

CG1112 Engineering Principle and Practice

Semester 2 2020/2021

“Alex to the Rescue”

Final Report

Team: B04-6B

Name	Student #	Sub-Team	Role
Tan Rui Yang	A0219814B	Firmware/Hardware	Firmware leader
Tan Xing Jie	A0217711M	Hardware/Software	Hardware leader
Shyam Ganesh	A0218448X	Software/Firmware	Software leader

Content Page

Content Page **1**

Section 1 Introduction **2**

Section 2 Review of State of the Art **2**

2.1. Walk-Man Robot 2

2.2. RAPOSA Robot 3

Section 3 System Architecture **4**

Section 4 Hardware Design **5**

4.1. RPI 3 Model B 5

4.2. Arduino Uno 5

4.3. RPI Lidar 5

4.4. Wheels 6

4.5. Colour Sensor (TCS34725 RGB Colour Sensor) 6

4.6. Ultrasonic Sensor (HC-SR04 Ultrasonic Sensor) 6

4.7. Speaker (Grove Speaker) 6

Section 5 Firmware Design **8**

5.1. High Level Algorithm on Arduino Uno 8

5.2. Communication Protocol 8

5.3. Colour Sensor Implementation 8

5.4. Ultrasonic Sensor Implementation 9

5.5. Speaker Implementation 9

Section 6 Software Design **10**

6.1. Teleoperation 10

6.2. Colour Detection 11

6.3. ROS on Multiple Machines 11

Section 7 Lessons Learnt - Conclusion **12**

References **13**

Section 1 Introduction

According to the International Federation of Red Cross and Red Crescent Societies, the number of people killed by natural disasters each year from 2009 to 2019 ranged from 14,389 to 314,503[1]. In order to minimise such deaths, many parties have been working together, and one of the tools that has gradually been implemented and used in recent years is search and rescue robots.

Our project, Alex, is a remotely controlled search and rescue robot that aims to minimise the loss of lives by retrieving relevant and crucial information of survivors in affected areas in order to facilitate search and rescue missions. Alex is able to be tele-operated from a laptop, receive instructions from the operators and transmit the surroundings (using Lidar) and the objects (via detection of colours) data back to the operators. Such features means that Alex will be able to navigate and aid the operators to search for survivors even in an unknown environment.

Section 2 Review of State of the Art

A standard **S&R robot** must be able to scale rough terrains and climb stairs to operate in disaster sites, hence track based robots are used which provide greater stability and manoeuvrability. RGB cameras are used for tele-operation of the robot and the robot has a number of additional sensors mounted to it. For outdoor activities a GPS module is attached to the sensor rig. For GPS restricted indoor scenarios depth and ranging sensors such as 3D and 2D Lidars are equipped. The evaluation of the sensors and the movement of the actuators are enabled using the Robot Operating System (ROS) as a high level API. For 2D and 3D map generation, the cartographer algorithm is used. The cartographer algorithm detects and calculates online loop closure with graph optimisation hence reducing translational and rotational errors when generating the map for the environment.[2]

2.1. Walk-Man Robot

The Walk-Man is a humanoid search and rescue robot developed by the Istituto Italiano di Tecnologia (IIT) in Genova, Italy. Standing at over 6ft the Walk-Man is equipped with over 32 engines and control boards, a 3D laser scanner, a microphone and chemical sensor which allows

it to map out its surroundings. Weighing just over 22lbs the Walk-Man's lightweight allows for it to maneuver through tough and uneven terrains with relative ease. The Walk-Man is still in the testing phase and has been able to navigate through damaged rooms, control gas leaks, remove debris and even put out fires. Currently Walk-Man is teleoperated, but researchers hope that one day Walk-Man or a similar humanoid robot could search through wreckage autonomously.[3]



Figure 1. Walk-Man Robot [4]

2.2. RAPOSA Robot

The **RAPOSA robot** was developed by a consortium led by IDMind along with University of Southern California. It is a semi-autonomous robot whose front body uses one thermal camera and two web cameras. Additional sensors include gas, temperature and humidity sensors, webcams, light diodes, microphone and loudspeaker. The robot uses wireless communication with an option for tethered operation. The robot was tested in multiple scenarios at the Fire Fighters school and it was found that the robot reduced inspection time by upto 25 percent as compared to trained firefighter teams.[5]

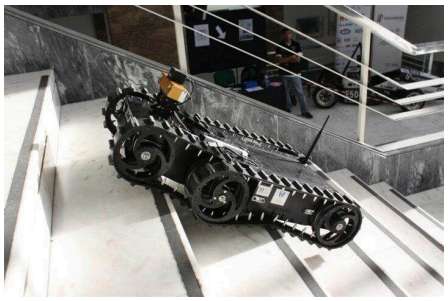


Figure 2. RAPOSA robot [6]

A weakness in using robots for search and rescue operations is that the operators of the S&R robots must be extensively trained to use the GUI which takes time and money. The extent to

which the rescue robots are effective in real time search and rescue operation depends upon its hardware peripherals like sensors and scanners, this adds to the overall cost of the robot. S&R robots also need constant maintenance which may add to more overhead costs. Rescue robots also lack the ability to function like a human, which may be a hindrance when responding to real time emergencies.

Section 3 System Architecture

The diagram below shows how the components of Alex are integrated together in order for Alex to function properly.

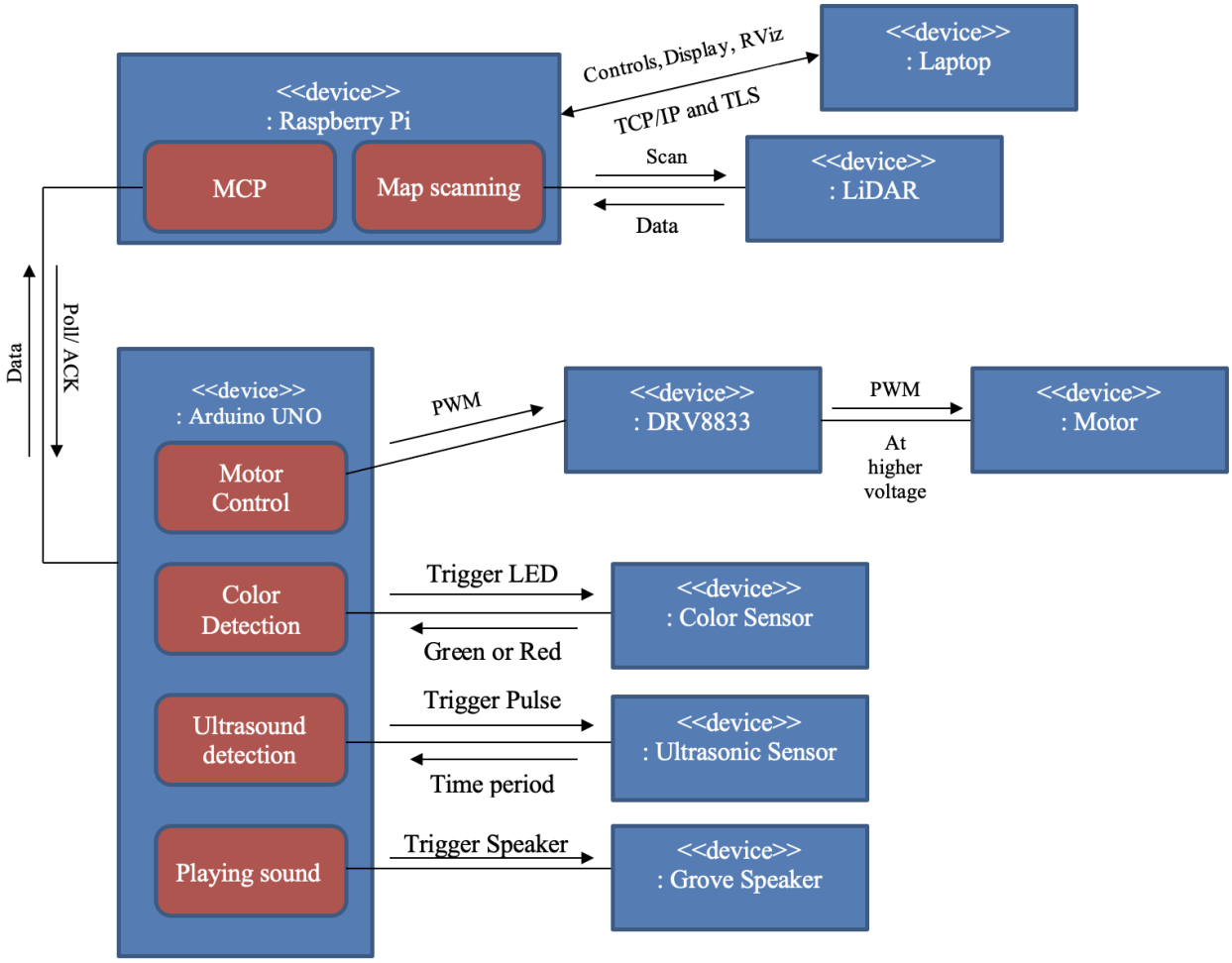


Figure 3. Diagram of the Overall System Architecture

More details regarding how the components interact with one another can be found in Section 4 Hardware Design.

Section 4 Hardware Design

In this section, we will discuss the main hardware components of Alex.

4.1. RPI 3 Model B

Raspberry Pi (Pi) is a microprocessor that is similar to typical computers. In order to remotely control Alex, the operators will SSH into Pi from a laptop. The Pi will then be used to collect data from the Lidar unit and create a virtual map of the environment. The operators can view this "virtual map" via the laptop that was used to SSH into Pi. On top of that, Pi contains the master control program (MCP) which is used to translate movement instructions sent by the operators to the Arduino board via serial communication in order to navigate Alex in the simulated environment.

4.2. Arduino Uno

Atmega328p, the microcontroller of the Arduino UNO is used to control the two motors on Alex wheels which then enables the robot to traverse through the simulated environment. As stated in the project requirement, Alex will be able to move straight, back, turn left and right. The Arduino is also used for ultrasonic detection to measure the distance between Alex and the obstacles, and colour detection of the colour sensor for the 'Miss Scarlet is Alive' subtask in order to identify the colour of the object correctly. Moreover, we have also incorporated a speaker which plays a sound when a survivor is detected, and we thus use the Arduino UNO to control when the sound is played.

4.3. RPI Lidar

The RPI Lidar is a 2D laser scanner with a rotating 360 degrees mount, which is used to scan and map out the surrounding environment. The Lidar unit sends out a light pulse and waits for the pulse to be reflected off any surface before it is received by the detection unit inside the

Lidar. The time taken for the light to be transmitted and received can be measured to find the distance between the Lidar unit and the surface the light is reflected off. In our project, we have decided to incorporate the Hector SLAM algorithm to map out the surrounding environment. The Hector SLAM algorithm is rather ingenious that it does not require odometry data. This is possible because the algorithm uses new information it has received from the Lidar and tries to match it with the pre-existing map, this process allows the algorithm to find the current location of the robot without the use of odometry data.

4.4. Wheels

The wheels of Alex enable it to move. When a movement command is sent to Arduino, the left and right motors will execute the command and move the wheels in the correct directions according to the command. The distance moved by the wheel is calculated by using the wheel encoders. Each wheel encoder calculates the number of times the wheel has rotated. Each encoder consists of two parts, a hall effect sensor which measures the strength of the magnetic field and a ring magnet. The ring rotates along with the wheel and causes a change in the magnetic field which is detected by the hall effect sensor positioned close to the ring.[7] This process thus enables the sensor to count how many times the wheel has rotated.

4.5. Colour Sensor (TCS34725 RGB Colour Sensor)

Colour sensor connected to Arduino is used to enable Alex to identify the survivor. By measuring the 8-bits Red(R), Green(G), and Blue(B) values of the light detected. Arduino can identify the colour detected according to the different ratio of R, G and B values measured. For instance, since red-coloured object will cause the R value received to be higher than the others, Arduino can then detect red colour by using this characteristic. Different colours have different R, G, B values. Hence the colour sensor can detect a lot of colours by setting different range values of R, G and B. The range value for R, G and B for different colours are found by trial-and-error.

4.6. Ultrasonic Sensor (HC-SR04 Ultrasonic Sensor)

Ultrasonic sensor is used to measure the distance between Alex and the obstacles (including walls and survivors). This is important because it can help to prevent Alex from hitting the walls or survivors while moving forward or turning. The ultrasonic sensor sends back

the distance information to Arduino. With this information, Alex is able to stop and avoids bumping into the obstacles.

4.7. Speaker (Grove Speaker)

The speaker is one of the additional features we have added to Alex. It is used to generate sound upon detecting the red-coloured object. Alex being a small and compact search and rescue robot cannot help survivors escape from the rubble in a disaster site. Hence, we use the sound to guide the rescuers to the survivor's exact location, saving precious time which could potentially help to rescue the survivor.

In order to make Alex as compact as possible, our group has decided to arrange our hardware components as follows.

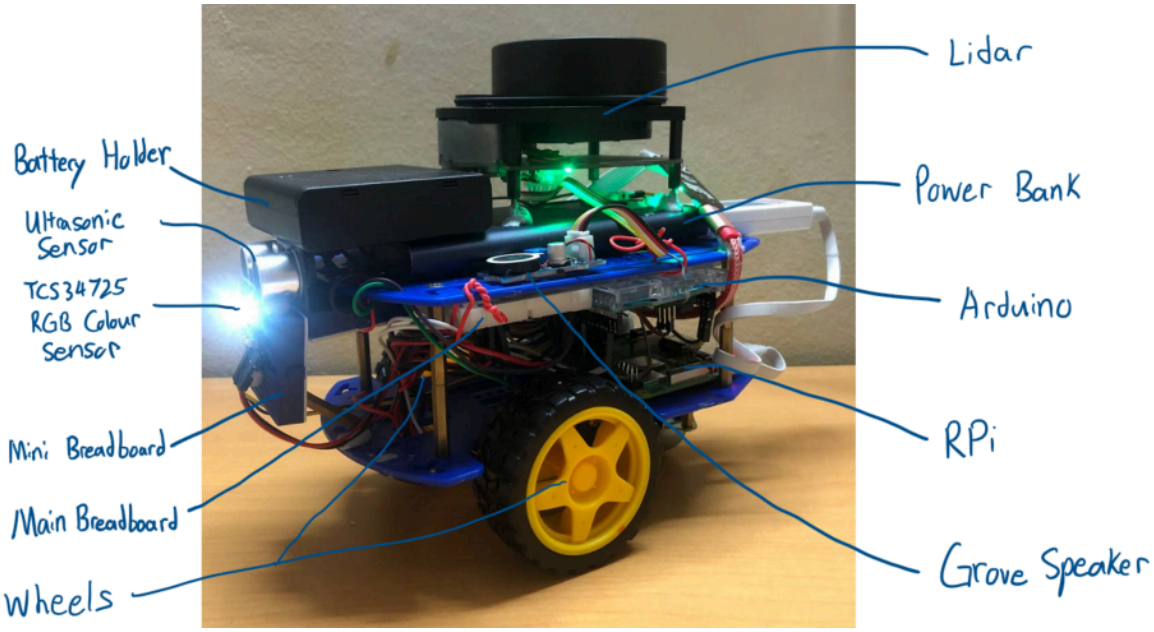


Figure 4. Components on Alex

The Arduino, RPI and main breadboard are placed in between the two boards, allowing us to manage the cables more neatly and ensuring that most of the wires stay within the body of Alex. The power bank, battery holder, speaker and Lidar are mounted on the top board, while the ultrasonic sensor and light sensor are mounted on smaller breadboards at the front of Alex. With this design, not only did we try to compact Alex as much as possible, but we are also able to keep the weight of Alex fairly distributed, ensuring that it will not topple over easily.

Section 5 Firmware Design

In this section, we will talk about how the Arduino is used to control the behavior of Alex.

5.1. High Level Algorithm on Arduino Uno

When Arduino Uno receives command packets sent from the RPI, it will execute the corresponding instruction being sent. There are 6 commands available for the user: ‘f’ for moving forwards, ‘b’ for moving backwards, ‘l’ for turning left, ‘r’ for turning right, ‘g’ for getting current status of Alex (encoder ticks value and distance travelled), ‘c’ for clearing all the statuses. The operator can send a particular command in the form of a single character from the laptop to the RPI via the internet. The RPI will send this character together with other information such as desired distance to travel, turning angle and motor speed to Arduino. After which, the Arduino identifies this character and executes the corresponding instructions it received.

5.2. Communication Protocol

In order to control the Arduino using the RPI, we need to set up a communication protocol. This means that we are transferring data from RPI to Arduino and also from Arduino to RPI. We need to decide the baud rate, length of the data, parity bit and stop bit to allow both the RPI and Arduino to communicate via serial lines. In our project, the setup is 9600 8N1, which means that the baud rate is 9600 bits per second, data length is 8 bits, parity bit is N, and stop bit is 1. We also need to make sure that the endianness and data sizes between Arduino and RPI are standardized. The ‘char’ data type in Arduino is only 1 byte but it is 4 bytes in RPI, thus we need to pad 3 bytes of empty ‘char’ data when we send any information from Arduino to RPI. [8]

5.3. Colour Sensor Implementation

To use the library function, we first include the “Adafruit_TCS34725.h” library.

```
#include <Adafruit_TCS34725.h>
```

For colour sensing, we get the R, G, B values of the light being reflected back by the object we need to sense. We declare a structure called “Adafruit_TCS34725”so that we can use the function corresponding to the structure. Then we use the “getRGB()” function to get the R, G, B values. We call the function “getRGB()” three times and take the average to reduce the error of R, G and B colour values. [9]

```
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);
#define RED 125
int colour = 0;

void scanColour(){
  float R[3], G[3], B[3];
  for (int i = 0; i < 3; i ++){
    tcs.getRGB(&R[i], &G[i],&B[i]);
  }
  float Ravg, Gavg, Bavg;
  Ravg = (R[0] + R[1] + R[2]) / 3;
  if (Ravg > RED){
    colour = 1;
  }
  else{
    colour = 2;
  }
}
```

5.4. Ultrasonic Sensor Implementation

For distance measuring, we first connect the ultrasonic sensor to pin 9 and pin 8 and store the distance information in the variable “distance”. With this distance information, we can decide when to stop Alex from moving. In our code we decided to stop the motors if the distance returned by the ultrasonic sensor was less than or equal to 4cm.

```
#define TRIGPIN 9
#define ECHOPIN 8
int distance;
void setupUltra(){
  //pinMode(TRIGPIN, OUTPUT);
  DDRB |= 8b00000010;
  //pinMode(ECHOPIN, INPUT);
  DDRB &= 8b11111110;
}
```

```
long ultra_detect(){
  long duration, distance;
  //digitalWrite(TRIGPIN, HIGH);
  PORTB |= 8b00000010;
  delay(1);
  //digitalWrite(TRIGPIN, LOW);
  PORTB &= 8b11111101;
  duration = pulseIn(ECHOPIN, HIGH);
  distance = (duration/2)/29.1;
  return distance;
}
```

```
if (distance <= 4){
  deltaDist = 0;
  newDist = 0;
  stopMovement();
  delay(500);
  scanColour();
  if (colour == 1){
    song3 = 1;
    while(song3){
      sing(2);
      song3 = 0;
    }
  }
```

5.5. Speaker Implementation

To enable the speaker to play music, we first define some musical notes' frequency. Then we create 2 arrays to change the notes according to the tempo.

```
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_C4  262
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_C5  523
#define NOTE_D5  587

#define melodyPin 4
```

```
//Underworld melody
int underworld_melody[] = {
  NOTE_C4, NOTE_C5, NOTE_A3, NOTE_A4,
  NOTE_AS3, NOTE_AS4, 0,
  0,
  NOTE_C4, NOTE_C5, NOTE_A3, NOTE_A4,
  NOTE_AS3, NOTE_AS4, 0,
  0
};
```

```
int underworld_tempo[] = {
  12, 12, 12, 12,
  12, 12, 6,
  3,
  12, 12, 12, 12,
  12, 12, 6,
  3
};
```

After setting the speaker output pin (pin 3), the speaker can play the music by calling the functions "sing()" and "buzz()". [10]

```
int song3 = 0;
int song = 0;
void sing(int s) {
  song = s;
  if (song == 2) {
    //play super mario underworld theme!
    int size = sizeof(underworld_melody) / sizeof(int);
    for (int thisNote = 0; thisNote < size; thisNote++) {
      int noteDuration = 1000 / underworld_tempo[thisNote];
      buzz(melodyPin, underworld_melody[thisNote], noteDuration);
      int pauseBetweenNotes = noteDuration * 1.30;
      delay(pauseBetweenNotes);
      buzz(melodyPin, 0, noteDuration);
    }
  }
}
```

```
void buzz(int targetPin, long frequency, long length) {
  long delayValue = 1000000 / frequency / 2; // calculate
  long numCycles = frequency * length / 1000; // calculate
  for (long i = 0; i < numCycles; i++) { // for the calcul
    digitalWrite(targetPin, HIGH); // write the buzzer pin
    delayMicroseconds(delayValue); // wait for the calcula
    digitalWrite(targetPin, LOW); // write the buzzer pin
    delayMicroseconds(delayValue); // wait again or the ca
  }
}
```

Section 6 Software Design

The Pi sends the instructions in the form of packets which are then serialized and sent to the Arduino UNO which then deserializes the sent packet and carries out the specific action

specified by the teleoperator. Similarly, when the Arduino sends any information back to the user it does so by following a similar set of steps as specified above. We also employ measures like checksums and check the packet length being received by comparing it with packet length which was sent, this enables us to examine if the packets sent via the serial lines are received correctly, this enables us to efficiently send information from one system to another without much data corruption.

6.1. Teleoperation

Operating the search and rescue robot is done by a remote teleoperator. The operator sends a specific set of instructions, i.e., the direction, distance and speed at which to move the robot. As was explained earlier the instruction is received by the Pi and is then sent to the Arduino UNO to carry out the specific task (in this case movement). The operator can also ask the robot for its current status which may include information like the amount of distance travelled, and the relevant information is then sent by the Arduino to the operator via the Pi. There are 2 ways we can deploy to send instructions to the RPI. One method is to simply SSH and control Alex directly from RPI using VNC viewer. Another way is to send the instructions from our laptop to the RPI using the internet. In this project we use the second method. We first set up the Transport Control Protocol (TCP) to enable data transmission. We also use Transport Layer Security (TLS) to safely transfer data from the sender to the receiver, this is enabled by signing up a certificate authority key (CA) to only enable devices that have the certificate to send and receive information from the RPI. This prevents any hackers to hijack our Alex and take control of him. [11]

6.2. Colour Detection

The colour detection process in the project simulates a situation where the search and rescue robot needs to detect survivors. As was explained earlier, all the data processing for the colour sensor is taken care of by the Arduino UNO. The colour detected by the Arduino is then sent to the RPI in the form of an integer where ‘1’ indicates the object being detected is red and ‘2’ indicates that it is green. The teleoperator can get this information by calling the get stats feature.

```
printf("Colour (RED = 1, GREEN = 2):\t\t%d\n", packet->params[9]);
```

6.3. ROS on Multiple Machines

The ability to use Hector SLAM for mapping the environment is enabled by ROS (robotic operating system). ROS is an open-source meta-operating system which provides us

tools and libraries to send and receive information from multiple submachines. In our project we decided to run the Rviz application which maps out the environment locally on our laptop, this was done to reduce the load on the RPI and to save power. ROS running on multiple machines uses the master slave architecture. In our case the local laptop running Ubuntu 16.04LTS was the master and the RPI was the slave. The Lidar connected to the RPI scans the data and sends this information to the laptop. The information being received is then processed using the Hector SLAM algorithm and the map is then displayed on the Rviz mapping application.[11]

Section 7 Lessons Learnt - Conclusion

This project aimed to create a search and rescue robot, Alex, that can navigate and detect survivors in a simulated environment.

One of the difficulties that we encountered over the course of this project, is that we are not able to make some of our hardware components function properly. For instance, our group faced a problem with the wheel encoders. We were not able to get it to count the number of turns the motor has rotated properly, hence the distance Alex moved was not very accurate. One other problem we had was with our colour sensor. We were unable to get it to function and detect colours properly when we combined the colour sensor's Arduino code with Alex's Arduino code. However, we managed to resolve both issues after relooking at the materials from our labs and information found online.

From this mistake, our group has learned that it is very important to fully understand how the components work before implementing it in our project. For the wheel encoder, our group had initially forgotten to configure the pull-up resistor in Arduino, hence the wheel encoder could not detect the rotations properly. For the colour sensor, we disabled the I2C pin that was required for the TCS34725 RGB Colour Sensor to sense colour, while attempting to reduce power in Week 9 Studio 2. These two mistakes clearly show that we did not fully understand how the components work initially. Only after relooking in the lab materials and searching for information online, we were able to debug the problems with both components. Hence, one of the takeaways from our group is that we should always fully understand how a component is supposed to work first before rushing to implement it in any projects.

Additionally, another difficulty that our group faced while doing this project was that our Arduino would not receive the commands properly at times. As a result, Alex could not be remotely controlled anymore. After numerous trials, we found out that this was because when the command was sent, Arduino was busy executing another piece of code, and hence it was unable to receive the command properly, resulting in a "bad checksum" error. Also, Arduino tends to stop functioning properly when we send consecutive commands too quickly.

To resolve this issue, our group tried to ensure that polling does not occur in our Arduino code and not to send consecutive commands too quickly to Arduino. This mistake clearly highlights the limitations that hardware components could have, and the importance to understand such limitations when we are using the hardware components to overcome the limitations and make the project successful.

References

1. Jon Walker, published at 21 November 2019, "**Search and Rescue Robot, Current Applications on Land, Sea and Air**", https://emerj.com/ai-sector-overviews/search-and-rescue-robots-current-applications/

2. Georg A. Novotny, Simon Emsenhuber, Philipp Klammer, Christoph Pöschko, Florian Voglsinger & Wilfried Kubinger (2019). "**A MOBILE ROBOT PLATFORM FOR SEARCH AND RESCUE APPLICATIONS**". Published by DAAAM International, ISBN 978-3-902734-22-8, ISSN 1726-9679, Vienna, Austria. https://www.daaam.info/Downloads/Pdfs/proceedings/proceedings_2019/131.pdf

3. Kerrie Brooks, published 14 September 2018, **"Walk Man has Entered the Building"**, https://www.distrelec.de/current/en/robotics/5-inspirational-search-and-rescue-robots/

4. Adapted from, EurekAlert, Published 22 February 2018, https://www.eurekalert.org/pub_releases/2018-02/iidt-hrw022118.php

5. Adapted from, Wikipedia, published 4 January 2020, **"RAPOSA"**, https://en.wikipedia.org/wiki/RAPOSA

6. Adapted from, SPARC, published 25 May 2014, **"RAPOSA NG - a search and rescue land wheeled robot"** https://www.eu-robotics.net/sparc/success-stories/raposa-ng-a-search-and-rescue-land-wheeled-robot.html?changelang=2

7. Michael Frontz, published at 2019, "**Wheel Encoders**", available here: https://docs.idew.org/code-robotics/references/physical-inputs/wheel-encoders

8. Colin Tan Keng Yan, published on 28 February 2021, "**Week 7 Studio - 2 Lecture Notes: Communication Protocol**" pg3, 13, 20 - 31.

9. Kevin Townsend, published at 10 April 2013, "**Adafruit TCS34725**", https://github.com/adafruit/Adafruit_TCS34725

10. Jorge Rance, published at 4 December 2017, "**Super Mario Theme Song for Arduino**", https://create.arduino.cc/projecthub/jrance/super-mario-theme-song-w-piezo-buzzer-and-arduino-1cc2e4

11. Colin Tan Keng Yan, published on 5 March 2021, "Week 8 Studio 2 - Securing Alex to the Cloud" pg 18-21, 44 - 47.

12. Austin Hendricks, published at 20 March 2019, **"Running ROS across multiple machines"** http://wiki.ros.org/ROS/Tutorials/MultipleMachines