



CG2111A Engineering Principle and Practice
Semester 2 2023/2024

“Alex to the Rescue”
Final Report
Team: B03-6B

Name	Student #	Main Role
Charly Chandra	A0281413M	Arduino, Electrical
Liang Kuan Hsien	A0286416X	Arduino, Electrical
Li Shunyang	A0288428M	Software, 3D Design and Printing
Ramdas Om Jayant	A0276054H	Software, TLS, ROS, Linux

Table of Contents

1. Introduction	2
2. Review of State of the Art	3
2.1 Spot by Boston Dynamics [1]	3
2.2 Guardian S by Sarcos Robotics [2]	4
2.3 Strength and Weakness assessment of Robot Platforms	4
3 System Architecture	5
4. Hardware Design	6
4.1 Hardware Design Considerations	6
4.2 Hardware Functionalities	6
4.2.1 TCS3200 Colour Sensor [5]	6
4.2.2 HC-SR04 Ultrasonic Sensor [6]	7
4.2.3 Speaker	7
4.2.4 LED Module	7
4.2.5 3D printed Mounts	7
Section 5 Firmware Design	8
5.1 High level algorithm in Arduino Mega	8
5.2 Communication Protocol	8
5.2.1 Message Format	9
5.3 Firmware Software Design	9
5.3.1 Integrating LED and colour sensor	9
5.3.2 Integrating Speaker and colour sensor	9
5.3.3 Ultrasonic Sensor	9
5.4 Object-Oriented Programming (OOP) Firmware Code	9
6 Software Design	10
6.1 Overall Algorithm	10
6.2 Initialisation	10
6.2.1 Additional Code Implementation: Setup bash script	11
6.3 Use of ROS for Environment Mapping	11
6.4 Teleoperation: Commands Key Mapping	12
6.5 Teleoperation: Transmission and Execution of Command	13
6.5.1 Colour Sense Command	13
6.6 Additional Code Implementation: Use of Transformation Frames (TF)	13
7 Lessons Learnt - Conclusion	14
7.1.1 Lesson 1: Task Delegation and Proper Execution	14
7.1.2 Lesson 2: Implementing iterative development and regression testing	14
7.2.1 Mistake 1: Postponing Plans	14
7.2.2 Mistake 2: Poor Github Control	14
Reference	15
Annex A Raspberry Pi Architecture	16
Annex B System Architecture	17
Annex C packetType	19
Annex D Command Types	19
Annex E Response Types	20
Annex F OOP Class Hierarchy	20

1. Introduction

In this project, Alex to the Rescue, we have been specified to design and make a robot named “Alex”. Its primary objective is to navigate and map an unknown terrain and submit the map of it. Its secondary objective is to search for casualties amidst a mixture of dummies. Alex is given six minutes to complete the maze and its components and is required to park at the end to signify the completion of its mission. To ensure the safety of the victims and Alex itself, it should not collide with any objects.

Alex will be tele-operated from a laptop (PC) communicating with Alex's brain, the Raspberry Raspi (Raspi), via a secure TLS TCP/IP transmission layer. The operator must rely on Alex and its peripheral hardware to identify and locate the casualties and dummies. In its arsenal Alex has been equipped with a LiDAR (Light Detection and Ranging) to enable it to map out the maze and locate the casualties. We identify and determine between casualties (Red or Green) and dummies (White) using an onboard colour sensor to analyse their colour.

2. Review of State of the Art

2.1 Spot by Boston Dynamics [1]



Figure 1. Spot by Boston Dynamics

(i) Functionalities

Spot is designed for various applications other than search and rescue such as inspection, data collection and research. It can perform simple tasks like opening doors and manipulating objects with its versatile robotic arm attachment.

(ii) Hardware

It is built with a four-legged design which is more versatile than regular wheel-based robots since it can move more freely in environments which have irregular surfaces or various obstacles. Thus, this allows it to navigate through complex terrain. The hardware consists of several sensors such as cameras, LiDAR, Inertial Measurement Unit and depth sensors for perception and navigation.

(iii) Software

It has proprietary algorithms for perception, navigation and control. Furthermore, with its software, it has autonomous navigation, obstacle avoidance and the ability to interact with its environment.

2.2 Guardian S by Sarcos Robotics [2]



Figure 2. Guardian S by Sarcos Robotics

(i) Functionalities

It is a cloud-connected mobile IoT (Internet of Things) and sensor platform that provides inspection and surveillance capabilities to augment human-based inspections in challenging environments.

(ii) Hardware

It is equipped with a built-in LTE modem for future cloud-based services (700 Mhz). It can encrypt audio signals and video streams allowing two-way real time video, voice and data communication. It is also IP65 certified which makes it water protected.

(iii) Software

It implements cloud computing functionality and incorporates all aspects of robot control, data gathering and analysis in an easy-to-use cloud-based platform.

2.3 Strength and Weakness assessment of Robot Platforms

Robot Platform	Strengths	Weaknesses
Spot	Able to navigate through complex terrains due to its four-legged design and enhanced mobility	Unable to lift heavy debris or obstacles as it does not come with a built-in arm
Guardian S	Able to lift obstacles of around 4 kg	Small in size and thus not be able to go over large obstacles

3 System Architecture

All code regarding our Firmware and Software design can be viewed at the following public repository: <https://github.com/Charly2312/CG2111/tree/main>.

Alex combines the following 9 components: Arduino Mega (Mega), Raspi , LiDAR, Motors, Wheel Encoders, Colour Sensor, Ultrasonic Sensor, Speaker and PC. The PC will be running Ubuntu Focal Fossa 20.04.6 while the Raspi will be running Debian 10 Buster, both will be running ROS Noetic. The TLS Client will be set up on our PC while the TLS Server will be on Alex's Raspi.

Unified Modeling Language (UML) diagram below shows the system architecture and how all these components and programs are connected and interact with each other.

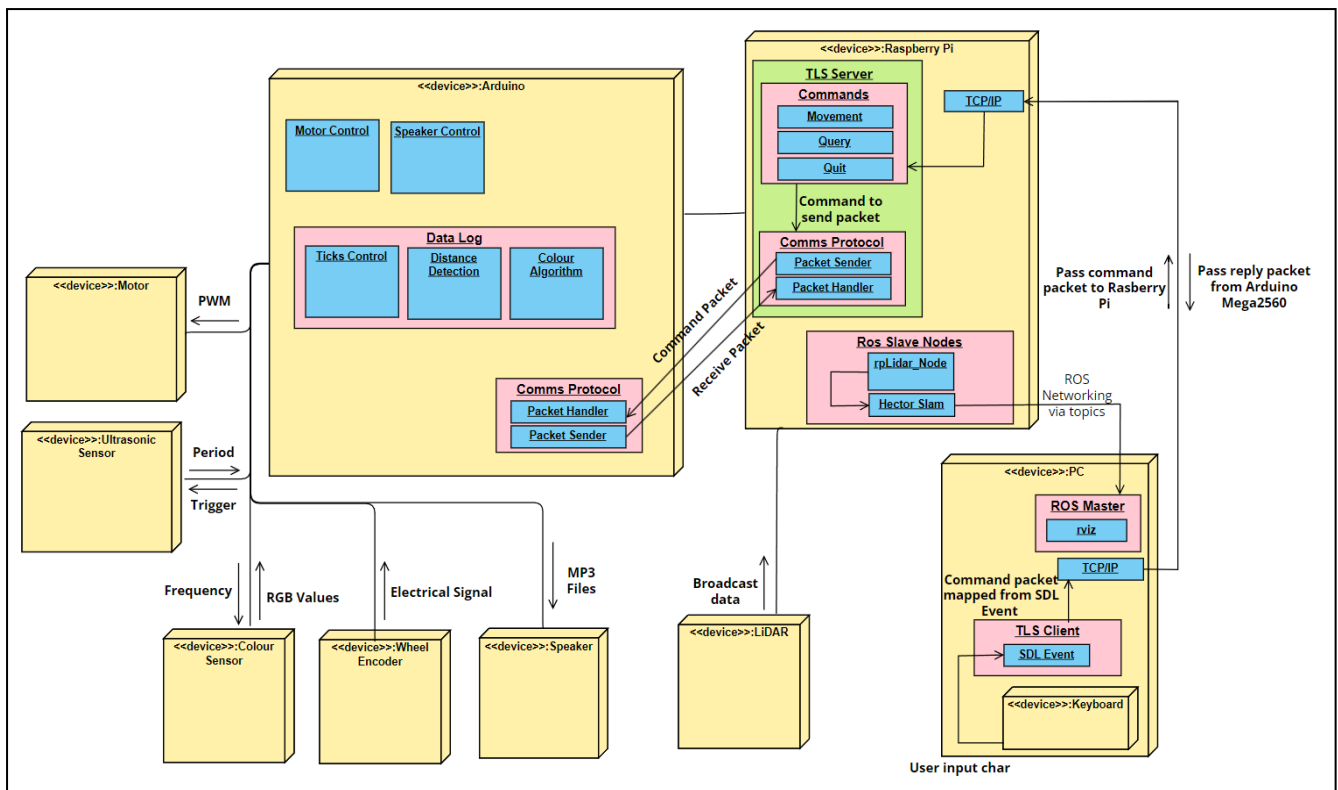


Figure 3: UML Diagram for Alex Robot

4. Hardware Design

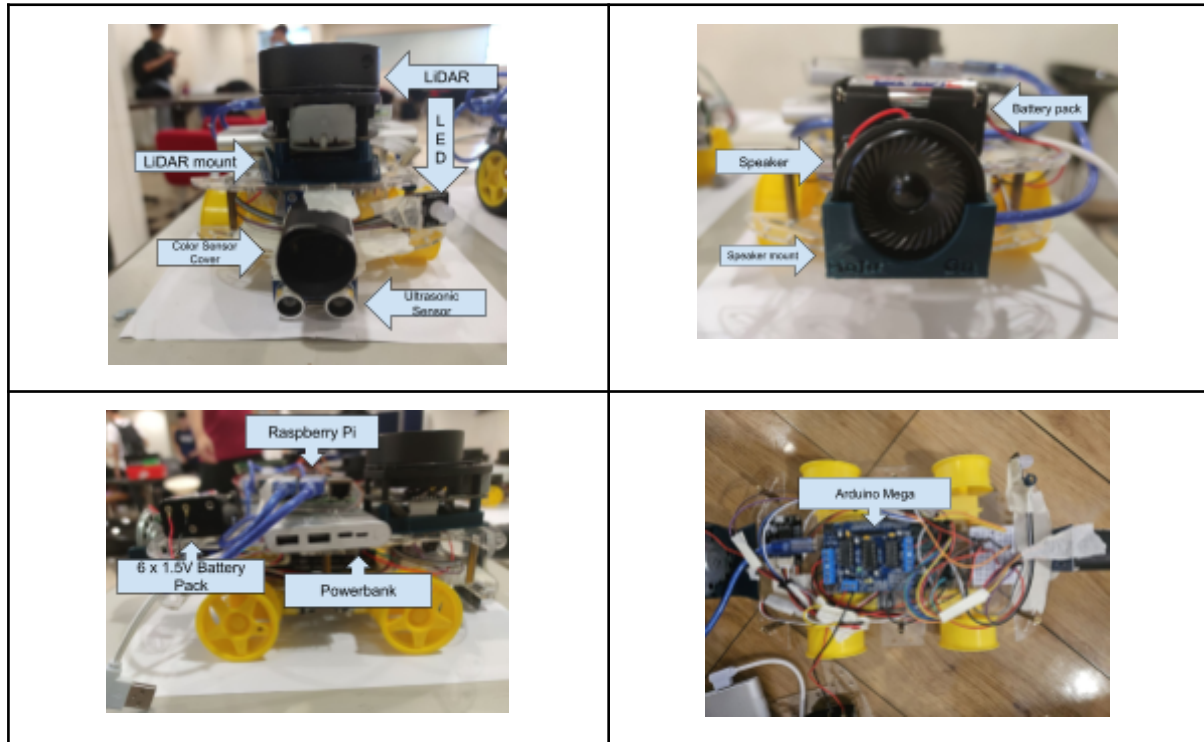


Figure 4: Alex Photoshoot, in order of Front, Back, Side and Inside View

Alex has a 4-wheel drive configuration with an acrylic chassis. For power Alex Utilised a $6 \times 1.5V$ battery pack to power its Motors and another $3.7V$ $10000mAh$ power bank to power the Raspi (refer to Annex A) and the Mega (refer to Annex B). Wheel encoders [3] were used to control the distance Alex moved whilst providing odometry data to the tele-operator, however with our software implementation discussed in the later section 6, the encoders were a fallback to stop Alex after moving a set distance.

4.1 Hardware Design Considerations

Due to the weight of the components and the fixed motor configuration, Alex requires considerably more power and torque, approximately double compared to straight driving, for the motors to conduct a point turn as there is a need to overcome the high lateral resistance forces during skid steering due to a greater sideslip angle (Benjamin Shah, 1999). Therefore we removed the tires to reduce friction and enable Alex to accomplish the same turning radius whilst reducing the power and torque required of the motors.

4.2 Hardware Functionalities

4.2.1 TCS3200 Colour Sensor [5]

Using a sensitive photodiode attached in the middle with four small LEDs. The LEDs will shine white light on the object and each photodiode will then receive a frequency value for red, green or blue. The frequency values can be obtained from reading from the photodiode by changing the control pins S2 and S3 via changing bits 3 and 4 in PORTC to either HIGH or LOW.

The sensor will use a current-to-frequency converter to convert the readings from the photodiode into a square wave with the frequency directly proportional to the light intensity. We then calibrate the colour sensor using a white and black paper to map the frequencies to RGB values of 0 to 255 while fixing the distance between the colour sensor and the object to 10cm..

We implemented a Colour Sensor Cover made from black paper to increase the accuracy of its readings by preventing the light from the surroundings LEDs and light reflecting off the environment from affecting the readings.

4.2.2 HC-SR04 Ultrasonic Sensor [6]

We mounted the sensor at the front of Alex in line with the colour sensor, this was done to measure the distance between Alex and the object when it is trying to colour sense. This ensures Alex does not collide with the wall or victim whilst increasing the accuracy of the colour detection. To maximise detection we had to hit the 10cm distance from the object based on our calibration.

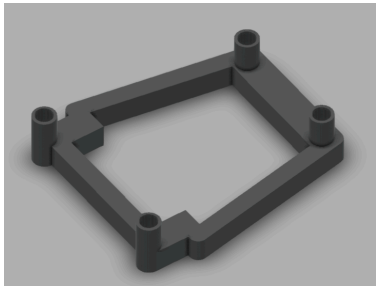
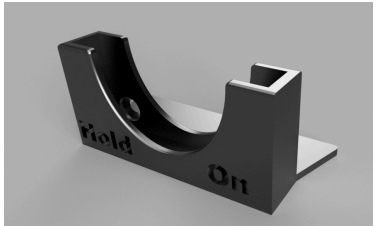
4.2.3 Speaker

We mounted a speaker at the back of Alex which voices out the identity of the object depending on what type of victim (Red or Green) or if it is a dummy (White). The MP3 files have been attached to our github.

4.2.4 LED Module

For aesthetic purposes our LED will shine the colour of the object that it has detected (Red, Green or White)

4.2.5 3D printed Mounts

LiDAR Mount		Ensures no object blocks LiDAR scan and holds the LiDAR more firmly to the platform
Speaker Mount		To provide secure space to hold the speaker whilst ensuring a clear sound

Section 5 Firmware Design

5.1 High level algorithm in Arduino Mega

The following diagram describes the communication between the Mega and the Raspi.

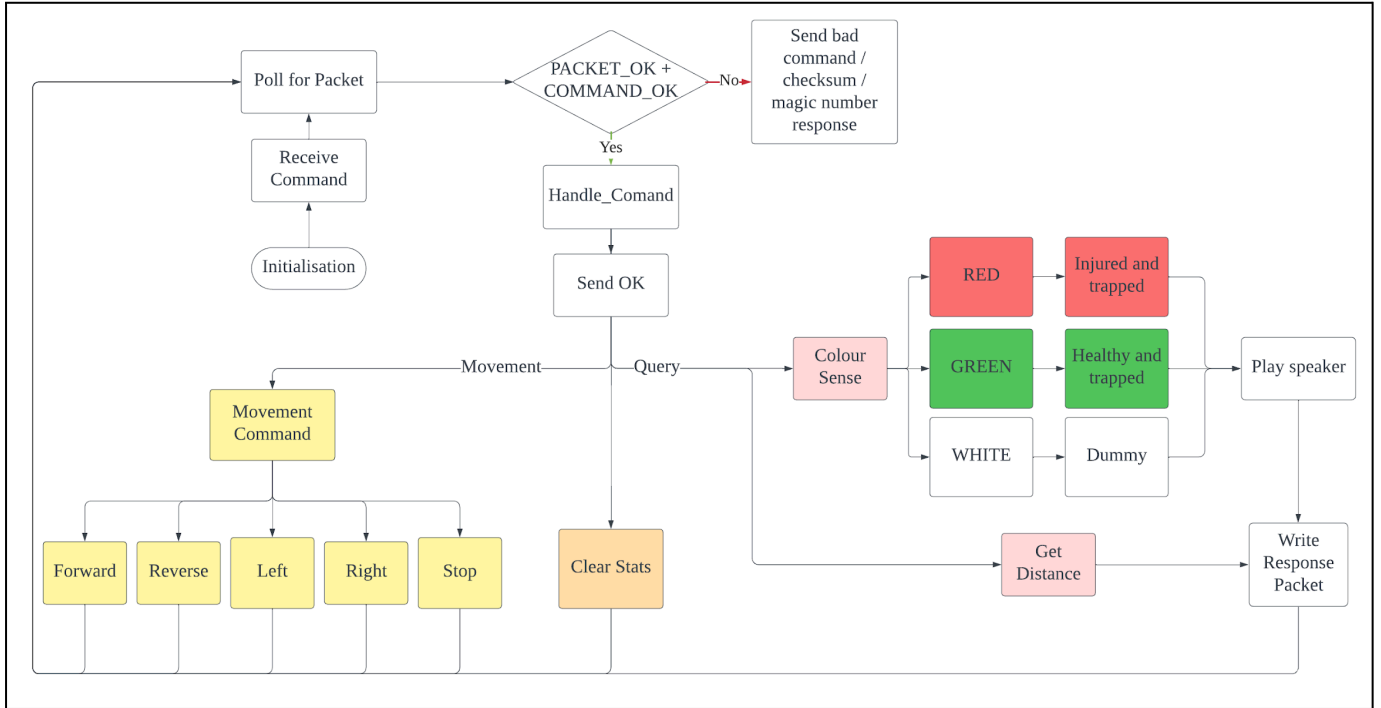


Figure 5: Flowchart for Mega

5.2 Communication Protocol

Communication between the PC and Raspi is done using a TLS protocol on top of a TCP/IP transmission layer over a wireless network with our phones acting as a mobile router to connect Alex to the PC.

The Mega and Raspi communicate asynchronously via UART through the standard 8N1 frame format with a baud rate of 9600 bps. In the Mega this is done with bare-metal programming by initialising UCSR0A to 0, setting UCSZ0([0:2]) to 011 and the UBRR value to 103 (for 9600 bps). To start the serial, UCSR0B is set to 0b00011000 to enable RXEN0 and TXEN0 which enable the receiver and transmitter respectively, while setting UCSZ02 to 0. This sets serial communication through polling method.

In this process, data is read through the readSerial function which fills in a buffer array from bit 0 with UDR0 when RXC0 in UCSR0A is set to 1. For transmitting data through writeSerial function, a packet is serialized into a buffer which will be written into UDR0 sequentially from bit 0 when UDRE0 in UCSR0A is cleared which is when it is set to 1.

5.2.1 Message Format

In the packet the Mega receives, it consists of a char of packetType and another char of command along with other parameters such as distance/angle and speed which we have kept fixed. Each packetType (refer to Annex C) and command (refer to Annex D) has their own unique number code which will then be used to distinguish them and how they should be executed. Afterwards, the Mega will carry out the command and return its respective responses (refer to Annex E) based on what it read from the packet sent by Raspi.

5.3 Firmware Software Design

5.3.1 Integrating LED and colour sensor

The colour sensor was set up through bare metal programming to set S0 to S3 pins as outputs and setting the frequency scaling to 20% through setting bit 5 as 1 and clearing bit 6 of PORTC to 0. To read the PWM values of red, green and blue from the photodiode we set S2 and S3 to either HIGH or LOW. For the LED module, there are three functions to shine different colours which are shineWhite(), shineRed() and shineGreen(), which shines their respective colour. In the Mega, there is a 'checkColor' function which decodes the colour of the victim. Once the colour is identified by the 'if' conditions, it will call and execute either of the three functions to shine the LED. The ratio is calculated using the following formula: $nRatio = (nValue)/(red + green + blue)$, where n is the colour we want to obtain the ratio for.

- White : $rRatio < 0.4$, $gRatio < 0.4$ and $bRatio < 0.4$
- Red : $rRatio \geq 0.39$, $rRatio > gRatio$ and $rRatio > bRatio$
- Green : $rRatio < 0.5$, $gRatio > bRatio$ and $gRatio > rRatio$

5.3.2 Integrating Speaker and colour sensor

Using the MP3 library from Kuriosity [7], we play a unique track for each colour. This is done similar to 5.3.1, after the colour is identified, a unique track is played using the function 'mp3.play_track(x)' where x is the track number at the volume of 30 using 'mp3.volume(30)'.

5.3.3 Ultrasonic Sensor

We implemented a getDistance() function which would return the uint32_t value of the distance. We used bare-metal programming to set up the TRIG and ECHO pins which are PD0 and PD1 respectively, This function will be called when we press 'G' (get stats) or 'R' (get Distance) and the statusPacket.params will be updated with the distance and sent to Raspi where the operator can read in the value from the PC.

5.4 Object-Oriented Programming (OOP) Firmware Code

We attempted to make an OOP version of our Mega code in our Github. The use of OOP greatly enhances the readability and maintainability of the code. We have set up four header files to manage the class declarations of the four peripherals: colour sensor, LED, MP3, and ultrasonic sensor. The implementations of the class functions are located in their respective .cpp files. Class Hierarchy (refer to Annex F) has been set up for our OOP code in an attempt for easier debugging and readability of the code.

6 Software Design

6.1 Overall Algorithm

Many blocks and loops have been omitted for readability sake.

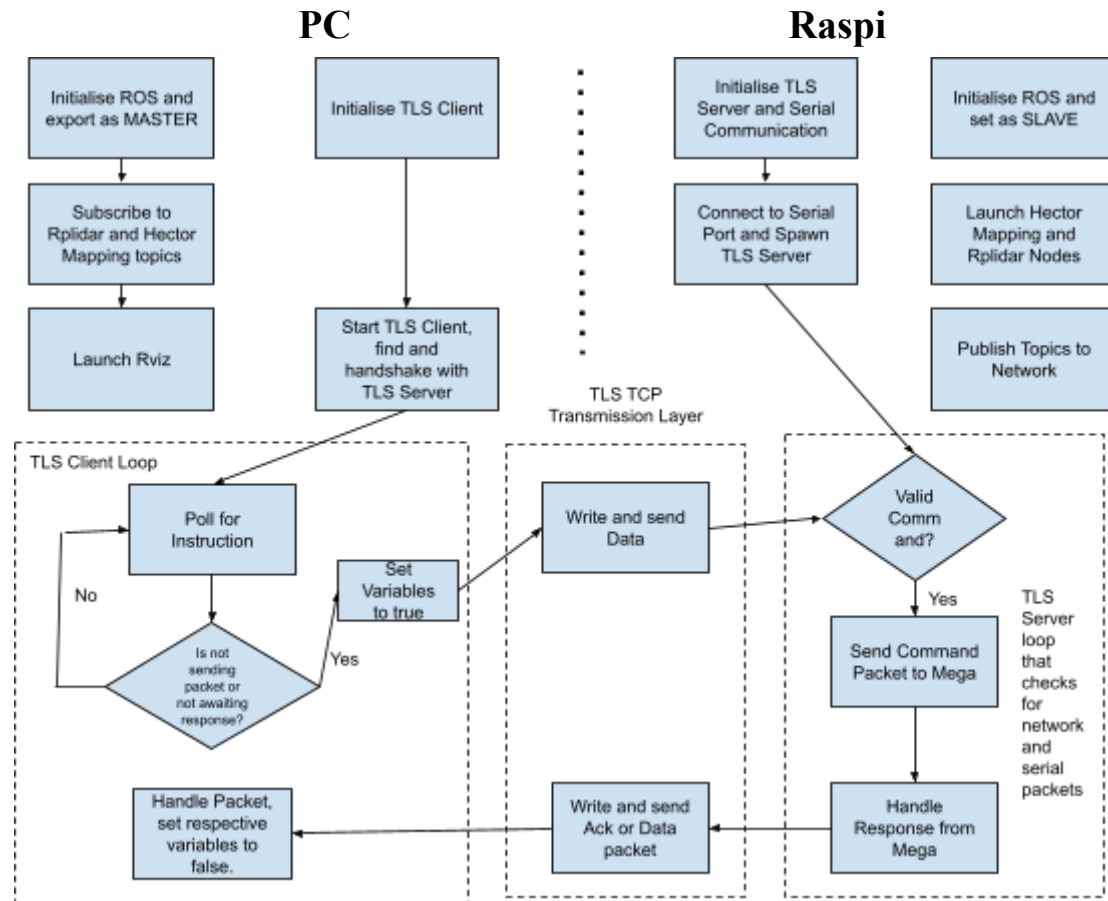


Figure 6: Overall Algorithm for Raspi and PC

6.2 Initialisation

We follow this sequence of steps to initialise Alex:

1. Initialise ROS Master on PC
2. Initialise RpLiDAR and Hector mapping Slave nodes on the Raspi
3. Start TLS Server on Raspi
4. Raspi will first initiate a link with Mega by sending a “Start Search” command.
5. In response, the Mega will reply with “Ready to Search” back to Raspi.
6. Initialise Rviz on PC
7. Start TLS Client on PC

Once communication between PC to Raspi to Mega is successful, Alex is ready to start navigating through the terrain.

6.2.1 Additional Code Implementation: Setup bash script

Our group noted that the setup of Alex was time consuming, roughly taking 3 minutes to navigate and execute the necessary tasks on the PC and Raspi in order and accurately. Thus we created a bash setup script (Fig 7) to automate the entire initialisation process and reduce the chance of human error.

```
#!/bin/bash
echo 'Make sure IP addys have been added to hosts'
read -p "Enter your Raspi IP Addy: " raspi_ip
read -p "Enter your laptop IP Addy: " laptop_ip
echo $raspi_ip $laptop_ip
Port=5001

source ~/cg2111a/devel/setup.bash
export ROS_MASTER_URI=http://$laptop_ip:11311
export ROS_IP=$laptop_ip

roscore &

sleep 5
gnome-terminal -- bash -c "ssh -t pi@$raspi_ip './rplidar-setup.sh $laptop_ip $raspi_ip; bash'"
gnome-terminal -- bash -c "ssh -t pi@$raspi_ip 'cd Alex-main/TLS_server/;./tls-alex-server2; bash'"
sleep 5
gnome-terminal -- bash -c "source ~/cg2111a/devel/setup.bash;roslaunch rplidar_ros view_slam.launch; exec bash"
sleep 2
gnome-terminal -- bash -c "cd TLS_client;./tls-alex-client-final $raspi_ip $Port;exec bash"
wait
```

Figure 7: Bash Setup Script on PC `./setup.sh`

```
#!/bin/bash

#Variables
USERNAME="pi"
PI_IP="172.20.10.11"

#Setup ROS
source /opt/ros/noetic/setup.bash
source ~/cg2111a/devel/setup.bash
export ROS_MASTER_URI=http://$1:11311
export ROS_HOSTNAME=$2

roslaunch rplidar_ros start_nodes.launch &
```

Figure 8: Bash Setup Script on Raspi `./rplidar-setup.sh`

We added sleep statements between instructions to ensure that there was a buffer to monitor the processes as well as to give enough time for the TLS server to connect to the Mega's serial port and spawn the TLS server before the TLS client was created. The script calls on another script on the Raspi (Fig 8) to start the LiDAR and Hector Mapping nodes. Now the total time to set up Alex has been cut down to approximately 20 seconds.

6.3 Use of ROS for Environment Mapping

The LiDAR and Hector Mapping nodes will continuously obtain and communicate the LiDAR and SLAM (Simultaneous Localization And Mapping) data back to the PC over ROS networking via their respective topics. The Rviz running on the PC will subscribe to these topics and will compute the visualisation of the map. Thus we have moved the computational task of generating the map to the PC, reducing the strain on the Raspi.

6.4 Teleoperation: Commands Key Mapping

The tele-operation of Alex is accomplished using the keyboard on the PC leveraging the SDL2 library [8]. The client will create an SDL instance which monitors and polls the keyboard input which subsequently creates an SDL event which we then map and send the corresponding command. We added this functionality as we decided the vanilla manual control was not only slow but inaccurate, especially for turning and other manoeuvres. Whilst this method has issues with network latency, where if the network transmission speed is slow the responsiveness of Alex will also be noticeably slow and sluggish, it comes with the benefit where it is intuitive to control as anyone who has played a WASD movement game. Additionally, the implementation of SDL also enabled us to detect when the instance of the key was no longer being pressed, allowing us to send one final stop command.

We utilised the following Commands:

Command s (Key)	Command Type	Command Description	Remarks
W “Move Forward”	Movement	Move forward by preset distance and speed	Keep executing the move command while the keyboard key is pressed. Client will transmit the Stop command once it detects that the key is being released.
A “Move Left”	Movement	Move left by preset distance and speed	
S “Move Backward”	Movement	Move backwards by preset distance and speed	
D “Move Right”	Movement	Move right by preset distance and speed	
E “Stop”	Movement	Stop	
Q “Quit”	Program	Quit Program	
R “Get Distance”	Communication	Get Stats + Distance	Send updated values of tick counters and distance Created a separate GetDistance function in the TLS Client and Server programs
G “Get Stats”	Communication	Get Stats + Colour + Distance	Send updated values of tick counters + colour code + RGB values + distance
C “Clear Stats”	Communication	Clear Stats	

6.5 Teleoperation: Transmission and Execution of Command

In order to send commands efficiently and accurately, we had to use 2 static volatile bools, “sending_data” and “waiting_for_data”. The former was to prevent bad magic number errors from occurring by the client writing too many packets to the network before the server could respond. Now the client will send a packet and continue polling the most recent command, it will only send the next command once it receives the “Ack” packet back from the Mega over the network. The “waiting_for_data” bool ensured that we do not disrupt the colour detection execution with a new command.

6.5.1 Colour Sense Command

Before calling the colour detection command, the distance of Alex from the victim will be measured. The photodiodes will then read in the reflected frequencies and send them to the Mega where they will be mapped to RGB values to determine the surface colour using the map() function in the Mega and stored in their respective variables in the statusPacket array which will be sent out to Raspi.

The checkColour function compares the ratio of the individual RGB component to the total sum of all the RGB component values in checkColor. With these individual ratios we determine the colour of the object we are measuring through calibration from an optimal distance of 10 cm between the object and the colour sensor. After going through the ‘if’ conditions, based on the colour identified, a uint32_t colour variable will be updated. A Colour code of value ‘1’ means it is white, ‘2’ means it is red and ‘3’ means it is green. This value will be sent to Raspi in the status packet with its respective RGB values.

6.6 Additional Code Implementation: Use of Transformation Frames (TF)

Using TFs created in the Rviz launch files, we modelled Alex using the pose of the LiDAR as our reference. We measured and mapped out the approximate dimensions of Alex onto our Rviz map. We had it so that the front 2 corners corresponded to demarcate the optimal distance (10cm) for our colour sensor to work. So aligning to corners to be on top of the object we wanted to measure and we would then call the coloursense algorithm. The back 2 corners just corresponded to the back of Alex with a little bit more wiggle room. Using these TFs it enabled us to reduce the probability of collisions with walls and other obstacles whilst making manoeuvring around the map easier for the operator.

```
<node pkg="tf2_ros" type="static_transform_publisher" name="alex_tr" args="0.11 -0.1 0 0 0 laser topRight" />
<node pkg="tf2_ros" type="static_transform_publisher" name="alex_tl" args="0.11 0.1 0 0 0 laser topLeft" />
<node pkg="tf2_ros" type="static_transform_publisher" name="alex_br" args="-0.165 -0.1 0 0 0 laser bottomRight" />
<node pkg="tf2_ros" type="static_transform_publisher" name="alex_bl" args="-0.165 0.1 0 0 0 laser bottomLeft" />
```

Figure 9: view_slam.launch for Rviz

7 Lessons Learnt - Conclusion

In conclusion, we would like to thank Prof Ravi as well as our TA Sim Justin for helping us out tremendously with this project. This has been an incredible learning journey and here are some of the lessons and mistakes we learnt along the way.

7.1.1 Lesson 1: Task Delegation and Proper Execution

As a group, we have learned how crucial it is to distribute the workload evenly between each member. This will ensure that no one feels overwhelmed by the project and promote having fun with the process. Consequently, each member will be able to focus on their respective tasks and be more productive at it. Furthermore, the group can also build trust between one another as each member is entrusted with completing their own tasks. Individually, we also feel a greater sense of responsibility and accountability. Overall, this ensures a smooth flow of the team's progress as we try to accomplish the tasks given and make progress along the way.

7.1.2 Lesson 2: Implementing iterative development and regression testing

As the project progressed, our group kept finding new ideas that we wanted to implement for Alex. This helped us to explore our creativity in trying to solve loopholes or gaps in our current code and hardware implementation. While optimising Alex, we also tried to think critically to see if there are any logical faults in our implementation. Through continuous testing and iteration of Alex's design, we built up multiple prototypes along the way. Each prototype was then extensively tested and evaluated with our learning points used in the next iteration. This development methodology helped overcome times in which we were stuck and enabled us to solve some of our overzealous design concepts. Furthermore, when we did implement a new feature we would apply regression testing to ensure it was backwards compatible with the work we had already done.

7.2.1 Mistake 1: Postponing Plans

Since we had a timeline to follow for each week, we had some ideas in mind that we wanted to do. However, there were several circumstances which made us postpone certain ideas to prioritise other issues. For instance, our group wanted to 3D print some mounts to make Alex not only more aesthetically pleasing but also ensure the other hardwares were more secure and stable. However, we faced issues along the way and were bogged down prioritising other implementations such as how to get the speaker to work and optimising the movement of Alex. Consequently, we neglected 3D printing and had to rush it at the last minute. While this did not affect our progress that much, with better planning, we could have designed and built a better Alex.

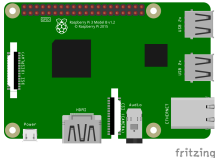
7.2.2 Mistake 2: Poor Github Control

Our group is not super adept at using Github and utilising all the functionalities especially the push and pull system. This thus caused issues where the code uploaded on Github would not run out of the box, unnecessary branches were made or code would be uploaded to the wrong branch. This hindered our workflow and version control which while it did not have any damaging effects, slowed down our productivity when it came to putting Alex together. We could have possibly added more functionalities or ironed out more bugs if we had done so.

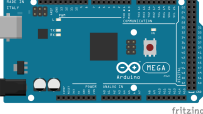
Reference

- 1 Spot by Boston Dynamics:
[Spot Specifications \(bostondynamics.com\)](https://www.bostondynamics.com/spot-specifications)
- 2 Guardian S by Sarcos Robotics:
<https://www.sarcos.com/wp-content/uploads/sarcos-guardian-fact-sheets.pdf>
- 3 Wheel encoders
[Wheel Encoders | Code: Robotics \(idew.org\)](https://www.idew.org/code-robotics/wheel-encoders)
- 4 Benjamin Shamah (1999, March) Experimental Comparison of Skid Steering Vs. Explicit Steering for a Wheeled Mobile Robot
https://www.ri.cmu.edu/pub_files/pub1/shamah_benjamin_1999_1/shamah_benjamin_1999_1.pdf
- 5 TCS3200 Color sensor
<https://lastminuteengineers.com/tcs230-tcs3200-color-sensor-arduino-tutorial/#:~:text=The%20range%20of%20the%20typical,%2C%2020%25%20or%20100%25.>
- 6 Ultrasonic Sensor
<https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>
- 7 MP3 Speaker Library
[GitHub - curiosity-sg/mp3](https://github.com/kuriosity-sg/mp3)
- 8 SDL2 Library
<https://www.libsdl.org/>

Annex A Raspberry Pi Architecture

Central system	Peripheral device		Communication Protocol/Interface
Raspberry Pi 4 	RpLiDAR		UART/Serial
	PC		TCP/IP over Secure Shell protocol
	Xiaomi 10000mAh Power Bank Power delivery		Universal Serial Bus 2.0
	For Testing	Monitor	Mini-HDMI Type C
		Keyboard	USB 2.0/PS/2
		Mouse	USB 2.0/PS/2

Annex B System Architecture

Central system		Peripheral device	Pin specification	Mega GPIO	2560 Pin	Communication Protocol/Interface
Arduino Mega 2560 	Adafruit L2939 Motor Driver shield	9V Battery Pack with 6 AA alkaline cells, 1.5V each)	Power delivery			
		DC Gear Motor * 4 (via H-bridge)	Front left wheel control	M1	Pulse-width-modulated (PWM) speed control by Shield	I2C
			Front right wheel control	M2		
			Back left wheel control	M3		
			Back right Wheel control	M4		
	Arduino GPIO (Mini BreadBoard used for the convenience)	Hall Effect sensors	Trigger interrupt and count ticks of left motor	Digital Pin 18	PD3 -PIN Interrupt source: INT3	IC2
			Trigger interrupt and count ticks of right	Digital Pin 19	PD2-PIN Interrupt source:INT2	
		MP3 Module Speaker	Receive mp3 file	Digital Pin 16	PH1	UART/Seria 1
			Transmit command	Digital Pin 17	PH0	
		HC-SR04 Ultrasonic	TRIG generates	Digital Pin 21	PD0-PORT	I2C

		Sensor	pulse				
			ECHO receives pulse		Digital Pin 20	PD1-PIN	
		RGB 3 Colour LED Module	BLUELED		Digital Pin 46	PL3-PORT	I2C
			REDLED		Digital Pin 45	PL4-PORT	
			GREENLED		Digital Pin 44	PL5-PORT	
		Colour Sensor TCS3200	Select frequency Scaling	S 0	Digital Pin 32	PC5-PIN	I2C
				S 2	Digital Pin 31	PC6-PIN	
			Select	S 3	Digital Pin 33	PC4-PIN	
				S 4	Digital Pin 34	PC-PIN	
			Output frequency		Pin 41	PG0-PORT	

Annex C packetType

Number code	Packet Type
0	PACKET_TYPE_COMMAND
1	PACKET_TYPE_RESPONSE
2	PACKET_TYPE_ERROR
3	PACKET_TYPE_MESSAGE
4	PACKET_TYPE_HELLO

Annex D Command Types

Number code	Response Type	Result
0	COMMAND_FORWARD	Alex moves forward
1	COMMAND_REVERSE	Alex moves backward
2	COMMAND_TURN_LEFT	Alex turns left
3	COMMAND_TURN_RIGHT	Alex turns right
4	COMMAND_STOP	Alex stops any movement or turn
5	COMMAND_GET_STATS	Output tick values, color code, RGB values and distance from object into an array
6	COMMAND_CLEAR_STATS	Execute clearCounter() function
7	COMMAND_GET_DISTANCE	Execute getDistance() function

Annex E Response Types

Number code	Response Type	Result
0	RESP_OK	Send an acknowledgement to Raspi
1	RESP_STATUS	Mega sends tick values, colour code, RGB values and distance from object
2	RESP_BAD_PACKET	Raspi will receive a bad packet warning
3	RESP_BAD_CHECKSUM	Raspi will receive a bad checksum warning
4	RESP_BAD_COMMAND	Raspi will receive a bad command warning
5	RESP_BAD_RESPONSE	Raspi will receive a bad response warning
6	RESP_DISTANCE	Arduino sends an updated distance value

Annex F OOP Class Hierarchy

