



CG2111A Engineering Principle and Practice
Semester 2 2023/2024

**“Alex to the Rescue”
Final Report
Team: B01-2B**

Name	Student #	Subteam
Gao Yening	A0272253J	Software
Aravind Theertha	A0284675M	Software
Liu Haotian	A0258775J	Hardware
Chang Jia Jun	A0286456N	Hardware

Section 1 Introduction

The goal of this project is to create a search-and-rescue robot that is remote-controlled to navigate through an obstacle-filled area to execute environment-mapping and colour-detection.

The user access-point of this system is the Raspberry-pi, a single-board computer with Wi-Fi capability.

Information from a range-measuring peripheral is used to procure a map with the robot's location for navigation.

An Arduino interfaces between the pi and the peripherals responsible for colour detection and robot movement.

Colour detector module(s) allow the robot to identify red/green objects of interest.

The wheel encoder module(s) provide information about the physical state of the robot/wheels. This enables the program to navigate the robot with a certain precision.

Section 2 Review of State of the Art

1. Boston Dynamics Spot Robot

The Spot Robot helps organisations monitor and operate their sites, as well as collect data. It has stereo cameras, an IMU (Inertial Measurement Unit) and LiDAR. An easy-to-use and intuitive Python client library is used by applications to control Spot, read sensor information, and register payload services.

One advantage of the Spot robot is that it is very reliable. Spot can operate without intervention, with capabilities such as self-righting and dynamic re-planning. Moreover, Spot is very scalable due to an ecosystem that allows for easy management and access to data.

However, Spot has a weak battery life of 90 minutes which limits its operation time. In addition, Spot is unable to perform tasks that require fine dexterous manipulation or lifting heavy debris, which could be necessary during rescue missions.

2. Foster Millan TALON V

TALON V is a remotely operated vehicle, and it is a small, tracked military robot designed for missions ranging from reconnaissance to combat. TALON V has software that is compliant

with certain military standards, which allows interoperability at multiple levels within varying systems configurations. TALON V communicates with the user via a two-way radio or fibre-optic link.

TALON V features a high-performance drive motor option which is capable of speeds greater than 10.80 kph, with enough torque to climb a 46° staircase. This allows it to navigate in the most rugged terrain. Moreover, TALON V can communicate from as far as 1.2km.

One downside of TALON V is that it is complex to use and requires specialised training. In addition, the sophisticated technology utilised by the TALON V makes it power-consuming and it will require expensive lithium-ion batteries and chargers.

Section 3 System Architecture

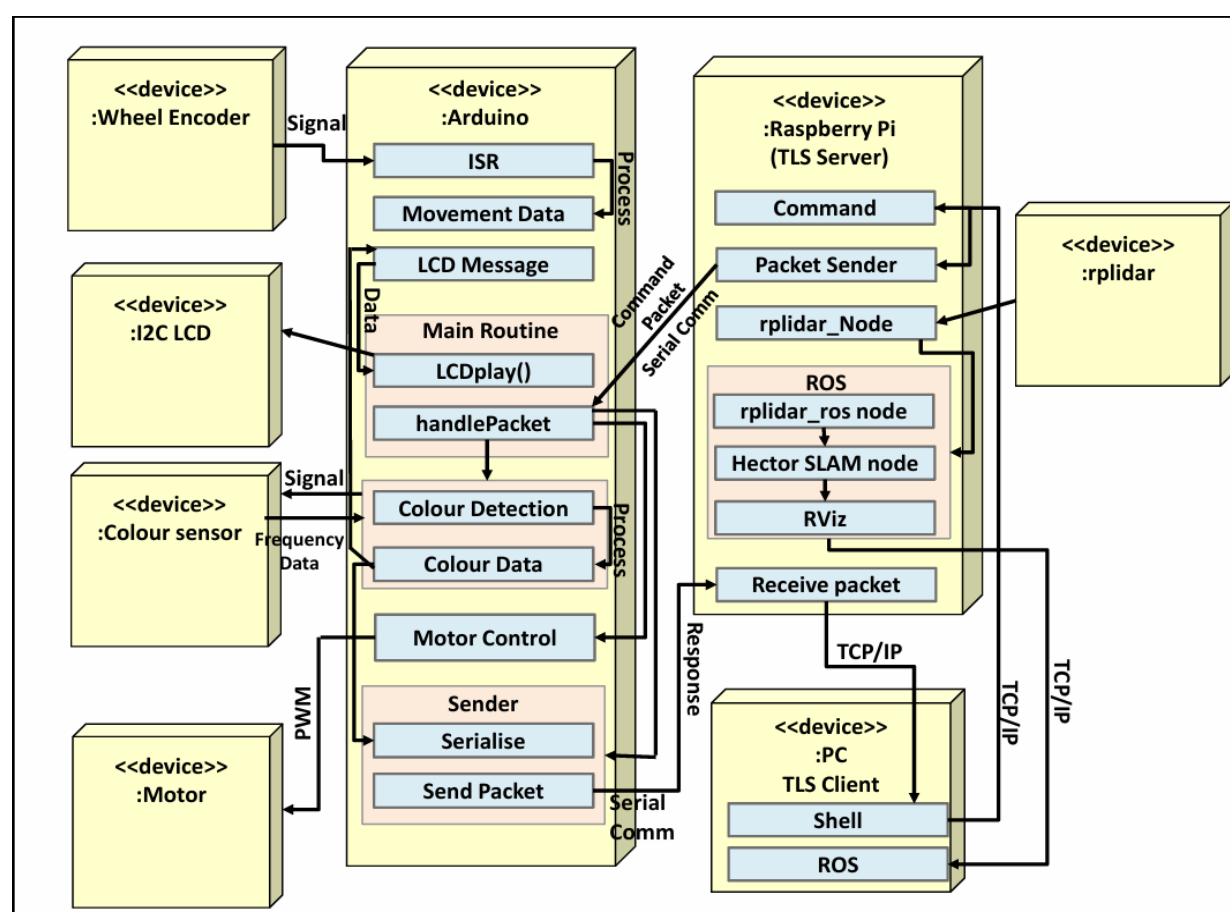


Fig 1. UML Deployment Diagram of Alex

Section 4 Hardware Design

4.1 Design Considerations for Alex

Below shows the hardware components layout on Alex from its Top, Front, Side and Bottom views.



Fig 2. Alex Side View

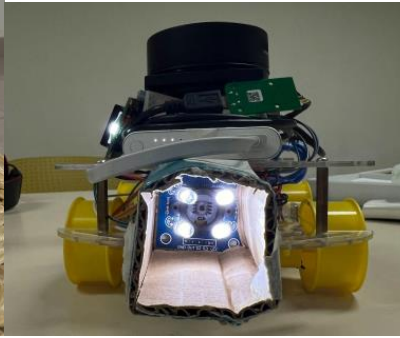


Fig 3. Alex Front View

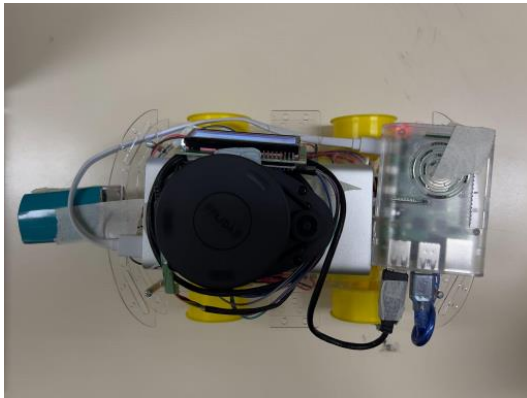


Fig 4. Alex Top View

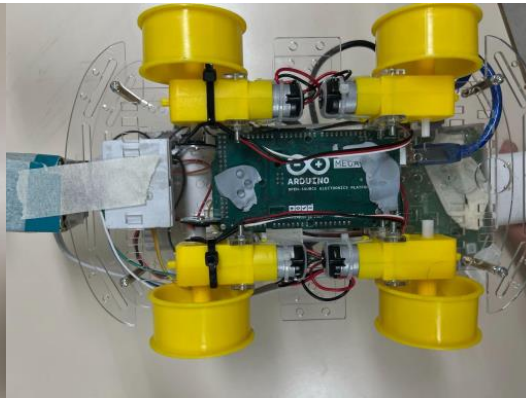


Fig 5. Alex Bottom View

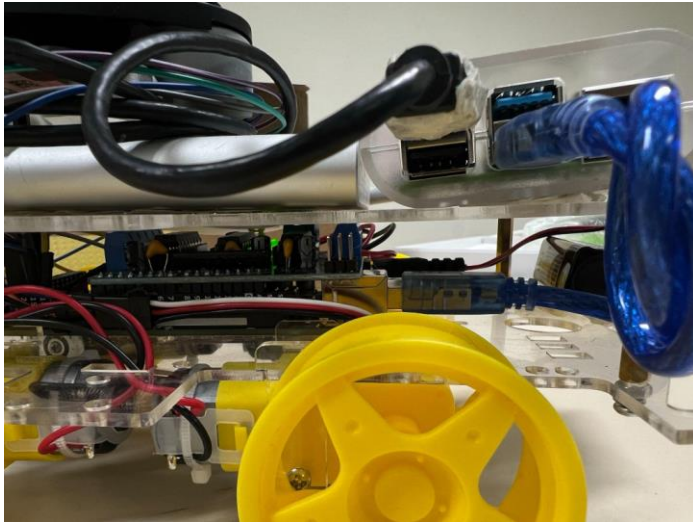


Fig 6



Fig 7. I2R LCD Display

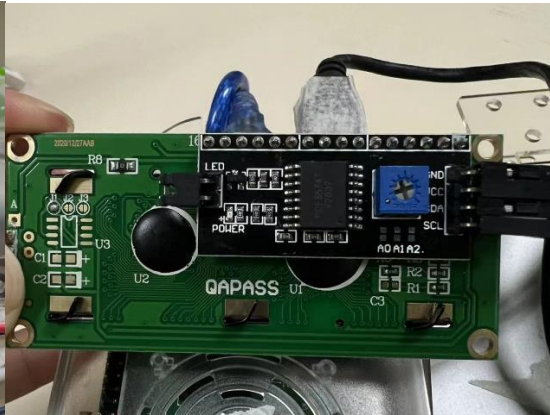


Fig 8. I2R LCD Display

We identified three main concerns when designing Alex, namely space management, centre of gravity.

Space Management

This includes the layout of the components and cable management. For example, the LIDAR must be mounted at the top facing the front for it to scan the environment without obstruction, colour sensors are mounted at the front to scan the dummy. We ensured that we minimised the components protruding out from the frame by using tape and blue tack. We loosened the screws of the frame to allow wires enough space to bend.

Note: We made an attempt to solder by threading a solid-core wire through a prototype board and soldering them, together with any relevant wire(s). In the end, this was not implemented.

Centre of Gravity (CG)

We distributed the weight of the objects evenly. The portable charger was positioned at the centre along the length of the robot while the LiDAR was slightly past the midline towards the front. The Rpi and battery were placed behind the midline.

4.2 Hardware

TCS3200 Colour Sensor

The colour sensor is mounted at the front of Alex to discern whether an object is a victim (Red/Green) or a dummy. We achieve this by measuring the RGB values of the reflected light. This process occurs when the user sends the command to identify the colour.

The module features an 8×8 photodiode array, consisting of 16 diodes with a red filter, 16 with a blue filter, 16 with a green filter, and 16 without any filter (clear). We selectively choose the appropriate filter and read its value.

Regarding the output pin, since the sensor provides a frequency signal, we measure the duration ($\text{Duration} = 1/\text{Frequency}$). Consequently, the higher the frequency associated with a colour, the shorter the measured duration. This relationship allows us to determine the colour of the detected object.¹

To mitigate the impact of ambient lighting on reflected light readings, we shielded the four white LEDs and the sensor using cardboard. This step significantly improves the accuracy of the sensor. During testing, we fine-tuned the colour values to ensure reliable readings. Additionally, we determined the optimal distance between Alex and the dummy to obtain accurate colour measurements.

Capacitor module

To enhance the robustness of the power supply, we placed capacitor(s) of 1000-2000 μf between GND and VCC to smoothen out the voltage supply. This may improve the reliability of the microcontrollers.

I2C LCD1602 Module (Unique Feature)

The purpose of the LCD module is to communicate with casualties by displaying certain messages. For example, the default message is “Help on the way” and “Scanning”. Other messages include classification of the casualty such as “injured, trapped”. Hence, casualty can know the mission the robot is undertaking and whether the robot has made the correct classification.

¹ <https://projecthub.arduino.cc/SurtrTech/colour-detection-using-tcs3200230-a1e463>

An I2C² module interfaces between the Arduino Mega and LCD module. Therefore, only two pins are used on the Arduino Mega for SDA and SCL respectively.

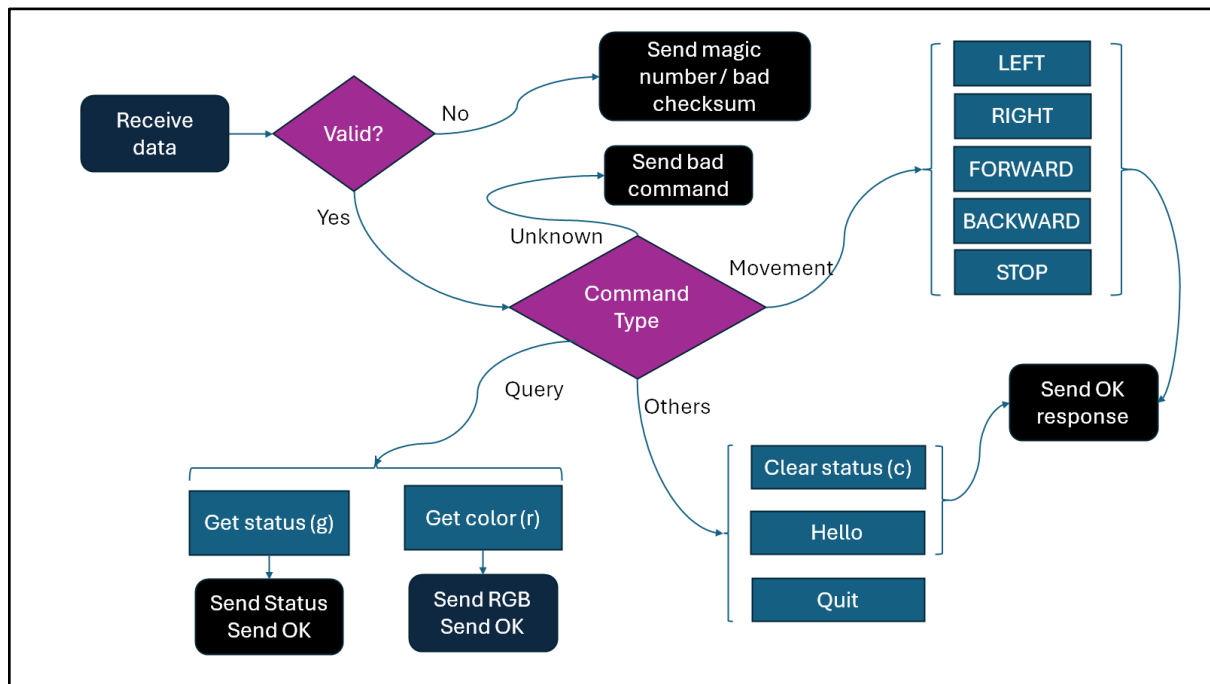
The LiquidCrystal_I2C.h to program how the message will be displayed (see appendix)

2

https://www.ti.com/lit/an/sbaa565/sbaa565.pdf?ts=1714100715786&ref_url=https%253A%252F%252Fwww.google.com%252F#:~:text=I2C%20is%20a%20two%2Dwire,and%20receive%20commands%20and%20data.

5.1 High Level Algorithm on Arduino Mega

In general, the Arduino polls the pi for incoming data. After receiving data, the Arduino checks its validity and handles the command appropriately.



5.2 Communication Protocol

The Arduino and RPi communicate asynchronously via UART with the standard 8N1 frame format with a baud rate of 9600 bps. Each data packet is 100 bytes long. appending magic number, data size and checksum plus the padding used to preempt the Pi's compiler to the data packet when serialising, each packet is 140 bytes long. This means that the time taken for RPi to send a command to Arduino and vice versa is $(140 \times 10) / 9600 = 0.15\text{s}$ or 150ms.

5.3 Bare Metal Programming

[references to appendix]

Section 6 Software Design

6.1 High Level Algorithm on Pi

1. Initialisation
2. Begins loop to scan and render map
3. User command
4. Arduino handles command
5. Repeat steps 2-4 until robot reaches the parking spot

6.1.1 Teleoperation

Movement commands:

We wanted the controls to be simple and intuitive. The template code was tiresome to use as it required a user input of direction and two parameters for power and distance/angle.

We found the robot could move with a precision of no more than 5 units of distance (cm) and 30 degrees of turning. The user would use the standard WASD keys to manoeuvre the robot, and appropriate parameters will be passed under the hood.

The variant capital 'W' and 'S' are used for the most precise forward/backward movements.

Lastly, by intelligently using the braking command 'e', one can extract greater precision.

Query commands:

The primary purpose of the Get optometry data command was for debugging and calibration.

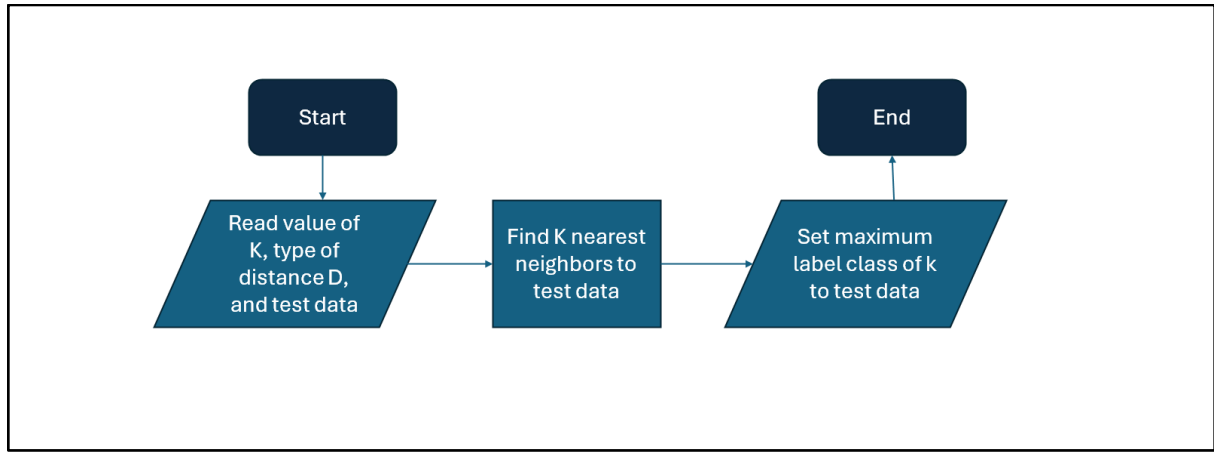
After identifying a possible colour object, the user may use 'c' to identify colour.

Other commands:

The hello command was used to initialise communications between the arduino and pi and for debugging purposes.

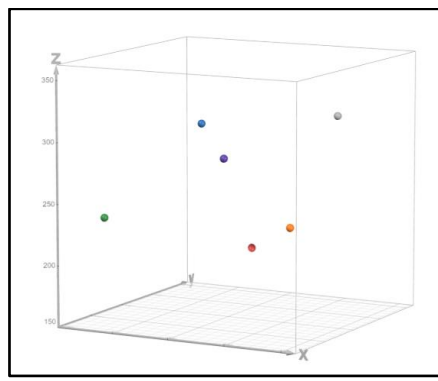
The quit command was useful to restart the system when 'magic number' errors were encountered. This happened when the interval between two successive user inputs was too short.

6.1.2 Colour Detection and Calibration



In our implementation, each colour is represented as a point in a 3-dimensional coordinate system, with the axes corresponding to the red, green, and blue values. We can establish the means of reference colours through data collection of their RGB values across various lighting conditions. In the colour detection process, the RGB values of a detected colour are obtained, and its Euclidean distance from the reference points is calculated. The colour is then classified based on the shortest Euclidean distance.

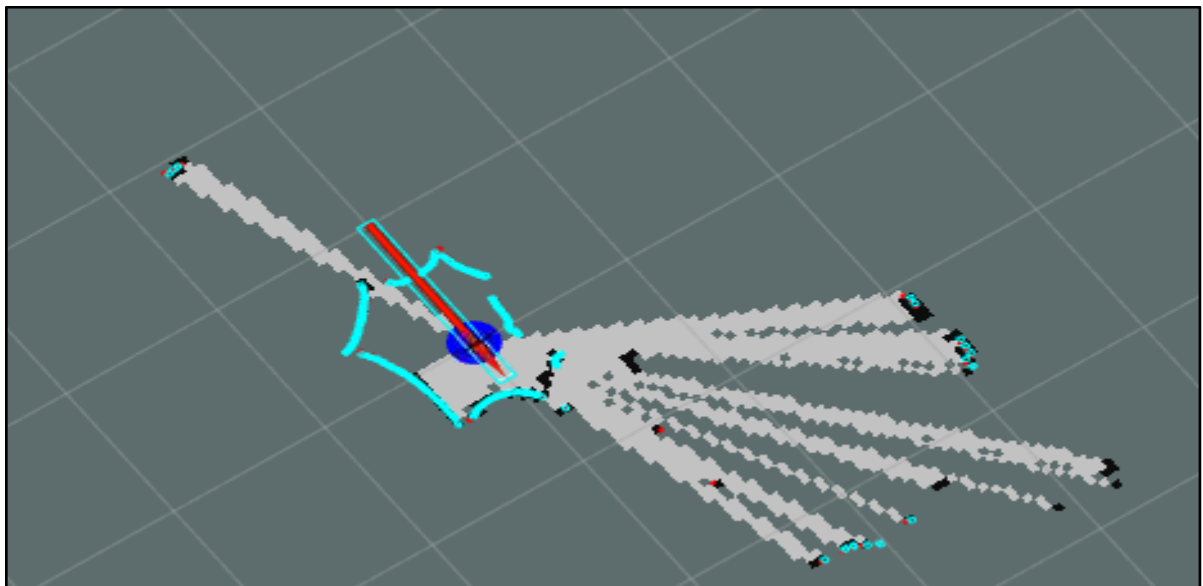
A graphical representation of this idea is as follows:



6.2 Additional Software Features

Alex's hitbox in Rviz


To prevent collision, we had to know the size of Alex relative to its surroundings. This was difficult to gauge when Alex was illustrated as a dot. To address this issue, we used Rviz to model Alex's hitbox. As a form of calibration, we physically surrounded the robot with four items that were just touching it. Then, we referred to Rviz and adjusted the size of the circle representing Alex accordingly. This proved to be a good approximation of Alex's width and to a certain, its length. With a hitbox, the operator is able to make better judgments when controlling the robot.



Calibrating hitbox with Rviz



Rviz 'pose' configuration

 Axes	<input checked="" type="checkbox"/>
+ ✓ Status: Ok	
Reference Frame	laser
Length	0.0001
Radius	0.3
Show Trail	<input type="checkbox"/>
Alpha	1

Rviz 'Axes' configuration



Use of hitboxes in game can aid player judgement

Section 7 Conclusion

7.1 Mistakes made

Poor time management

We fell short of our schedule and had to rush numerous things a few days prior to the final run. Moreover, we did not manage to procure materials we wanted such as heatsink/cooling fan due to time limitations. Hardware design and soldering were also not given as much attention as we desired due to a lack of time awareness.

Insufficient communication

We could have afforded to communicate more with one another to update the team on the progress of the project. By leaving things up to chance, we created a scenario in which we

were passively waiting for instructions. This made us less efficient and have a higher tendency to miss deadlines.

7.2 Lessons learnt

Anticipate error and randomness

Some things that did not work as expected include :

1. The wheel encoders were error prone. For instance, a single movement might create more revolutions than expected due to braking distance. A subsequent movement in the opposite direction would then be of a smaller magnitude than expected since the wheel would be pre-rotated.
2. Last minute technical difficulties caused us to be unable to flash code from the pi to the arduino and push code from the pi to github via the command line.

Delegation of Work

We learnt the importance of assigning tasks or roles to each team member. While our roles may eventually change, the act of distributing labour helps create a sense of direction for the team. This is helpful in getting things done.

Appendix

Colour Detection:

```
// Def for Colour sensing pins
#define PinOut (1 << 4) // OUT, PL4, 39
#define S0 (1 << 0) // PL0, 35
#define S1 (1 << 1) // PL1, 36
#define S2 (1 << 2) // PL2, 37
#define S3 (1 << 3) // PL3, 38
int Red = 0;
int Blue = 0;
int Green = 0;

void sendColour();

struct RGBColour {
    int r;
```

```

int g;
int b;
};

struct RGBColour colours[4] = {
    { 255,255,255 }, // White
    { 255, 0, 0 }, // Red
    { 0, 255, 0 }, // Green
    { 0, 0, 255 }, // Blue
};

void ColourSetup() {
    DDRL = (S0 | S1 | S2 | S3) & ~PinOut;
    PORTL |= (S0 | S1);
}

void getColour(TPacket *colourPacket) {
    // Shine red S2, S3: L L
    PORTL &= ~(S2);
    PORTL &= ~(S3);
    while (!(PINL & PinOut))
        ;
    curr_time = micros();
    while (PINL & PinOut)
        ;
    Red = micros() - curr_time;
    Red = map(Red, 70, 45, 10, 245);
    delay(20);

    // Shine blue S2 S3 L H
    PORTL |= S3;
    while (!(PINL & PinOut))
        ;
    curr_time = micros();
    while ((PINL & PinOut))
        ;
    Blue = micros() - curr_time;
    Blue = map(Blue, 55, 78, 180, 10);
    delay(20);

    // shine Green, S2 S4 H H
    PORTL |= S2;
    while (!(PINL & PinOut))

```

```

;
curr_time = micros();
while ((PINL & PinOut))
;
Green = micros() - curr_time;
Green = map(Green, 55, 78, 245, 10);
delay(20);
int ColourDist = 32767;
int ClosestColour = -1;
for (int i = 0; i < 4; i++) {
    int distance = sqrt(pow(Red - colours[i].r, 2) + pow(Green - colours[i].g, 2) + pow(Blue -
colours[i].b, 2));
    if (distance < ColourDist) {
        ColourDist = distance;
        ClosestColour = i;
    }
}
switch (ClosestColour) {
case 0: // White
    colourPacket->params[0] = 3;
    colourPacket->params[1] = Red;
    colourPacket->params[2] = Green;
    colourPacket->params[3] = Blue;
    break;
case 1: // Red
    colourPacket->params[0] = 0;
    strcpy(array1, "Injured");
    strcpy(array2, "Rescuing");
    break;
case 2: // Green
    colourPacket->params[0] = 1;
    strcpy(array1, "Healthy");
    strcpy(array2, "Rescuing");
    break;
case 3: // Blue
    colourPacket->params[0] = 2;
    break;
}
}

```

I2C LCD Implementation:

```

void lcdplay() {

```

```

// For array1
lcd.setCursor(0, 0);    // set the cursor to column 0, line 0
if (sizeof(array1) > 17) { // If more than 16 characters, scroll
    lcd.setCursor(15, 0); // set the cursor to column 15, line 0
    for (int positionCounter1 = 0; positionCounter1 < sizeof(array1) - 1; positionCounter1++) {
        lcd.scrollDisplayLeft();    //Scrolls the contents of the display one space to the left.
        lcd.print(array1[positionCounter1]); // Print a message to the LCD.
        delay(tim);                //wait for 250 microseconds
    }
} else {                // If 16 or less characters, display directly
    lcd.print(array1); // Print a message to the LCD.
}
delay(tim); //wait for 250 microseconds
lcd.clear(); //Clears the LCD screen and positions the cursor in the upper-left corner.

// For array2
lcd.setCursor(0, 1);    // set the cursor to column 0, line 1
if (sizeof(array2) > 17) { // If more than 16 characters, scroll
    lcd.setCursor(15, 1); // set the cursor to column 15, line 1
    for (int positionCounter = 0; positionCounter < sizeof(array2) - 1; positionCounter++) {
        lcd.scrollDisplayLeft();    //Scrolls the contents of the display one space to the left.
        lcd.print(array2[positionCounter]); // Print a message to the LCD.
        delay(tim);                //wait for 250 microseconds
    }
} else {                // If 16 or less characters, display directly
    lcd.print(array2); // Print a message to the LCD.
}
delay(tim); //wait for 250 microseconds
lcd.clear(); //Clears the LCD screen and positions the cursor in the upper-left corner.
}

```

References

https://aminoapps.com/c/league-of-legends/page/blog/what-is-scripting/72UP_u58b0pgdm4P5zrdvJpqjDBDIw
<https://github.com/jiajunchang2002g/cg2111a.git>