggest mistakes.

First and foremost, we learnt that time management is crucial. Although we planned a robust time schedule, we did not adhere to it since we focused too much time fixing the minor and less important aspects of the robot. As a result, we had less time for the basic functionalities.

Moreover, we learnt the importance of teamwork and how it should be adaptable to changing situations. With the coronavirus situation limiting our face-to-face meetups, and the limitation of only one person keeping the robot, we had to coordinate our ideas and codes to continue developing the robot beyond the lab sessions and still meet our final deadlines. As a result, we placed greater emphasis on online platforms such as GitHub and Asana.

One major mistake that we felt we made was not carefully planning the placement of the parts of the robot before assembling them. We placed the battery pack holder in bottom deck of the robot, which prevented us from removing the batteries easily to recharge them. As a result, we wasted a lot of time restructuring the design of the robot.

In addition, we felt that we did not spend time on innovation or improvements. This happened because we closely followed the rubrics and did not think out of the box. After looking at other robots and their improvements, we realized we could have devoted some time to make our robot a little different and advanced.

References

[1] R. R. Murphy, "Activities of the Rescue Robots at the World Trade Center from 11–21 September 2001," *IEEE Robotics and Automation Magazine,* pp. 50-61, September 2004.

[2] E. Guizzo, "Japan Earthquake: More Robots to the Rescue," IEEE Spectrum, 18 March 2011. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/industrial-robots/japan-earthquake-more-robots-to-the-rescue. [Accessed 5 April 2020].

[3] T. Williams and C. Kelley, "gnuplot homepage," gnuplot, April 2020. [Online]. Available: http://www.gnuplot.info/. [Accessed April 2020].

[4] "WALK-MAN: Whole-body Adaptive Locomotion and Manipulation," WALK-MAN, 2018. [Online]. Available: https://www.walk-man.eu/. [Accessed 3 March 2020].

[5] E. Guizzo and E. Ackerman, "WALK-MAN Team Built Brand New, Highly Custom Robot for DRC Finals," IEEE Spectrum, 3 June 2015. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/humanoids/walkman-humanoid-robot-iit. [Accessed 13 March 2020].

[6] P. U. Lima, "RAPOSA NG – a search and rescue land wheeled robot," SPARC, 25 May 2014. [Online]. Available: https://www.eu-robotics.net/sparc/success-stories/raposa-ng-a-search-and-rescue-land-wheeled-robot.html?changelang=2. [Accessed 13 March 2020].

[7] P. U. Lima, J. Frazão, I. Ribeiro and R. Ventura, "RAPOSA: Semi-autonomous robot for rescue operations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, 2006.

[8] Atmel Corporation, "ATmega328P Datasheet," 2015. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. [Accessed March 2020].

[9] C. Rush, "How to save power on your Raspberry Pi," Pi Supply Maker Zone, 26 March 2019. [Online]. Available: https://learn.pi-supply.com/make/how-to-save-power-on-your-raspberry-pi/. [Accessed 22 March 2020].