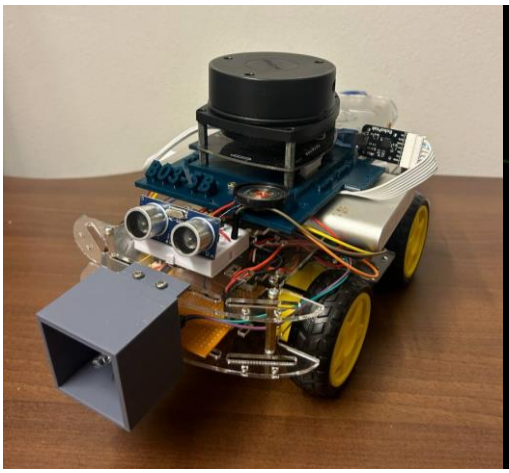
CG2111A Engineering Principle and Practice

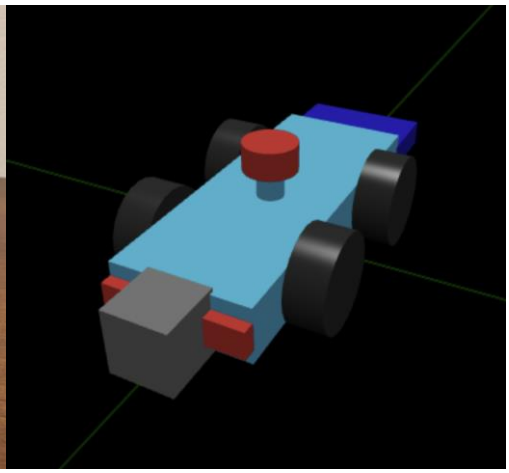
Semester 2 2023/2024

"Alex to the Rescue"

Final Report

Team: B03-5B





Name	Student #	Main Role
Chan Sheng Bin	A0273241M	Design & Assembly Lead ROS/SLAM Lead
Kam Sheng Jie	A0269862L	Serial Communications Lead
Cheng Jia Wei Andy	A0281149B	Bare-metal Programming Lead
Kashfy Ilxilim Bin Zulkarna'in	A0272245H	Colour Sensor & Procurement Lead
Miao Qiaozi	A0282034M	Networking Lead

Table of Contents

1. Introduction

The dire consequences of natural disasters do not stop at just the events themselves. Very often, their effects can be felt long after their occurrence, with victims remaining trapped under debris too dangerous for human hands to reach. Through thorough research, diverse robotics have been expressly designed to surmount our physical limitations and save lives while ensuring our safety. It is only apt that we as engineers continue in their footsteps and dedicate ourselves to making the world a safer place.

Enter Alex, CG2111A's flagship search and rescue bot designed to be piloted by an operator remotely through a secure network to search for trapped "victims" in an unfamiliar terrain. Equipped with an RP-Lidar sensor, Alex is capable of performing Simultaneous Localization and Mapping (SLAM) to provide accurate estimation of its current position in the environment. Alex uses a TCS3200 colour sensor to identify victims and their health conditions. Such information is relayed back to the operator through the secure network. Furthermore, teleoperation of Alex is done through the secure network. This secure network uses TLS which adds a layer of authentication to prevent hackers from intercepting transmission of data.

In our run, Alex adeptly traversed unfamiliar terrain, swiftly pinpointing all victims and securing a safe parking position in less than four minutes. By streamlining our setup processes, Alex took a total of less than six minutes to complete its mission from start to end. This report will go in depth into the implementations done by the team and the hardware/software modifications made to improve Alex. Links to our github repository are included in this report.

Link to Github repository (TLS, Arduino)
Link to Github repository (ROS Workspace)

2. Review of State of the Art

Rover-X

Rover-X, a robotic dog developed by Singapore's Home Team Science and Technology Agency (HTX), aids frontline officers in security patrols and search and rescue missions. Equipped with 3D LiDAR, multiple cameras, gas detection sensors, and thermal imaging, it assists in HazMat incidents both remotely and autonomously. Utilizing legged-locomotion and various sensors, Rover-X navigates uneven terrain like stairs or kerbs, unlike wheeled robots. Rover-X's 3D LiDAR and depth camera enable it to obtain precise 3D information of its surroundings, while its other cameras (front camera, rear camera and a zoom camera) provides visual data of its surroundings. Video analytics software allows the robot to process this data, enabling it to identify victims or possible hazards.

Strengths

- Controlled remotely to reduce the need for human intervention in high-risk situations.
- Legged-locomotion allows Rover-X to navigate through terrains that are challenging for conventional wheeled robots.
- Variety of sensors and payloads allow Rover-X to be perform multiple tasks to aid search and rescue operations.

Weaknesses

- Most of the robot is exposed to the external environment. Hence the robot may be made more prone to physical damage due to its lack of structural reinforcement.
- Operator is unable to verbally communicate with personnel interacting with the robot on the other end.

Wheelbarrow (Bomb Disposal Robot)

Dating as far back as 1972, one tele-operating platform is the wheelbarrow, a bomb disposal robot used by British Army bomb disposal teams [1]. This robot aims to remove the need for humans to go near any bombs for defusal or transportation. This robot is controlled by humans using game controllers with the aid of on-board cameras and sensors through wireless communications. Some of the main functionalities of the robot include the ability to move explosives around with its robotic arm and to spray high pressure water jets to defuse the bomb. In view of making the robot adaptable to all kinds of terrains, caterpillar tracks are implemented so that it can climb staircases and move on low grip surfaces. Wire cutters and multiple arms are also used so that they can bypass fences or even opening car boots.

Strengths

- Removing the need for humans to risk their lives going near any explosives.
- The ability to traverse all kinds of terrains and obstacles.

Weakness

- With all machines that uses wireless communication, it faces risks of being hacked
- It may not be able to handle bombs of sizes larger than robotic arm of robot.
- Back in the past, there were no cameras to provide visual from the robot's perspective. Hence operating distance was limited to line-of-sight distance to robot.

3. System Architecture

The diagram below illustrates the system architecture of Alex.

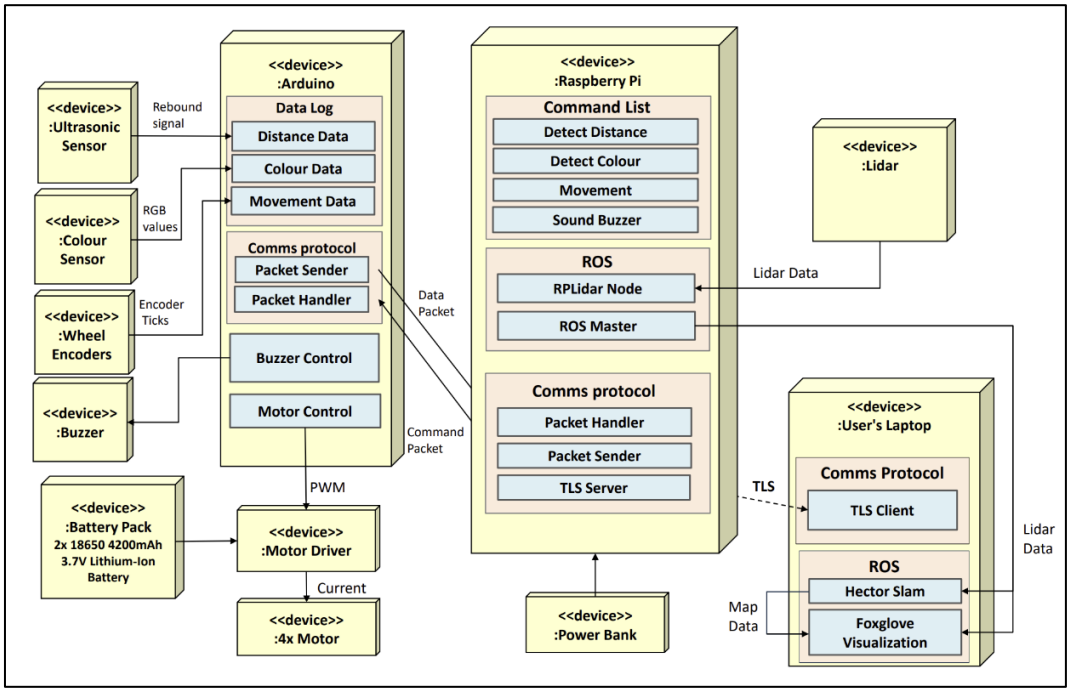
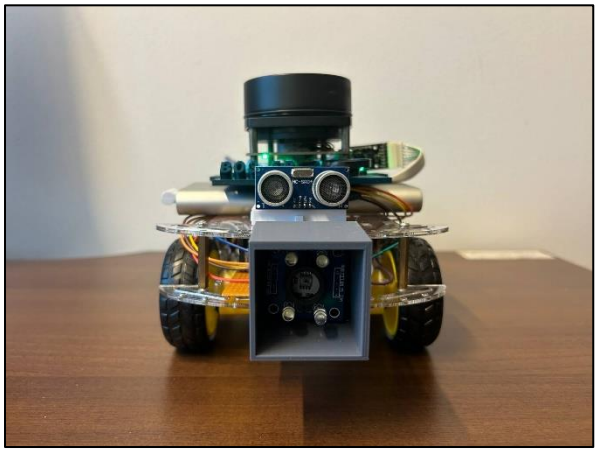


Fig. 1 – Alex System Architecture

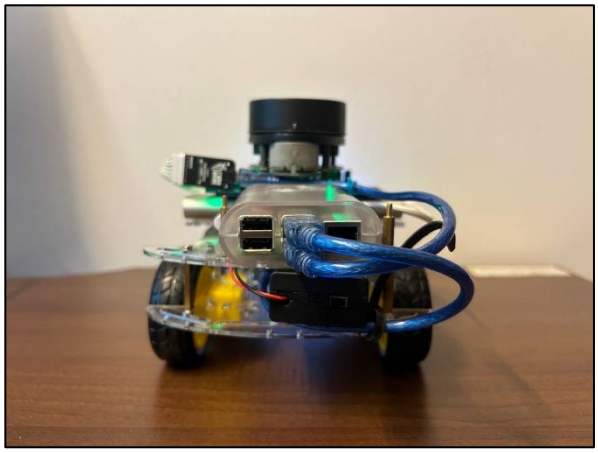
4. Hardware Design

In this section, we will first illustrate the final placement of our hardware components and the wiring of the electrical components before elaborating on the inclusion of non-standard hardware to improve the performance of Alex.

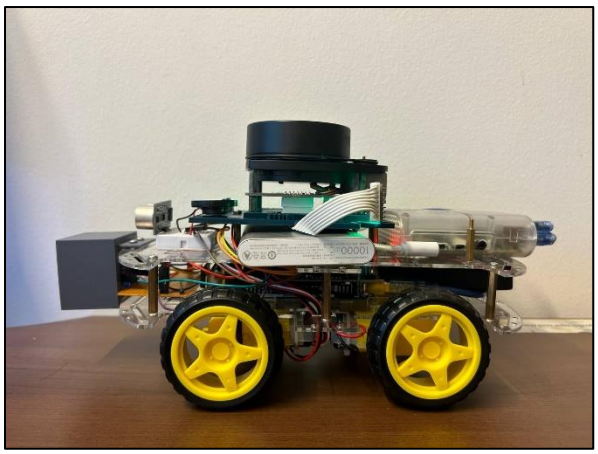
4.1 Final Placement of Hardware Components



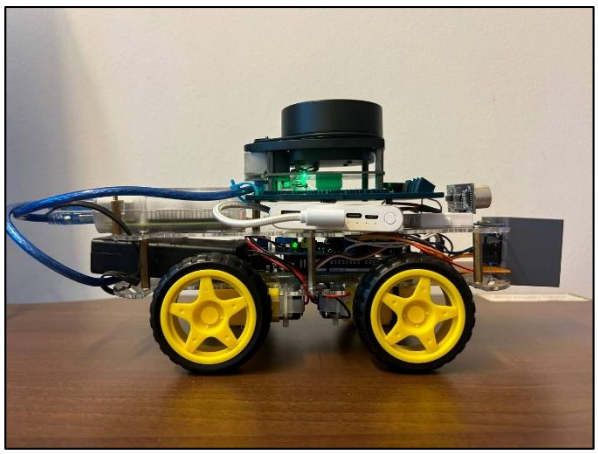
Front View



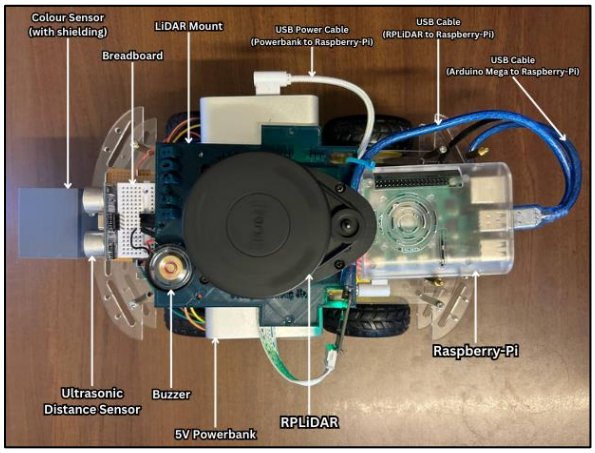
Rear View



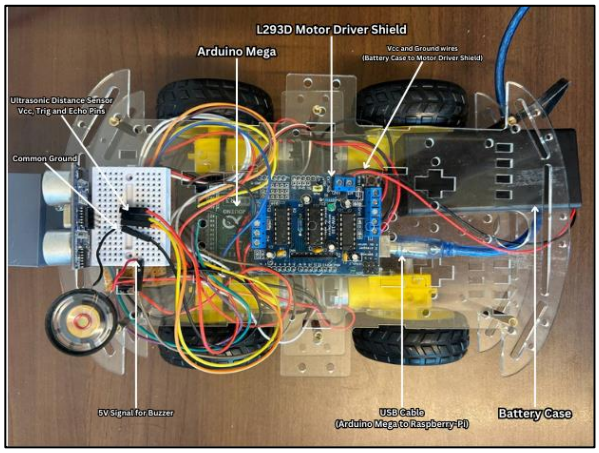
Side View (Left)



Side View (Right)



Top-Down View (Top Layer)



Top-Down View (Bottom Layer)

Fig. 2 – Different Views of Alex

For a clearer view of the image, refer to Appendix 9.1 Hardware Placement.

4.2 Electrical Design

The schematic below illustrates the wiring of electrical components pertaining to the Arduino Mega, motors and sensors.

Red wires: Wires carrying V_{cc} or +5V

Black wires: Common ground.

Pink wires: Input signals to the Arduino Mega's

Blue wires: Output signals from the Mega to the respective sensors.

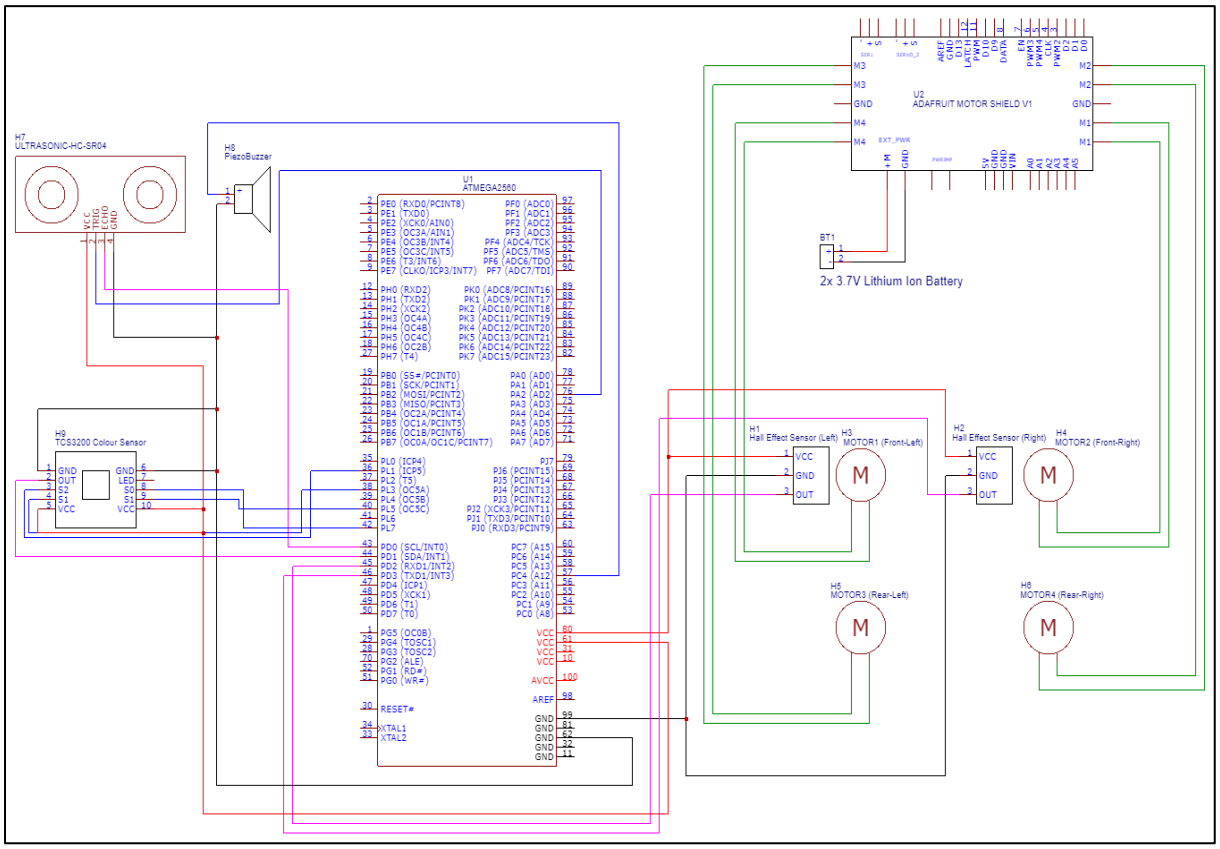


Fig. 3 – Electrical Schematic of Alex

4.3 Non-standard Hardware

4.3.1 Lidar Mount

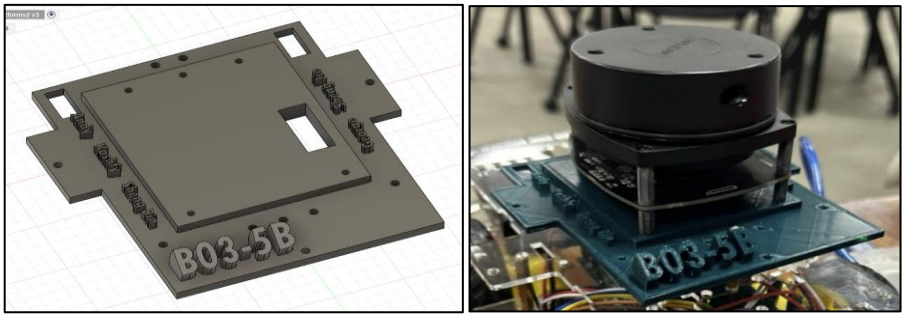


Fig. 4 – Lidar Mount (CAD and Actual)

The Lidar Mount was designed specifically to mount the lidar onto the acrylic chassis. The mount positions the lidar at the center of the chassis to minimize translational movement of the lidar when Alex rotates, which improves performance of the SLAM algorithm used. Holes were included for efficient cable management, helping to prevent accidental collisions that would otherwise compromise the mission. Additionally, this lidar mount features our team number and the names of the team members to add in our personal touch and identity.

4.3.2 Colour Sensor Shielding

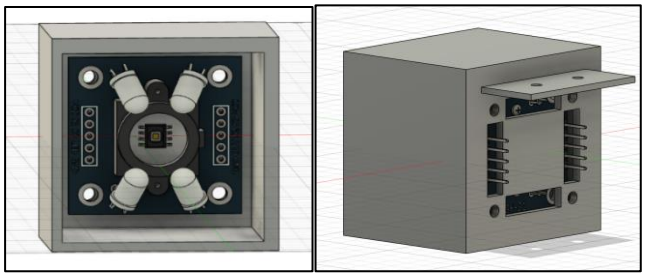


Fig. 5 – CAD of Colour Sensor Shield

The colour sensor shield limits the amount of ambient light entering the sensor from the sides. This improved the effective distance of our colour sensing algorithm under various lighting conditions. We were able to achieve an effective distance of 10cm (measured from the opening of the shielding) in all tested lighting conditions.

4.3.3 Lithium-Ion Batteries and Battery Case

Two 3.7V Lithium-Ion Batteries were used in place of the four AA batteries provided. These two batteries alone were able to provide our motor driver with a total voltage of 7.4V as opposed to the 4.8V output of four AA batteries. As such, Alex's motors were made more powerful, allowing Alex to move faster and make more consistent turns without stalling.

In addition, these batteries were much lighter than the provided AA batteries. This reduced the overall weight of Alex.

For a detailed illustration of Batteries used, refer to Appendix 9.2 Lithium-Ion Batteries.

4.3.4 Soldered Protoboard (Power Distribution)

Our team made use of lab equipment available to create our own power distribution board using a protoboard. The idea of the power distribution board is to allow for easier connection of Vcc and Ground pins of electrical components. The protoboard features a common Vcc and Ground. This allowed us to adjust Vcc and Ground connections without having to directly access the Vcc and Ground ports of the Arduino Mega.

Furthermore, the protoboard relays hall effect sensor output signals to the Arduino Mega, which also allows us to detach the sensor wires for debugging without having to directly access the ports of the Arduino Mega.

For a detailed illustration of protoboard, refer to Appendix 9.3 Soldered Protoboard.

4.3.5 Buzzer

A simple 0.25W speaker was added to Alex to play various tunes during the run, with specialized tunes to play during various phases of the run - start-up, identification of a green victim, identification of a red victim, mission completion and a bonus tone. While this buzzer was implemented purely out of amusement, the buzzer's functionalities can be adapted for real-life practical purposes such as notifying victims of its presence.

5. Firmware Design

5.1 High Level Algorithm on Arduino Mega

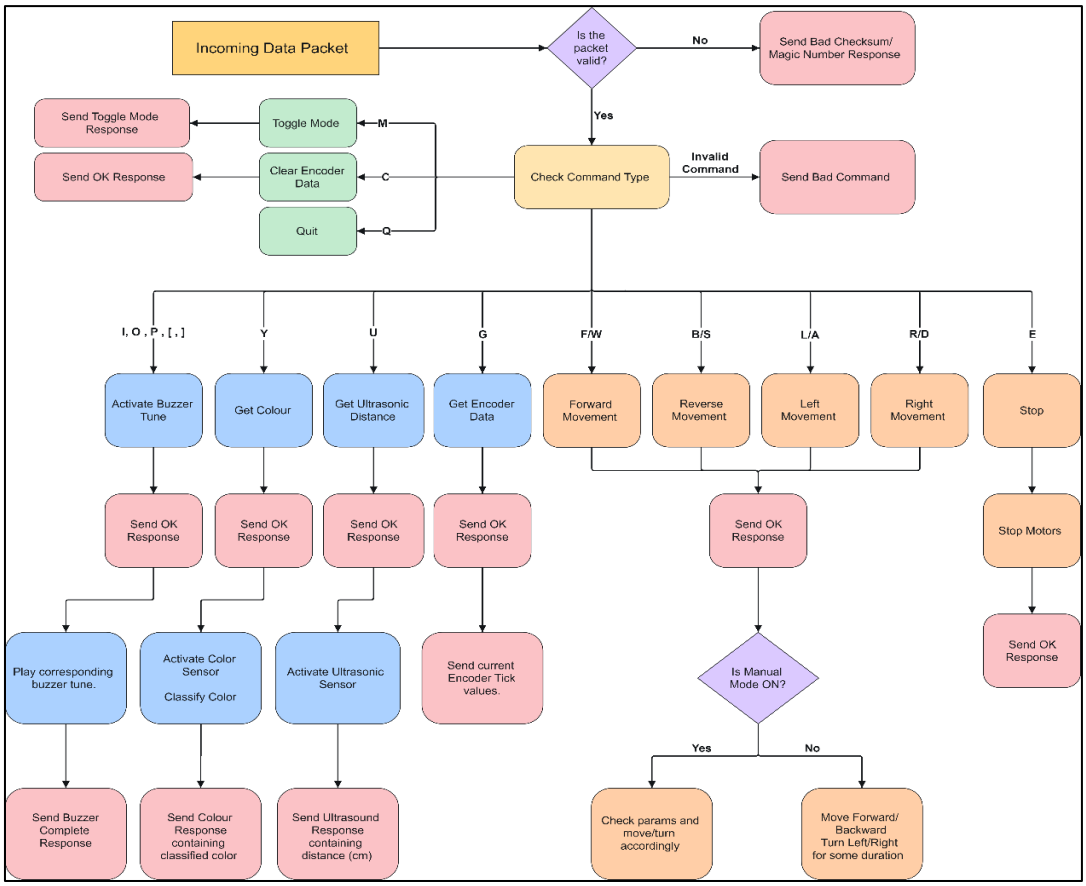


Fig 6. – Program Flow of Mega

5.2 Communication Protocol

5.2.1 Communication Protocol

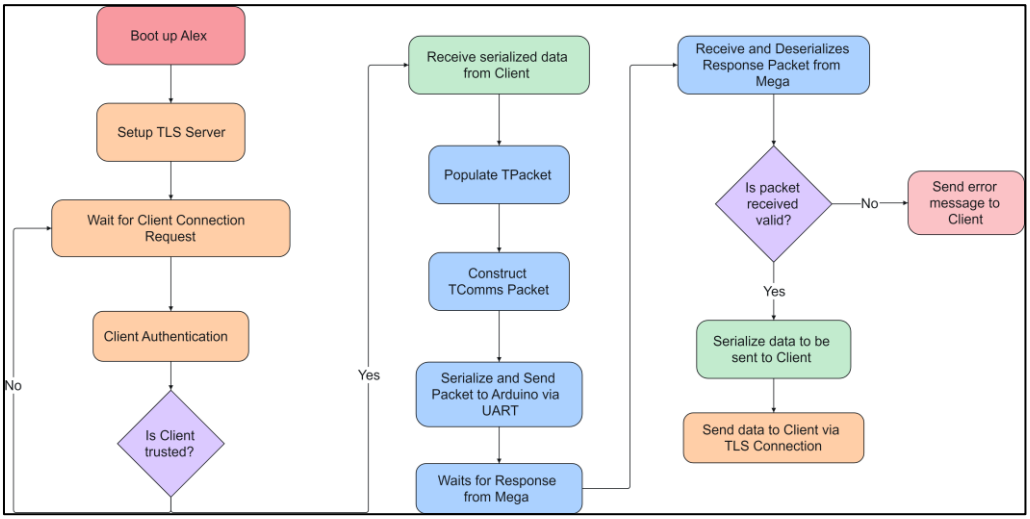


Fig 7. – TLS and Serial Communication Process on RPi

Brief overview of Communication Protocol

1. Serial communication is established between the Pi and Mega at 9600 bps, with a frame format of 8N1. The Mega polls for availability of data via serial.available(), populating a buffer which is then deserialized into a TPacket. Packet integrity is also verified during the deserialization process (details in 5.2.2).
2. After successful deserialization of incoming data into a TPacket, the Mega invokes the relevant packet handler function based on the command type of the packet.

a. Using COMMAND_GET_ULTRASONIC as an example, the Mega would trigger the ultrasonic sensor and get a distance reading.
b. The distance reading is then packaged into a new TPacket that is then included in a TComms object which is serialized and transmitted over the serial port to the Pi.

3. The transmitted data is then deserialized by the Pi into a TComms object, before the TPacket is extracted to be processed.
4. A similar process then occurs on the Pi, where based on the command type of TPacket, the corresponding packet handler function is invoked. Using the ultrasonic command example, the ultrasonic data is copied into a buffer which is then transmitted to the operator via the TLS connection.
5. The client parses the buffer and prints to the terminal the ultrasonic reading and the process repeats similarly as the operator continues teleoperation.

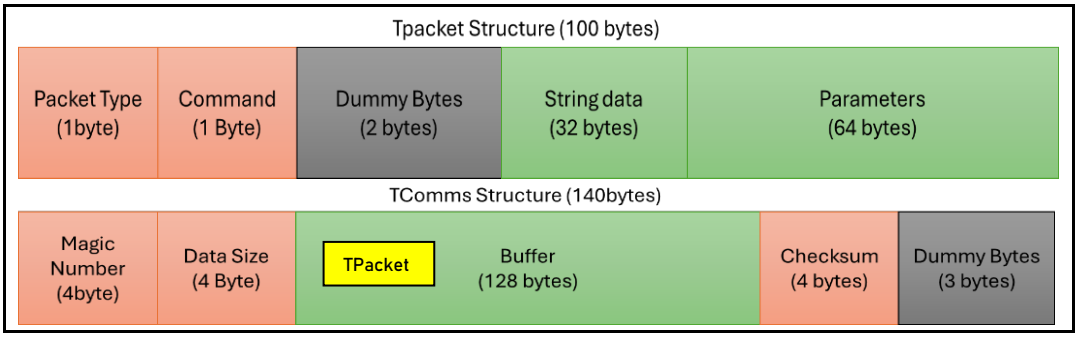


Fig 8. Packet Structure

5.2.2 Packet Integrity

The packet integrity is verified using magic number and checksum. The magic number is predetermined 4-byte chain of 1s and 0s. On the receiving machine, any incoming packets must have this magic number for it to be valid. Any difference in this sequence of bytes will mean that there are bit flips when transferring the data and thus the packet cannot be used. Another way to verify the packet integrity is through the checksum which is computed on the sending machine by applying bitwise XOR operation on the data being sent, this computed result is appended to the data packet. The same computation will be done on the receiving machine and the result is matched with the attached checksum. If they are the same, then we can determine that the data has a good chance that it is not corrupted.

5.3 Serial Communication (Bare Metal)

Our Bare Metal Serial implementation was designed to work exactly like Arduino's own Serial library. We implemented a Circular Buffer class which is used by our UART class. Our UART class contains two Circular Buffers, each one for receiving and transmission respectively. These buffers are of size 512 bytes to comfortably accommodate incoming packets without loss of data. Interrupts are used to read and write from the buffers. USART0_RX_vect triggers the population of the rx_buffer while USART0_UDRE_vect triggers the sending of data from the tx_buffer, disabling the interrupt when the buffer is empty. To improve efficiency, we can directly write transmission data to UDR0 instead of populating tx_buffer first when tx_buffer is empty. We used a baud rate of 9600 bps and a frame format of 8N1 to ensure compatibility with other parts of the program. Using our implementation, we were able to fully replicate the functionality of the Serial sketch library in Bare Metal. Full implementation details can be found in Appendix 9.4 Bare-Metal Serial Implementation.

5.4 Ultrasonic Sensor (Bare Metal)

The Bare Metal implementation for ultrasonic sensor is done by using one of the internal clocks of the Arduino Mega (TCCR5A&B). When getDistance() function is invoked, the Arduino will enable the interrupt (INT0) on the Echo pin to be triggered during a logic change. By directly manipulating the PORTA register to control the Trigger pin, the ultrasonic sensor emits an ultrasonic pulse. This will activate the interrupt for the first time to start the timer. When it receives the reflected signal, the interrupt will trigger again to stop the timer. The time is then taken (value of TCNT5) and converted to distance and the interrupt is disabled. Refer to Appendix 9.5 Bare-Metal Ultrasonic Sensor Implementation for Code.

5.5 Colour Sensor (Bare Metal)

In setupColor(), GPIO pins PL1, PL3, PL5 and PL7 were set to output in DDRL. This allows for the selection of different photodiodes (via PL1 and PL3. This also allows for the selection of frequency scaling (via PL5 and PL7) for the current-to-frequency converter in the TCS3200. PD1 is set as an input in DDRD. By using the built-in pulseIn() function, we are able to obtain the wavelength of the waveform received by PD1, which corresponds to the intensity of the light detected. The table below illustrates the selection of filters using BareMetal. Refer to Appendix 9.6 Bare-Metal Colour Sensor Implementation for code.

Photodiode Colour	PL3	PL1	Bare Metal Implementation
Red	LOW	LOW	PORTL &= ~(1 << PL3); PORTL &= ~(1 << PL1);
Blue	LOW	HIGH	PORTL &= ~(1 << PL3); PORTL |= (1 << PL1);
Clear (No filter)	HIGH	LOW	Not used
Green	HIGH	HIGH	PORTL |= (1 << PL3); PORTL |= (1 << PL1);

5.6 Colour Classification Algorithm (K-NN)

A Euclidean distance variation of K-Nearest Neighbours (K-NN) algorithm was used to classify the colours detected by Alex. This approach uses a known dataset and compares the new datapoint to existing datapoints to perform classification.

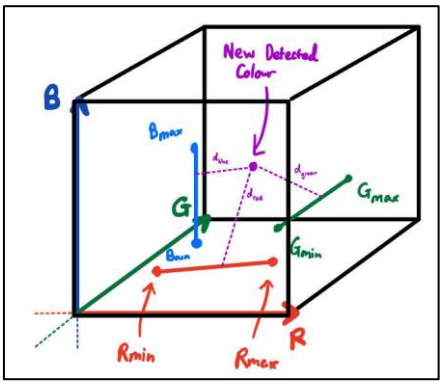


Fig 9. K-NN in RGB Colour Space

The dataset collected for the colours, green, red and white, consists of minimum and maximum RGB intensity values obtained over a range of distances and in different ambient lighting. Plotting these values in the RGB Colour Space, we get a line of RGB intensity values for each colour. These lines form datapoints to be used in the classification algorithm.

When the command to detect colour is received by the Arduino Mega, the Arduino Mega samples intensities of red, green and blue (RGB) light for the newly detected colour. The RGB intensities of the detected colour are represented as a point in the RGB Colour Space. The algorithm calculates the shortest Euclidean distance between this point and the lines formed from the known datapoints. The colour corresponding to the line with the shortest distance will be identified as the colour detected.

5.7 Buzzer

By using the tone() function in the Arduino library, a specified PWM pin on the Arduino can be set to output a square wave of 50% duty cycle at a specified frequency, for a specified duration. As such, when connected to a buzzer, the buzzer is able to emit sound waves of various frequencies. This allows the Arduino Mega to play different notes for varying durations, thus playing a tune. By iterating through an array of pre-defined frequencies and durations, Alex is able to play multiple tunes set by the team during its run.

6. Software Design

6.1 High Level Algorithm on Pi

1. Initialize communication between RPi, Arduino and PC
2. Launch ROS nodes.
3. Operator makes decisions based on visualization.
4. Operator sends command to RPi through TLS Connection.
5. RPi communicates with Arduino through Serial
6. RPi sends information back to operator through TLS Connection.
7. Repeat steps 3-6 until objectives are completed.

Refer to Fig. 7, for TLS and Serial Communication Process on RPi.

6.2 Teleoperation

6.2.1 Movement Commands

Enabling Manual Mode requires the user to key two inputs into terminal to specify the set distance/angle to move/rotate and the motor speed in percentage.

Keyboard Input	Command	Description
F / B	Move Forward / Backward	Alex moves forward / backward at a specified speed over a specified distance
L / R	Turn Left / Right	Alex turns left / right at a specified speed over a specified angle of rotation

By disabling manual mode, Alex now moves in small, fixed distances/angles. This mode allows users to control Alex by simply pressing the desired keyboard input without the need to press enter to confirm their choice.

Implementation for this mode will be elaborated more under “6.6.1 Free mode using getch()”.

Keyboard Input	Command	Description
W	Move Forward	Alex moves forward at 70% speed for a duration of 170ms
A	Turn Left	Alex turns left at 100% speed for a duration of 110ms
S	Move Backward	Alex moves backward at 70% speed for a duration of 170ms
D	Turn Right	Alex turns right at 100% speed for a duration of 110ms

6.2.2 Sensor Commands

Keyboard Input	Command	Description
U	Get Ultrasonic Data	Activate Ultrasonic Sensor
Y	Get Colour Data	Activate Colour Sensor
G	Get Encoder Data	Returns encoder tick values

6.2.3 Other Commands

Keyboard Input	Command	Description
I / O / P / [/]	Active Respective Buzzer Tune	Plays start_tone / red_found / green_found / end_tone / tokyo_drift tunes
M	Toggle ManualMode	Switch between ManualMode: ON and ManualMode: OFF
C	Clear Encoder	Clear all encoder tick values
Q	Exit Session	Close TLS Connection to Server

6.4 Transport-Layer Security (TLS)

TLS is used to add a layer of authentication to our network. With this protocol, we can establish a secure way of sending information between the Pi and the operator's laptop during the operation. This is done using a private and public key generated from the RSA algorithm which serves as a lock-key pair in encrypting and decrypting message sent through the internet and also a certificate authority to validate the integrity of the server. The RPi acts as the server while the operator connects to the server as a client. With TLS, we ensure that the client trying to connect to the server is authenticated.

6.5 ROS Computation Graph

The diagram below illustrates the ROS nodes and topics essential for SLAM and visualization. It's crucial to highlight that all nodes, except for rplidar node, are executed on the operator's laptop. This strategy minimizes resource usage on the Pi, ensuring optimal performance and timely update on the map during teleoperation.

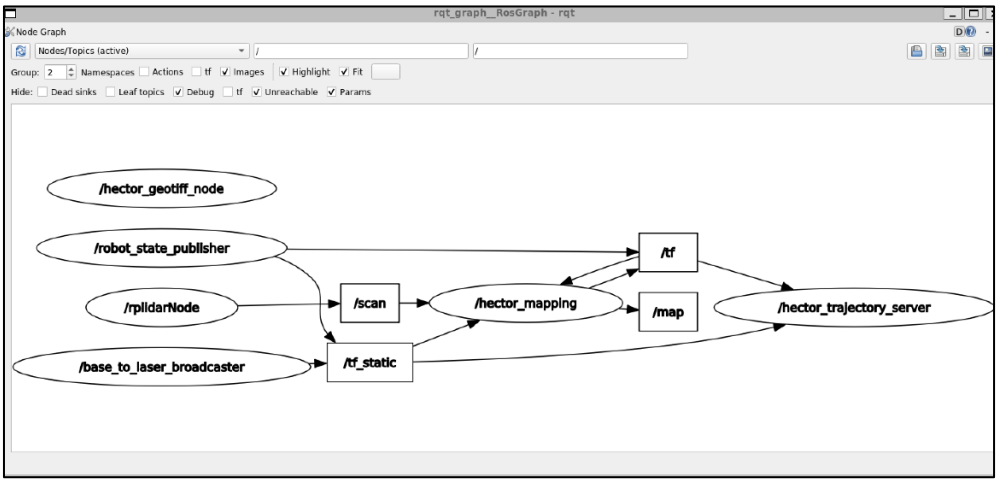


Fig. 10. – ROS Computation Graph

Description of each node and topic can be found in Appendix 9.7 ROS Computation Graph.
ROS launch file used can be found in Appendix 9.8 ROS Launch Files.

6.6 Additional Software Features

6.6.1 Free mode using getch()

Using our own getch() function allowed for more accurate and intuitive operation of Alex. While prudence is necessary when navigating territory, the nature of search and rescue missions also call for urgency. Free mode using getch() makes this possible by allowing for finetuned movement that is also responsive. getch() makes use of <termios.h> and timings for rapid user input without jamming up the Serial channel.

For more details on implementation, refer to Appendix 9.9 Getch() Implementation.

6.6.2 URDF model for visualization

Our team created an accurate model of Alex's dimensions to be used to give us the ability to discern Alex's boundaries with clarity. Combined with our visualization software, it is much easier to avoid collisions.

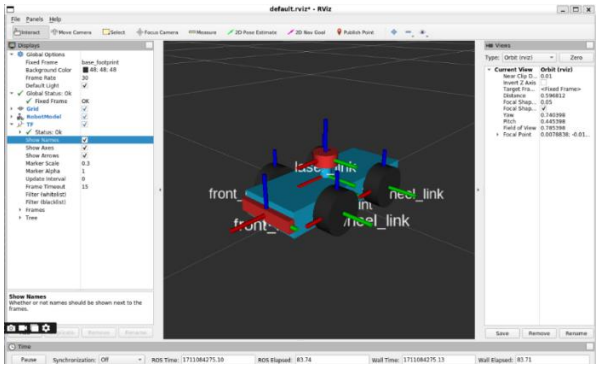


Fig. 11 (left) – URDF of Alex

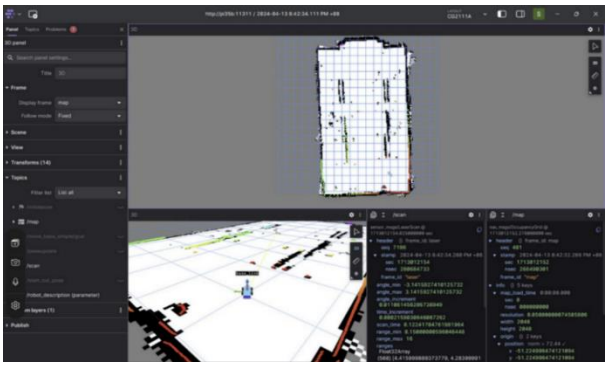


Fig. 12 (right) – Foxglove Visualization Software

6.6.3 Visualization on Foxglove Studio

For our team, visualization of ROS topics was done using Foxglove Studio instead of RVIZ. The use of Foxglove Studio allows multiple devices to access the same ROS network and visualize messages in active topics.

Foxglove Studio provides better flexibility to customize the user interface and provides an accurate distance measuring tool which is extremely helpful for operators in an actual search-and-rescue scenario. The URDF model can be viewed in Foxglove Studio and the pose of the model updates as Alex navigates the terrain.

6.6.4 TMUX: Start-up script

Using a terminal multiplexer such as TMUX allows us to customize terminal interface and run multiple programs using a single terminal. By writing a custom bash script called start_tmux.sh for the RPi and the operator's laptop, we can automatically set up ROS configurations and initialize connection between the RPi and the laptop. This streamlines the startup and initialization process and allows the operator to monitor multiple processes in a single terminal which is extremely useful in a real-world scenario.

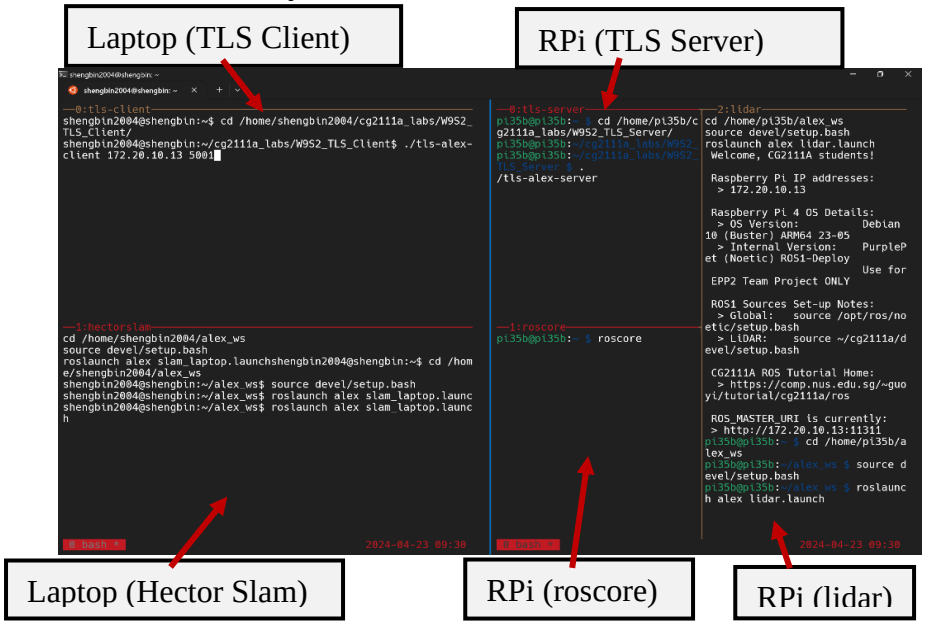


Fig. 13 – Terminal Interface for Operator

For more details on implementation, refer to Appendix 9.10 TMUX Scripts.

7. Conclusion

7.1 Mistakes Made

7.1.1 Lack of Contingency Planning prior to Practice Run

One mistake we made was not rehearsing the setup sequence prior to the trial run. Despite knowing what needed to be done, we did not set a proper contingency plan if an issue arises during setup. The problem surfaced during the trial run when our ROS launch files could not initialize the hector mapping properly, due to a mistake in our transform tree. We spent precious time trying to fix it, resulting in the overrun of the 5 minutes setup time. This reduced the time we had to explore the maze and thus were not able to identify all the colours and park in time. We thus rectified this before the final demo, by running the setup sequence multiple times to familiarize ourselves with it and create a backup plan if anything fails. This proved to be very useful as we were able to complete the setup in less than 3 mins during the actual run, which was a big improvement from the trial.

7.1.2 Misunderstood Inner-working of Sensors

Another mistake that we made was not understanding the inner workings of the sensors before attempting to write the bare-metal code. One example was when writing the bare-metal code for the ultrasonic sensor, we did not fully understand how it was measuring the distance using the pulse it sends out and receives. We initially thought that the echo pin was set to low when the trigger pin was activated and then pulled to high when it receives the pulse back. This resulted in us trying to detect a rising edge interrupt which was incorrect. After much more research, we got a better understanding of how ultrasonic sensor works and figured out that we needed to detect both rising and falling edge of the signal from the Echo Pin. As the Echo Pin outputs a rising edge indicating the emission of the pulse and a falling edge indicating the receiving of the reflected pulse.

7.2 Lessons Learnt

7.2.1 Code Version Control

One important lesson that we learnt was to use platforms like GitHub for version control and smoother code integration. With GitHub, we could always keep a copy of the latest working code to ensure that we always had a working version. Furthermore, everyone from the group can make changes simultaneously, increasing productivity. When it comes to testing, we are also able to test each individual portion which allows us to quickly identify problems. We are then able to merge all working updates to create the new latest working version. The platform together with good coding documentation also allows our group to debug each other's code easily which saves time and effort.

7.2.2 Development Planning and Management

Another lesson that we learnt was the need for the proper planning of project deadlines. By creating a timeline on what needs to be done and by when, we are able to stay on track for development. Considering that we do not have many dedicated labs sessions before the trial and actual run, this schedule ensures that we do not rush all of the project in the last minute. Moreover, our planned timeline clearly outlines the deliverables and what we planned to be implemented. As a result, we have been able to complete the building of Alex in time and have plenty of time for us to familiarize ourselves with the navigating and drawing of the map.

8. References

[1] P. R. Allison, "What does a bomb disposal robot actually do?," British Broadcasting Corporation, 16 July 2016. [Online]. Available: https://www.bbc.com/future/article/20160714-what-does-a-bomb-disposal-robot-actually-do. [Accessed 28 March 2024].

[2] Home Team Science and Technology Agency, "[MEDIA RELEASE] HOME TEAM'S ROBOTIC DOG WILL JOIN FRONTLINE OPERATIONS," Home Team Science and Technology Agency, 20 January 2022. [Online]. Available: https://www.htx.gov.sg/news/media-release-home-team-s-robotic-dog-will-join-frontline-operations. [Accessed 28 March 2024].

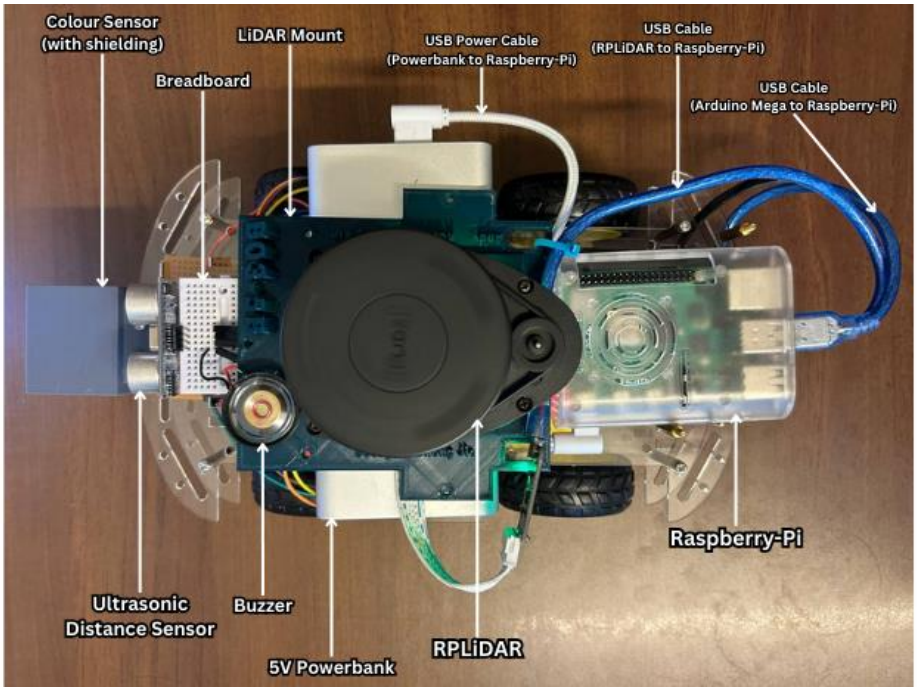
[3] S. Ang, "Four-legged robot can help SCDF respond to hazardous material incidents, assist police with patrols," The Straits Times, 21 January 2022. [Online]. Available: https://www.straitstimes.com/singapore/trials-for-new-hazmat-function-of-robotic-dog-rover-x-to-be-completed-by-mid-year. [Accessed 28 March 2024].

Link to Github repository (TLS, Arduino):
https://github.com/ShengBin-101/CG2111A-Final-Project

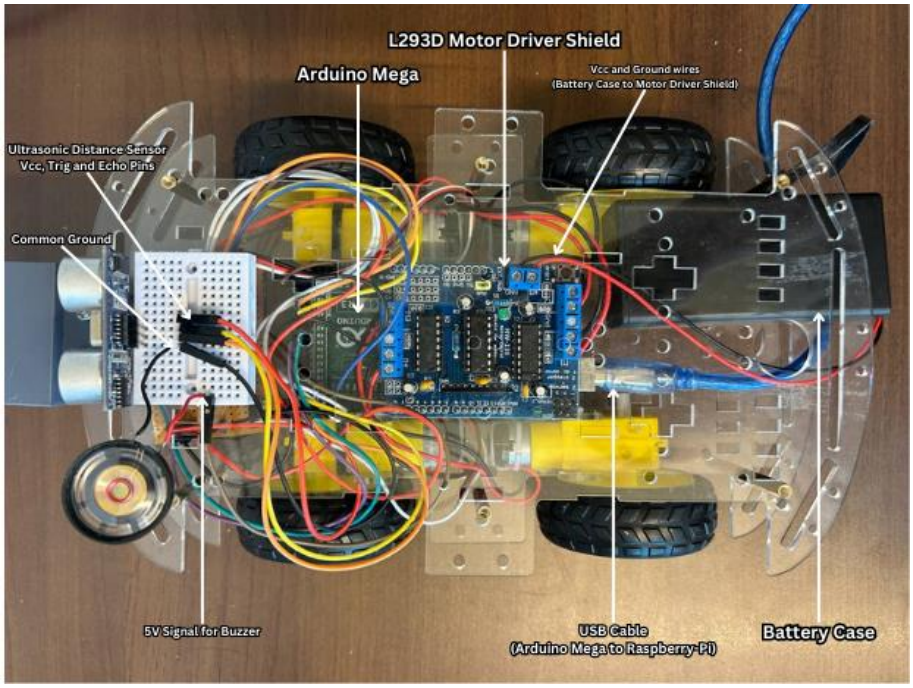
Link to Github repository (ROS Workspace):
https://github.com/ShengBin-101/alex_ws

9. Appendix

9.1 Hardware Placement

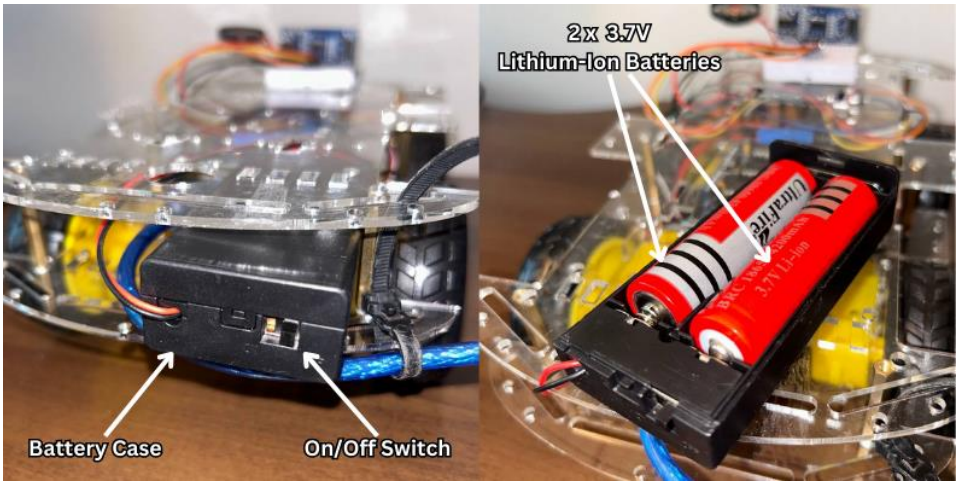


Top Layer

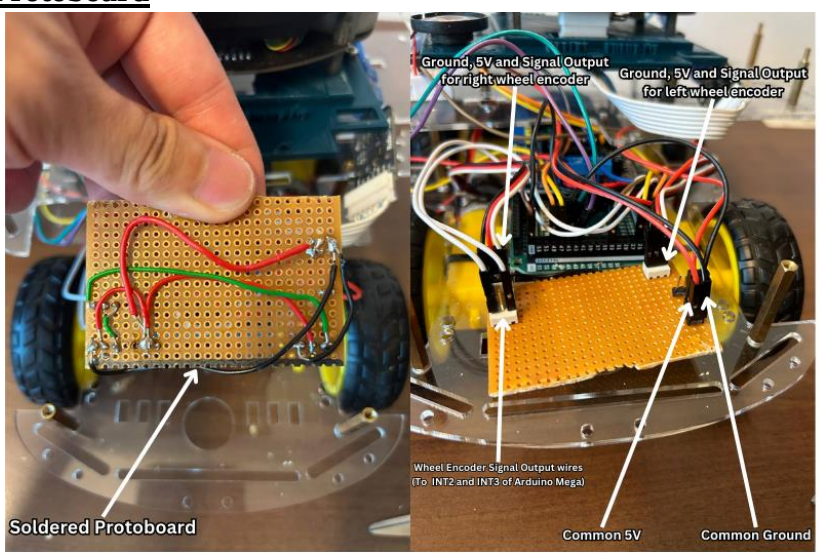


Bottom Layer

9.2 Lithium-Ion Batteries



9.3 Soldered Protoboard



9.4 Bare-Metal Serial Implementation

cbuffer.h (implementation of CBuffer class)

```
#ifndef CBUFFER_H
#define CBUFFER_H

#include <stdint.h>

// Give type and size of data type
template <typename T, int size>
class CBuffer
{
private:
    volatile T array[size];
    volatile int head = 0;
    volatile int tail = 0;

public:
    CBuffer(void) = default;
    bool write(T *data);
    bool read(T *data);
    bool is_full(void);
    bool is_empty(void);
};

template <typename T, int size>
bool CBuffer<T, size>::write(T *data)
{
    if (this->is_full()){
        return false;
    }
    this->head = (this->head + 1) % size;
    this->array[this->head] = *data;
    return true;
}

template <typename T, int size>
bool CBuffer<T, size>::read(T *data)
{
    if (this->is_empty()){
        return false;
    }
    this->tail = (this->tail + 1) % size;
    *data = array[tail];
    return true;
}

template <typename T, int size>
bool CBuffer<T, size>::is_full(void)
{
    if ((this->head + 1) % size == this->tail) {
        return true;
    }
    return false;
}
```

```
template <typename T, int size>
bool CBuffer<T, size>::is_empty(void)
{
    if (this->head == this->tail)
    {
        return true;
    }
    return false;
}
#endif
```

uart.h (uart class declaration and function signatures)

```
#ifndef UART_H
#define UART_H

#include <stdint.h>
#include <avr/pgmspace.h>
#include "cbuffer.h"

class uart
{
public:
    uart() = default;
    ~uart() = default;
    void begin(const unsigned long baud);
    void end(void);
    bool write(uint8_t c);
    void write(const char *c, int len);

    uint8_t read(void);
    bool available(void) const;
    void _rx_interrupt(void);
    void _tx_interrupt(void);

private:
    void turn_tx_on(void)
    {
        UCSR0B |= (1 << UDRIE0);
    }

    void turn_tx_off(void)
    {
        UCSR0B &= ~(1 << UDRIE0);
    }
    volatile CBuffer<uint8_t, 512> rx_buffer;
    volatile CBuffer<uint8_t, 512> tx_buffer;
};

extern uart serial;

#endif
```

uart.cpp (implementation details of uart class)

```
#include <avr/io.h>
#include <avr/interrupt.h>

#include "uart.h"
#include "cbuffer.h"

void uart::begin(const unsigned long baud){
    // Define UCSR0A
    UCSR0A = 0;
    uint16_t ubrr = (F_CPU / (16 * baud)) - 1;
    UBRR0H = (uint8_t)(ubrr >> 8);
    UBRR0L = (uint8_t)ubrr;
    // Frame format of 8N1
    UCSR0C = 0;
    UCSR0C = (1 << UCSZ00) | (1 << UCSZ01);
    // Enable TX and RX
    UCSR0B = 0;
    UCSR0B |= (1 << RXEN0) | (1 << TXEN0);
    // Enable RX interrupt
    UCSR0B |= (1 << RXCIE0);
}

void uart::end(void){
    UCSR0B &= ~((1 << RXEN0) | (1 << TXEN0));
    this->turn_tx_off();
    UCSR0B &= ~(1 << RXCIE0);
}

bool uart::write(uint8_t ch){
    if (this->tx_buffer.is_full()){
        return false;
    }
    // Writing straight if UDR0 is empty and buffer empty
    if ((UCSR0A & (1 << UDRE0)) && this->tx_buffer.is_empty()){
        UDR0 = ch;
    }
    else{
        this->tx_buffer.write(&ch);
        this->turn_tx_on();
    }
    return true;
}

void uart::write(const char *c, int len){
    for (int i = 0; i < len; i++){
        this->write(c[i]);
    }
}

uint8_t uart::read(void)
{
    bool status;
    uint8_t data;
    status = this->rx_buffer.read(&data);
```

```
    if (!status){
        return -1;
    }
    return data;
}

bool uart::available(void) const{
    if (this->rx_buffer.is_empty()){
        return false;
    }
    return true;
}

void uart::_rx_interrupt(void){
    uint8_t data = UDR0;
    this->rx_buffer.write(&data);
}

void uart::_tx_interrupt(void){
    if (this->tx_buffer.is_empty()){
        this->turn_tx_off();
    }
    else{
        uint8_t data;
        this->tx_buffer.read(&data);
        UDR0 = data;
    }
}

uart serial;

ISR(USART0_RX_vect){
    serial._rx_interrupt();
}

ISR(USART0_UDRE_vect){
    serial._tx_interrupt();
}
```

9.5 Bare-Metal Ultrasonic Sensor Implementation

Code Segment for Ultrasonic Sensor Bare-Metal

```
#define TRIGGER DDA2  // 24
#define ECHO DDD0     // 21 (Which is also tied to External Interrupt 0)
volatile uint16_t pulse;
volatile uint8_t edge = 0;
void setupUltrasonic() {
    DDRA |= (1 << TRIGGER); // Set TRIGGER as output
    DDRD &= ~(1 << ECHO);   // Set ECHO as input
    PORTA &= ~(1 << TRIGGER);
    TCCR5A = 0b00000000;
    TCCR5B = 0b00001000;
    TCNT5 = 0;
    //setup INT0 to trigger for ANY EDGE CHANGE
    EICRA |= (1<< ISC00);
    EICRA &= ~(1<< ISC01);
}
float readPulse() {
  return pulse / 2.0f;
}
ISR(INT0_vect)
{
    switch (edge)
    {
      case 0:
        edge = 1;
        TCCR5B |= (1 << CS51);
        break;
      case 1:
        edge = 0;
        TCCR5B &= ~(1 << CS51);
        pulse = TCNT5;
        TCNT5 = 0;
        break;
    }
}
void triggerUltrasonic()
{
  PORTA |= (1 << TRIGGER);
  _delay_us(15);
  PORTA &= ~(1 << TRIGGER);
  _delay_ms(20);
}
float getDistance() {
    float duration;
    EIMSK |= (1 << INT0);
    triggerUltrasonic();
    duration = readPulse();
    EIMSK &= ~(1 << INT0);
    return (duration / 2) * (0.0343f) - 5.0f; // in cm
}
```

9.6 Bare-Metal Colour Sensor Implementation

Colour Detection – Function to Calculate Euclidean Distance

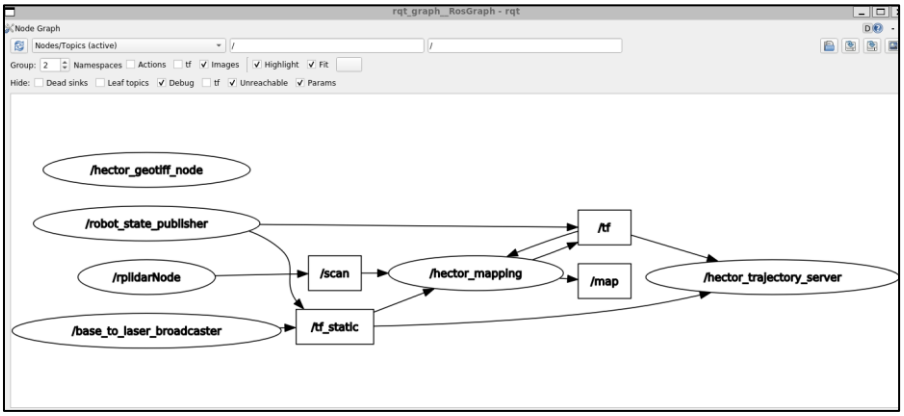
```
//return distance between test point and stored color point
float color_dist(float colourFrequencyMin[3], float colourDirVect[3], float
testArray[3]) {
  float colourMainVect[3];
  float calc_dist = 0;
  float min_dist = 0;
  for (int i = 0; i < VECT_INCR; i += 1) {
    // add the direction vector to the main vector
    for (int j = 0; j < 3; j += 1) {
      colourMainVect[j] = colourFrequencyMin[j] + i * colourDirVect[j];
    }

    float sum = 0;
    float dist_squared[3] = {0, 0, 0};
    for (int i = 0; i < 3; i += 1) {
      dist_squared[i] = colourMainVect[i] - testArray[i];
      dist_squared[i] *= dist_squared[i]; // square the distance value
      sum += dist_squared[i];
    }

    calc_dist = sqrt((double) sum);
    if (min_dist < 1 || calc_dist < min_dist) {
      min_dist = calc_dist;
    }
  }
//   Serial.print("returning color dist: ");
//   Serial.println(min_dist);
  return min_dist;
}
```

Colour Detection – Main Function for Colour Classification using K-Nearest Neighbours

```
int getColor() {
  // Setting RED (R) filtered photodiodes to be read
  PORTL &= ~(1 << S2);
  PORTL &= ~(1 << S3);
  redFrequency = getAvgReading(5);
  colorTestArray[0] = redFrequency;

  // Setting GREEN (G) filtered photodiodes to be read
  PORTL |= (1 << S2);
  PORTL |= (1 << S3);
  // Reading the output frequency
  greenFrequency = getAvgReading(5);
  colorTestArray[1] = greenFrequency;

  // Setting BLUE (B) filtered photodiodes to be read
  PORTL &= ~(1 << S2);
  PORTL |= (1 << S3);
  // Reading the output frequency
  blueFrequency = getAvgReading(5);
  colorTestArray[2] = blueFrequency;

  // K-nearest neighbours
  for (int i = WHITE; i < 3; i += 1) { // start looping from WHITE (i.e. 0)
```

```
    colorDist[i] = color_dist(colorFreqMin[i], colorDirVect[i], colorTestArray);
  };

  float minDist = colorDist[WHITE];
  int closestColor = WHITE;
  for (int color = WHITE; color < 3; color += 1) {
    if (colorDist[color] < minDist) {
      minDist = colorDist[color];
      closestColor = color;
    }
  }

  return closestColor;
}
```

9.7 ROS Computation Graph



Topic Name	Description of each message	Subscribers	Publishers
/scan	One revolution worth of lidar scan data.	hector_mapping	rplidarNode
/map	2D occupancy grid represented using 2D array where each cell stores probability value indicating occupancy likelihood ranging from 0 (free) to 100 (occupied).	rviz/foxglove	hector_mapping
/tf	Timestamped transformations between existing coordinate frames	hector_mapping hector_trajectory_server	hector_mapping robot_state_publisher
/tf_static	Static transformations that remain constant throughout the lifetime of the system, typically used for fixed frame relationships between components like sensors and the robot	hector_mapping hector_trajectory_server	robot_state_publisher base_to_laser_broadcaster

Node Name	Description	Subscribes To	Publishes To
rplidarNode	Node that publishes laserscan data from the lidar.	-	/scan
robot_state_publisher	Publishes the transform frames for robot description based on the URDF of Alex.	-	/tf /tf_static
base_to_laser_broadcaster	Links the transform frame of the lidar to the transform frame of Alex's chassis, ensures origin of scan data is mapped to the lidar's actual position on chassis in visualization.	-	/tf_static
hector_mapping	Performs hector slam using landmark extraction on lidar scan data to estimate odometry and outputs a 2D occupancy grid.	/tf /tf_static /scan	/tf /map
hector_trajectory_server	Keeps track of transform trajectories extracted from transform data and makes this data accessible via a service and topic. This can be used to visualize path taken by the robot.	-	-
hector_geotiff_node	Node that allows saving of map and robot trajectory data as geotiff maps. Allowing operators to save maps of explored terrains.	-	-

9.8 ROS Launch Files

lidar.launch

```
<?xml version="1.0"?>
<launch>
	<include file="$(find rplidar_ros)/launch/rplidar_a1.launch" />
</launch>
```

slam_laptop.launch

```
<?xml version="1.0"?>
<launch>

  <param name="robot_description"
textfile="/home/shengbin2004/alex_ws/src/alex/urdf/alex.urdf" />

  <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher"/>

  <node pkg="tf2_ros" type="static_transform_publisher"
name="base_to_laser_broadcaster" args="0 0 0 3.14159 0 0 base_link laser"/>

  <include file="$(find hector_mapping)/launch/mapping_default.launch">
    <arg name="base_frame" value="base_link"/>
    <arg name="odom_frame" value="base_link"/>
    <arg name="scan_topic" value="scan"/>
    <arg name="pub_map_odom_transform" value="true"/>
  </include>

  <include file="$(find hector_geotiff_launch)/launch/geotiff_mapper.launch">
    <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
    <arg name="map_file_path" value="/home/shengbin2004/alex_ws/src/alex/maps"/>
  </include>

</launch>
```

9.9 Getch() Implementation

Getch function

```
char getch()
{
	struct termios term, oterm;
	char c = 0;

	tcgetattr(0, &oterm);	//Obtain current terminal attributes
	memcpy(&term, &oterm, sizeof(term)); // Copy over into term
	term.c_lflag &= ~(ICANON | ECHO);	// Disable ICANON and ECHO
	term.c_cc[VMIN] = 1;
	term.c_cc[VTIME] = 0;
	tcsetattr(0, TCSANOW, &term);  // Apply changes
	c = getchar();				// Block function
	tcsetattr(0, TCSANOW, &oterm); // Revert changes
	return c;
}
```

9.10 TMUX Scripts

Tmux Script – Server/Pi

```
#!/bin/bash
# Start a new tmux session
tmux new-session -d -s alex-session
tmux split-window -h
tmux split-window -v
tmux select-pane -t 0
tmux split-window -v
# Rename panes
tmux select-pane -t 0 -T tls-server
tmux select-pane -t 1 -T roscore
tmux select-pane -t 2 -T lidar
tmux select-pane -t 3 -T hectorslam
# navigate to /home/pi35b/cg2111a_labs/W9S2_TLS_Server
tmux send-keys -t 0 "clear" C-m
tmux send-keys -t 0 "cd /home/pi35b/cg2111a_labs/W9S2_TLS_Server/" C-m
tmux send-keys -t 0 "./tls-alex-server"
# Run roscore
tmux send-keys -t 1 "clear" C-m
tmux send-keys -t 1 "roscore"
# Ready Lidar launch file
tmux send-keys -t 2 "cd /home/pi35b/alex_ws" C-m
tmux send-keys -t 2 "source devel/setup.bash" C-m
tmux send-keys -t 2 "roslaunch alex lidar.launch"
# Ready SLAM launch file (on pi, for backup)
tmux send-keys -t 3 "cd /home/pi35b/alex_ws" C-m
tmux send-keys -t 3 "source devel/setup.bash" C-m
tmux send-keys -t 3 "roslaunch alex slam.launch"
# Select the left pane and Attach to the tmux session
tmux select-pane -t 0
tmux attach-session -t alex-session
```

Tmux Script – Client/Laptop

```
#!/bin/bash
# Start a new tmux session
tmux new-session -d -s alex-session-client
tmux split-window -v
# Rename panes
tmux select-pane -t 0 -T tls-client
tmux select-pane -t 1 -T hectorslam
# navigate to /home/pi35b/cg2111a_labs/W9S2_TLS_Server
tmux send-keys -t 0 "clear" C-m
tmux send-keys -t 0 "cd /home/shengbin2004/cg2111a_labs/W9S2_TLS_Client/" C-m
tmux send-keys -t 0 "./tls-alex-client 172.20.10.13 5001"
# Run SLAM launch file
tmux send-keys -t 1 "cd /home/shengbin2004/alex_ws" C-m
tmux send-keys -t 1 "source devel/setup.bash" C-m
tmux send-keys -t 1 "roslaunch alex slam_laptop.launch"
# Select the top pane and Attach to the tmux session
tmux select-pane -t 0
tmux attach-session -t alex-session-client
```