CG2111A Engineering Principle and Practice
Semester 2 2022/2023

"Alex to the Rescue"
Final Report
Team: B03-3A

Name	Student #	Main Role
Mohamed Ijaaz s/o Mohamed Ismail	A0258417X	Firmware
Lui Ting Mun, Sharlyn	A0257393U	Software
Ymir Meddeb	A0265192B	Hardware
Ryan Loh Yong Rui	A0258299H	Software

Section 1 – Introduction

For this project we were tasked with creating a robot, Alex that could carry out a simulated search and rescue. In order to successfully complete the search and rescue Alex needed to be able to map out the simulated environment it was put inside (which we would then make a sketch of and submit) with the help of the LiDAR and using Hector SLAM, navigate from one end of the environment to the other within 8 minutes, successfully identify victims (represented by green and red targets) using a colour sensor while also not misidentifying any dummy victims. All of this needed to be done remotely, meaning we would be controlling Alex via teleoperation and we would need to avoid hitting any walls. We were also asked to include additional features on Alex and our group decided to use an ultrasonic sensor to aid our search and rescue by informing us when we were too close to a wall as well as allow us to have fine control over the distance between Alex and the victims so that we could get an accurate reading from the colour sensor. In the sections that follow we will be going in-depth on each aspect of Alex (Hardware, software etc.) and our thought process as well.

Section 2 – Review of State of the Art

Earthshaker

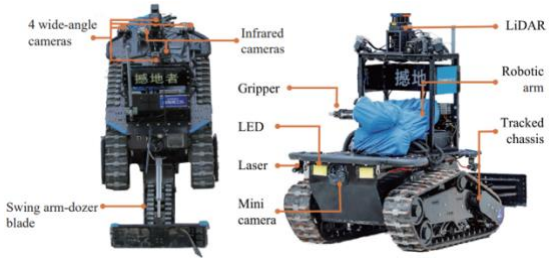


Figure 2.1

The Earthshaker is a mobile rescue robot designed for search and rescue missions through teleoperation and autonomous navigation. The Earthshaker utilizes wide-angle cameras, infrared cameras, LiDAR and RGBD camera, to map out areas, allowing the operator to control the robot remotely and avoid obstacles accordingly. The four wide-angle cameras mounted on the sides provide a panorama view of the surroundings while the infrared cameras aid in navigation in smoky environments. To ensure reliable remote communication, the Earthshaker enables multimodal teleoperation which includes MIMO-mesh radios, an extra relay radio mounted and a 4G/5G router.

The Earthshaker strengths include the functionality of its several maneuvering apparatuses which includes a swing arm-dozer blade that can be extended to increase the robot length to help cross wide trenches. However, the Earthshaker falls short of flexibility in movement due to its size, causing it to be hard to navigate through narrow spaces. Additionally, it also does not perform as well at tasks which require it to work with heavy loads to the limitation of the payload to the manipulator.

Snake Robot (Kulko)

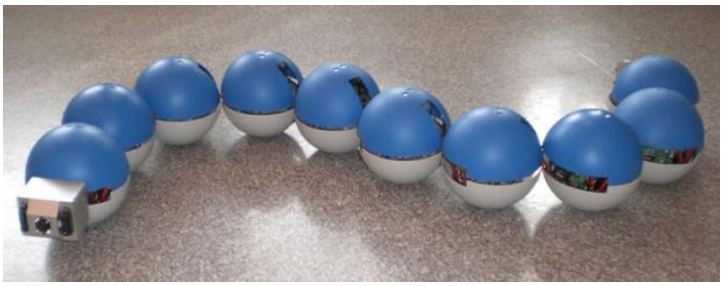
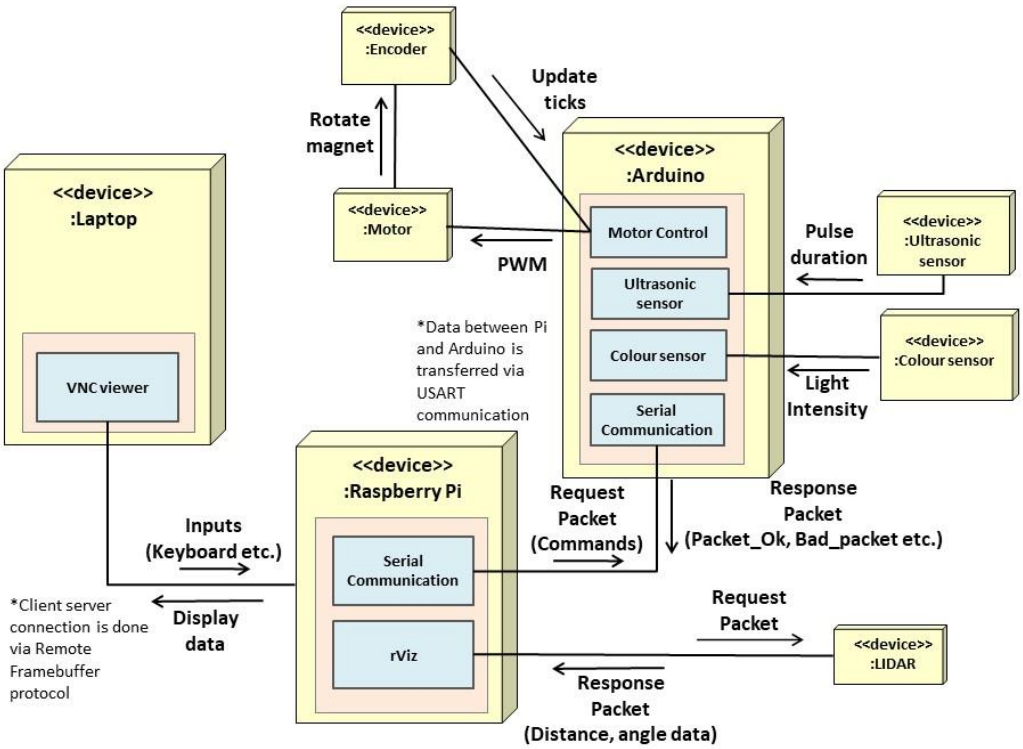


Figure 2.2

The snake robot design is inspired by biological snakes' mobility and was designed to navigate through uneven and cluttered grounds. The robot consists of identical joint modules with contact force sensors for the robot to sense the environment and the spherical design gives the robot a smoother exterior to allow for slithering motion in cluttered environment. The infrared distance sensors, camera, force sensors, angle sensors and motors in the joints of Kulko are connected to a microcontroller and can be controlled by the operator through Bluetooth.

Kulko strengths include its ability to display different motion patterns by controlling each joint independently to help navigate through complex environments effectively. Such motion includes lateral rolling and sidewinding. The limitation of Kulko is the need for sensors to completely cover the joint link as it senses the environment based on direct measurement of the external forces acting on each link which requires extensive use of sensors and subjects the sensors to being damaged by the environment.

Section 3 – System Architecture

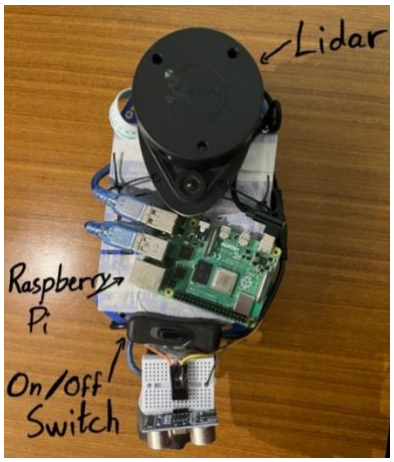


Section 4 – Hardware Design

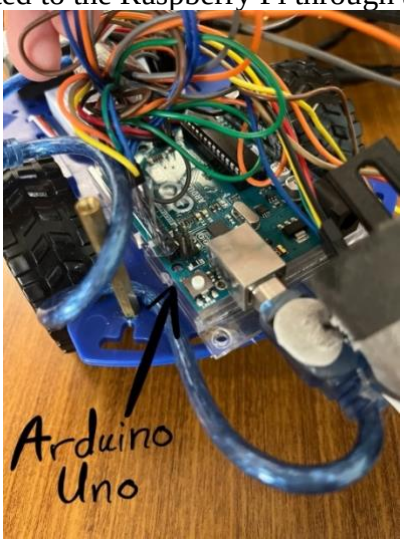
Important hardware components:

1. LiDAR: A Light Detection and Ranging (LiDAR) sensor is used for measuring distances and creating a 2D map of the robot's surroundings. The LiDAR is mounted on top of the robot's chassis and connected to the Raspberry Pi through a serial port.

2. Raspberry Pi: A Raspberry Pi microcontroller is used to process the data from the LiDAR sensor and send commands to the Arduino Uno.

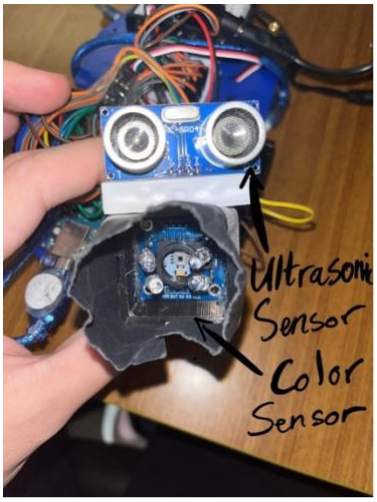


1. Arduino Uno: An Arduino Uno microcontroller is used to receive commands from the Raspberry Pi and control the motors, the color sensor and the ultrasonic sensor. The Arduino Uno is connected to the Raspberry Pi through a serial port.

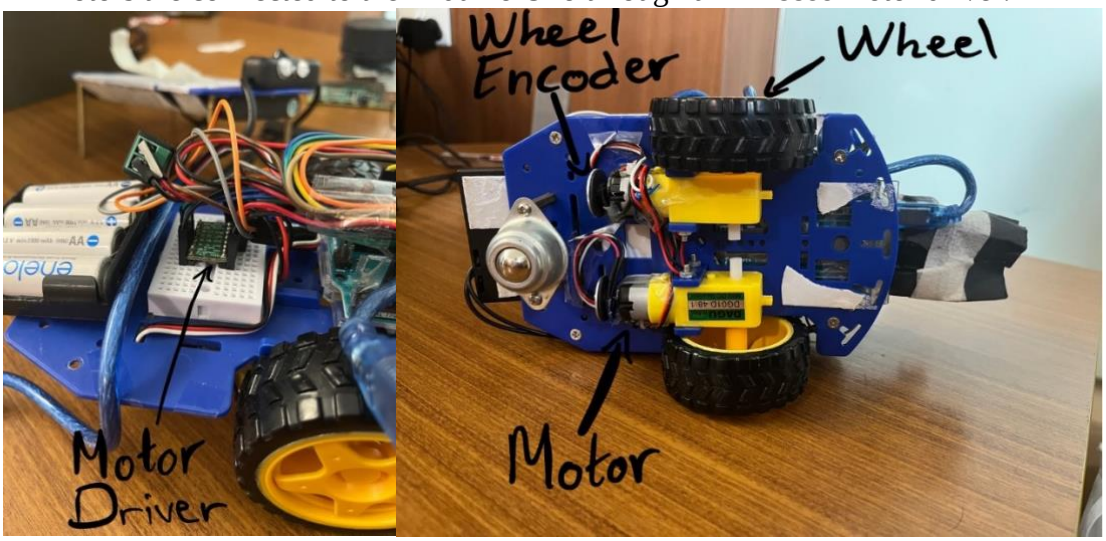


2. Color Sensor: A color sensor is used to detect colors of the victims in the maze. The sensor is placed on the front of the robot and is connected to the Raspberry Pi through a digital port.

3. Ultrasonic Sensor: An ultrasonic sensor is used to detect the distance between the robot and the victim so that the color sensor is more accurate. The sensor is placed on the front of the robot and is connected to the Raspberry Pi through an analog port.



4. Motors: The robot uses two motors to move forward, backward, right, or left. The motors are connected to the Arduino Uno through a DRV-8833 motor driver.



Section 5 – Firmware Design (Arduino)

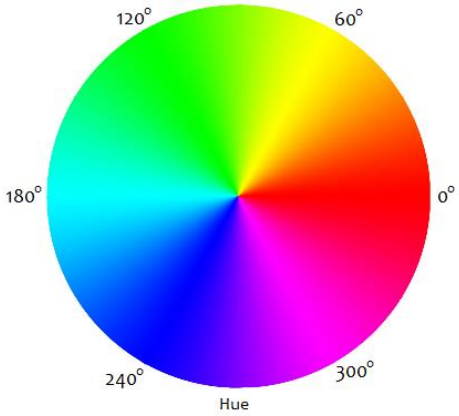
High Level Steps:

1. **High level Algorithms on the Arduino**
    1. **Bare Metal Programming**
    2. **Color detection**
    3. **Distance detection (Ultrasonic sensor)**
2. **Communication Protocols**
    1. RPi transmitting data to the Arduino
    2. Arduino transmitting data to the RPi

5.1.1 Bare Metal Programming

In order to create a efficient and faster robot which is essential in time sensitive operations such as a search and rescue. We implemented bare metal programming for functions such as toggling pins from high to low , initialization of pins to input and output as well as converting the analogWrite function to bare metal. (Full Bare metal code in Appendix 5.1) As seen from the code in the appendix under 5.1 the forward and reverse functionality of the left wheel make use of timer 0 while for the right wheel the forward functionality uses timer 1B and the reverse functionality uses timer 2A. In order to stop the movement of all motors the TCCRnA registers for the respective timers were set to 0 as seen in the stop function.

5.1.2 Colour Sensor implementation



As the maze has red and green objects to detect. We will be using a GY-31 Colour sensor module to help with colour detection. Using the colour sensor, we will read the red, green and blue frequency signals using the 'pulseIn()' function.

Lighting environments are sensitive and R,G,B values tend to vary. Hence for ease of configuration and debugging, instead of determining the colours using the individual RGB values, we used the hue colour wheel which is calculated using an algorithm(refer to Appendix 5.2) which takes in the R,G,B values as inputs. The algorithm then returns a hue value that ranges from 0-360(degrees represented as type int).

If the hue value(degrees represented as integers) is below 15 or above 330, we will determine the colour as red and we will send a message back to the RPi with the message "Colour is red". Else if the hue is between 90 and 130 we will send a message "Colour is green". Else if the hue is any other value we will send a message "some other colour".

5.1.3 Ultrasonic Sensor implementation

Based on the duration between sending and receiving the ultrasonic wave. Using the speed of sound, we can then calculate the distance from the object using the formula below. (Full code in Appendix 5.3)

```
long duration = pulseIn(ECHO, HIGH, TIMEOUT); // microseconds
float currentCMS = duration * SPEED_OF_SOUND / 2 / 10000;
```

5.2 Communication Protocol

5.2.1 RPi communicating to the Arduino

Step 1: Initialisation

1. when the RPi is turned on it will first attempt to establish a connection with the Arduino by sending a packet of structure TPacket, which is the struct that we'll be using for communicating between the Arduino and the RPi. And the Arduino polls for the USART port to receive the packet.

```
typedef struct
{

    char packetType;
    char command; //direction(front, backward, left and right) or identify object
    char dummy[2]; // Padding to make up 4 bytes
    char data[MAX_STR_LEN]; // String data
    uint32_t params[16];
} TPacket;
```

```
do
{ //polling done by the arduino
  result = readPacket(&hello);
} while (result == PACKET_INCOMPLETE);
```

2. We will ensure the data received is complete by checking the checksum as well as the magic number. A packet with structure Tcomms is deserialised and the magic number and checksum will be verified as shown below.

```
typedef struct comms
{
    uint32_t magic;
    uint32_t dataSize;
    char buffer[MAX_DATA_SIZE];
    unsigned char checksum;
    char dummy[3];
} TComms;
```

```
// Check that we have a valid packet
if(comms->magic != MAGIC_NUMBER)
{
    printf("BAD MAGIC NUMBER. EXPECTED %x GOT %x\n", MAGIC_NUMBER, comms->magic);
    return PACKET_BAD;
}
```

```
if(checksum != comms->checksum)
    return PACKET_CHECKSUM_BAD;
else
```

3. If the packet was received successfully, a data packet containing the success message will be send back to the pi.
4. If the packet received was incomplete (incorrect checksum/magic number), a data packet containing the failure message will be sent back to the RPi.
5. Once the success message is sent and communication between the RPi and the Arduino is established, a message as such will be printed. And now we can now input user commands into the RPi to control the robot

```
ALEX REMOTE SUBSYSTEM

Opening Serial Port
ATTEMPTING TO CONNECT TO SERIAL. ATTEMPT # 1 of 5.
Done. Waiting 3 seconds for Arduino to reboot
DONE. Starting Serial Listener
Starting Alex Server
```

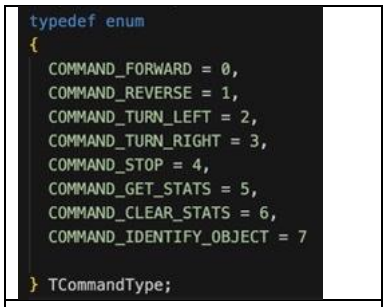
```
DONE. Sending HELLO to Arduino
DONE.
Now listening..
```

6. The SLAM algorithm will then be performed using data from the Lidar sensor and a mapping of the surroundings will be produced using the visualizer program rviz.

Step 2: Receive user command

1. The user can now input the command for the robot to perform (as mentioned in Section 6.1)
2. The RPi will receive the inputs and store them in a structure called TPacket(as seen in step 1). Since this is a command we wish the robot to perform, we will set the packet type as PACKET_TYPE_COMMAND.
3. This data is then serialised and sent to the Arduino for execution

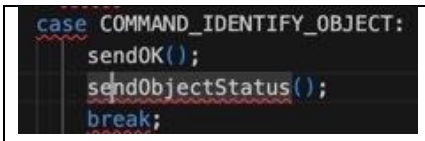
Step 3: Carrying out user command



```
typedef enum
{
  COMMAND_FORWARD = 0,
  COMMAND_REVERSE = 1,
  COMMAND_TURN_LEFT = 2,
  COMMAND_TURN_RIGHT = 3,
  COMMAND_STOP = 4,
  COMMAND_GET_STATS = 5,
  COMMAND_CLEAR_STATS = 6,
  COMMAND_IDENTIFY_OBJECT = 7

} TCommandType;
```

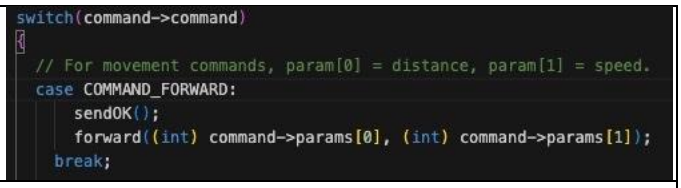
TCommandType constants

1. When the Arduino receives the data, it will first deserialize the data into the same structure of TPacket as seen in step 1. After deserialization, it reads the command that is to be performed(identify object/moving)



```
case COMMAND_IDENTIFY_OBJECT:
    sendOK();
    sendObjectStatus();
    break;
```

Command to identify object



```
switch(command->command)
{
  // For movement commands, param[0] = distance, param[1] = speed.
  case COMMAND_FORWARD:
      sendOK();
      forward((int) command->params[0], (int) command->params[1]);
    break;
```

Command to move forward

2. To move the robot
    1. An analog signal will then be sent to the respective pins to power the wheels as well as configure the wheels speed by setting its PWM frequency.
    2. As the wheels start moving, the encoder will get triggered which then in turn triggers the respective Interrupt Service Routines(ISRs), in these routines we will increment a variable until the required distance/angle is covered.
3. To identify the object (identify both the distance and the colour)
    1. In the function sendObjectStatus (Appendix 5.4), we will detect the distance using the ultrasonic and determine the colour using the GY-31 colour sensor.

5.2.2.Arduino transmitting data to RPi

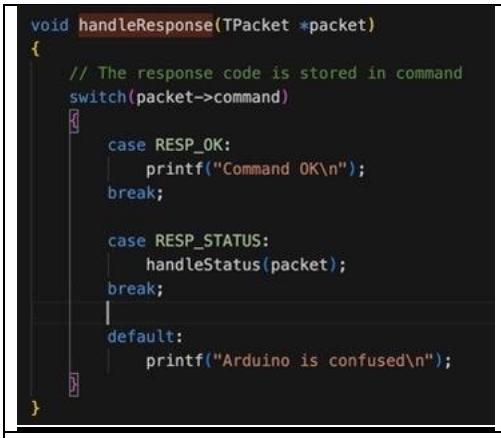
The Arduino will transmit data to the RPi via the TPacket structure(as seen in step 1 in 5.2). We mainly send information back from the Arduino to the RPi during these 3 scenarios

1. Sending success/failure messages to the RPi (sendOK, sendBadResponse, sendBadChecksum etc…)
2. Sending stats regarding the Arduino
3. Sending the colour of the identified object as well as the distance from the object.

For the first 2 cases we will be setting the packet type as PACKET_TYPE_RESPONSE. However for case 3. we will be setting the packet type PACKET_TYPE_MESSAGE.

Case 1 & 2

When we send a packet of type response, if we just want to indicate to the RPi that communication is successful, we set the command to RESP_OK. However if we wish to retrieve the stats of the Arduino we set the command to RESP_STATUS (Appendix 5.5) which the RPi will then print out the relevant stats onto the terminal(Appendix 5.6)

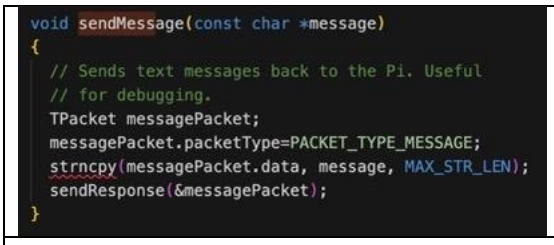


```
void handleResponse(TPacket *packet)
{
    // The response code is stored in command
    switch(packet->command)
    {
        case RESP_OK:
            printf("Command OK\n");
        break;

        case RESP_STATUS:
            handleStatus(packet);
        break;

        default:
            printf("Arduino is confused\n");
    }
}
```

Packet Received by the RPi

Case 3

When we send a packet of type message, we serialize whatever message in the data array of structure TPacket. Then message which is contained in the data array is then deserialized and printed out by the RPi



```
void sendMessage(const char *message)
{
  // Sends text messages back to the Pi. Useful
  // for debugging.
  TPacket messagePacket;
  messagePacket.packetType=PACKET_TYPE_MESSAGE;
  strncpy(messagePacket.data, message, MAX_STR_LEN);
  sendResponse(&messagePacket);
}
```

Serializing message on the Arduino

```
void handleMessage(TPacket *packet)
{
    printf("Message from Alex: %s\n", packet->data);
}
```

Printing the message on the RPi

Section 6 – Software Design (PI)

High Level Steps:

1. **Teleoperation**
2. **Hector Slam**

6.1 Teleoperation

6.1.1 Movement

Initially The user had to first input the direction for the robot (forward, backward, left and right) and then input 2 more inputs into the RPi, the distance/angles in cm/degrees and the power in %.

In order to save time we created ‘w’, ‘a’, ‘s’ and ‘d’ controls for ease of movement during the maze. For these cases the distance/angle and power were pre-set to default values.
Additionally we had the capitalised version of ‘w’, ‘a’, ‘s’ and ‘d’ for larger distances and angles **.(Refer to appendix 6.1)**

6.1.2 Identify Object

We added an additional command ‘i’ that will identify the colour of the object as well as the distance away from the object.

```
case 'i':
case 'I':
    commandPacket.command = COMMAND_IDENTIFY_OBJECT;
    commandPacket.params[0] = 0;
    sendPacket(&commandPacket);
    break;
```

A response in this format to this will be displayed onto the terminal.

```
Message from Alex: f ok
Message from Alex: handle packet
Message from Alex: hue is 33
Message from Alex: colour is green
Message from Alex: distance is 10cm
```

6.2 SLAM

For our Alex we had decided to use SLAM, more specifically Hector SLAM, to map out the maze Alex would traverse through. SLAM standards for Simultaneous Localization and Mapping and uses data provided from hardware components such as laser scanners to create a constantly updating map of its surroundings. Hector SLAM does not require odometry data and instead uses its distance from walls it has detected to estimate its relative position in the map.
For our group we ran both Hector SLAM and rviz (Our chosen visualization software) on the Raspberry Pi. We then remotely connected to the Pi using VNC viewer.

The process to set up and run hector SLAM and rviz on the Pi is as follows

```
1  #Open a new terminal and type te following
2  roscore
3  #Then open another terminal and type
4  roslaunch rplidar_ros view_slam.launch
5  #This will create a rplidar node on the raspberry
6  #pi as well as launch rviz
```

Furthermore we included the following lines

```
1 Source ~/cg2111a/devel/setup.bash
2 Source /opt/ros/noetic/setup.bash
```

into the .bashrc file so that we would not have execute these commands manually (which we needed to do to use ROS) every time we opened a new terminal.

Section 7 Lessons Learnt – Conclusion

Without a doubt, Alex, the remotely controlled vehicle this semester, posed harder and more challenges than the automated robot last semester and we were able to learn new and very important lessons from this process. One of the most important lessons we learnt in this project was the importance of trial running and 'playing' with our robot in the process of testing it. Towards the end of the project, when all the different system of Alex finally came together, with the ultrasonic sensor and colour sensor all mounted on Alex, the weight distribution of Alex did not come across as a problem to us since it had been alright all along. We did not realize that with all the new hardware added onto Alex, the center of gravity had slowly shifted to the front too much. After everything came together, we tested the system by running it through the maze several times and it continued to perform well. Luckily, when we brought Alex home to fine-tune the movement controls, we decided to test out the system by 'playing' with it and tried make it do bigger movements. It was through this testing, we realized that Alex was front-loaded excessively as Alex will fall to the front after a big movement such as turning large angles. We quickly shifted the battery pack to the back and added a metal weight to Alex. After this change, Alex was able to move smoothly with less jerks and no longer run the risk of falling forward after the hump. These update and improvement to Alex would not have been possible without 'playing' with Alex multiple times to test for bugs, thus from this project we learnt the importance of testing our system multiple times to check for bugs even if it seems to be performing alright.

Another important lesson we learnt was to be resourceful and open to others' advice. While it is true that we must have our own opinions and not be swayed by others, it is also important that we remain open to advice from people with more experience and decide for ourselves if it is useful for us, as these valuable pieces of advice can steer us away from mistakes they have made and allow us to have a smoother journey. For example, by listening to advice from our seniors and the lab technicians, our group paid close attention to ensuring that our Rpi board does not come into contact with metal surfaces which might cause the Rpi to short circuit and spoil. With our cautious attitude, the Rpi board was not damaged, and we were able to run the hector SLAM without worrying about the capacity of Rpi which was a prominent issue groups that switched to 3rd generation Rpi had to worry about. Additionally, we were also able to get help from our friends to 3D print a holder for the colour sensor which help us to attach the colour sensor securely to Alex. Thus, we learnt that while it is important for us to focus on our work, we must also learn from others and be resourceful.

On the other hand, we have also made several significant mistakes in this project. One of the biggest mistakes was our lack of communication as that took up a lot of our time on this time-crunched project. Since there are many incidents of us working on the different part of the same code at the same time, at first when we failed to communicate with each other what we are doing, we ended up overriding each other code and causing the codes to be edited without a copy of the previous version. After we realized the problem and would ensure that nobody is editing before we upload our code, we ran into new problems due to our lack of effective communication. As we chose to focus on completing our own part, we lacked understanding of the algorithm of other parts of the codes done by our groupmate thus if we edit the code without consulting the person who wrote it, we might run into several bugs. In one of the more notable incidents, while we were working on the code, one of us commanded out a crucial line of the code unintentionally and Alex left wheel could no longer reverse. Since we did not communicate effectively, we spent one lab session trying to figure out why Alex was malfunctioning, and we almost changed out the motor thinking that it was a hardware problem. That would have been disastrous as we had to dismantle the entire robot, replace the motor, replace the encoder and reconfigure the encoder when we are only one day away from the actual run. Luckily, we managed to find the problem after we dismantled

Alex, before we had to swap out the left motor. Therefore, given the time we wasted due to lack of communication throughout the project, it is one of the greatest mistakes we made as we should have ensured that everyone understand what is changed or updated to and actively clarify before editing to programme.

Lastly, another biggest mistake we made was our lack of foresight. Generally, we just followed the lesson schedule to complete our components. Due to our lack of foresight, we did not split our work earlier and work on each of the components individually between every lab as it did not occur to us that later it would become a problem. Towards the end of the project, when all components had to come together, we felt very handicapped as only one of us could work on Alex each time to install and configure each component. Thus, we were caught in a situation where time is tight, but we just had to wait while some of us were testing out the colour sensor before some of us could do the LiDar, ultrasonic or movement calibration. All of these resulted in time loss and reduced efficiency for the project. To mitigate this, perhaps we could have delegated work much earlier and start working on each component between lab sessions throughout the semester so that we can reduce the amount of time we spend waiting for each other to complete our parts.

References

1. Zhang, Y., Li, Y., Zhang, H., Wang, Y., Wang, Z., Ye, Y., Yue, Y., Guo, N., Gao, W., Chen, H., & Zhang, S. (2023). Earthshaker: A Mobile Rescue Robot for emergencies and disasters through teleoperation and autonomous navigation. *JUSTC*, *53*(1), 4. https://doi.org/10.52396/justc-2022-0066
2. Liljebäck Pål, Pettersen, K. Y., Stavdahl, Ø., & Gravdahl, J. T. (2013). *Snake robots: Modelling, mechatronics and control*. Springer.

Appendix 5.1 – Bare metal

```
26  void leftForward(int pwm) {
27   TCCR0A=0b10000001; //Prescalar 64
28    OCR0A=pwm;
29  }
30
31  void leftReverse(int pwm) {
32    TCCR0A=0b00100001;
33    OCR0B=pwm;
34  }
35
36  void rightForward(int pwm) {
37    TCCR1A=0b00100001; //clear on match set on down count phase correct pwm
38    OCR1BL=pwm;
39  }
40
41  void rightReverse(int pwm) {
42    TCCR2A=0b10000001;
43    OCR2A=pwm;
44  }
45
46
47  void stop() {
48    TCCR0A=0;
49    TCCR1A=0;
50    TCCR2A=0;
51  }
```

Functions to control wheel movement using Bare metal

```
void sendObjectStatus(){

  int whiteArray[] = {100,70,75};
  int blackArray[] = {350,310,200};

  PORTB &= ~(1<<S2); //set S2 pin as LOW
  PORTB &= ~(1<<S3); //set S3 pin as LOW
  R = pulseIn(OUT, LOW);
  delay(100);
  R = map(R, BR, WR, 0, 255);

  // Read Green freqency
  PORTB |= (1 << S2); //set S2 pin as HIGH
  PORTB |= (1 << S3); //set S3 pin as HIGH
  G = pulseIn(OUT, LOW);  // Reading the outpt Green frequency
  delay(100);
  G = map(G, BG, WG, 0, 255);

  // Setting Blue frequency
  PORTB &= ~(1<<S2); //set S2 pin as LOW
  PORTB |= (1 << S3); //set S3 pin as HIGH
  B = pulseIn(OUT, LOW);  // Reading the output Blue frequency
  delay(100);
  B = map(B, BB, WB, 0, 255);
```

Setting the pins to HIGH/LOW in bare metal

Appendix – 5.2 Hue algorithm

```
int determine_colour(int R, int G, int B){
  double red = (double)R/255;
  double green = (double)G/255;
  double blue = (double)B/255;
  double newArray[3] = {red,green,blue};
  for(long i = 0; i <2;i++){
    long flag = 0;
    for(long j = 0 ; j < 3 - (i + 1); j++){
      if (newArray[j+1] < newArray[j]){
          double temp = newArray[j + 1];
          newArray[j + 1] = newArray[j];
          newArray[j] = temp;
          flag += 1;
     }
      }
    if(flag == 0){
      break;
    }
  }

  double hue = 0;
  if(newArray[2] == red){
    hue = (green - blue)/(newArray[2] - newArray[0]);
  }else if(newArray[2] == green){
    hue = 2.0 + (blue - red)/(newArray[2] - newArray[0]);
  }else{
    hue = 4.0 + (red - green)/(newArray[2] - newArray[0]);
  }
  if(hue < 0){
     hue = hue * 60;
      return hue + 360;
  }
  return hue * 60;
}
```

Appendix 5.3 – Ultrasonic Code

```
int determine_distance(){
  //bare metal version --------
  PORTC &= ~(1 << TRIG);
  delayMicroseconds(2);
  PORTC |= (1 << TRIG); //set trig pin as HIGH
  PORTC &= ~(1 << TRIG); //set trig pin as low
  delayMicroseconds(10);
  //bare metal version --------

  long duration = pulseIn(ECHO, HIGH, TIMEOUT); // microseconds
  float currentCMS = duration * SPEED_OF_SOUND / 2 / 10000;

  if(currentCMS < 0){ //if its negative means that it is too far away from the wall
    return 0;
  }
  return (int)currentCMS;
```

Appendix 5.4 – sendObjectStatus function

```
void sendObjectStatus(){

  int whiteArray[] = {100,70,75};
  int blackArray[] = {350,310,200};

  PORTB &= ~(1<<S2); //set S2 pin as LOW
  PORTB &= ~(1<<S3); //set S3 pin as LOW
  R = pulseIn(OUT, LOW);
  delay(100);
  R = map(R, BR, WR, 0, 255);

  // Read Green freqency
  PORTB |= (1 << S2); //set S2 pin as HIGH
  PORTB |= (1 << S3); //set S3 pin as HIGH
  G = pulseIn(OUT, LOW);  // Reading the outpt Green frequency
  delay(100);
  G = map(G, BG, WG, 0, 255);

  // Setting Blue frequency
  PORTB &= ~(1<<S2); //set S2 pin as LOW
  PORTB |= (1 << S3); //set S3 pin as HIGH
  B = pulseIn(OUT, LOW);  // Reading the output Blue frequency
  delay(100);
  B = map(B, BB, WB, 0, 255);

  int hue = determine_hue(R,G,B);
  dbprintf("hue is %i \n",hue);

  char colour;
  if(hue < 15 || hue <330){
    colour = 'r';
    dbprintf("colour is red \n");
  }else if(hue > 90 && hue < 130){
    colour = 'g';
    dbprintf("colour is green \n");
  }else{
    colour = 'o';
    dbprintf("some other colour");
  }

  int distance = determine_distance();
  if(distance == 0){
    dbprintf("coast clear\n");
  }else{
    dbprintf("object is %i away \n",distance);
  }
```

Appendix 5.5 – TPacket to be sent to the RPi

```
void sendStatus()
{
  // Implement code to send back a packet containing key
  // information like leftTicks, rightTicks, leftRevs, rightRevs
  // forwardDist and reverseDist
  // Use the params array to store this information, and set the
  // packetType and command files accordingly, then use sendResponse
  // to send out the packet. See sendMessage on how to use sendResponse.

  TPacket statusPacket;
  statusPacket.packetType = PACKET_TYPE_RESPONSE;
  statusPacket.command = RESP_STATUS;
  statusPacket.params[0] = leftForwardTicks;
  statusPacket.params[1] = rightForwardTicks;
  statusPacket.params[2] = leftReverseTicks;
  statusPacket.params[3] = rightReverseTicks;
  statusPacket.params[4] = leftForwardTicksTurns;
  statusPacket.params[5] = rightForwardTicksTurns;
  statusPacket.params[6] = leftReverseTicksTurns;
  statusPacket.params[7] = rightReverseTicksTurns;
  statusPacket.params[8] = forwardDist;
  statusPacket.params[9] = reverseDist;
  sendResponse(&statusPacket);
}
```

Appendix 5.6 – Data that is printed out by the RPi

```
void handleStatus(TPacket *packet)
{
    printf("\n ------- ALEX STATUS REPORT ------- \n\n");
    printf("Left Forward Ticks:\t\t%d\n", packet->params[0]);
    printf("Right Forward Ticks:\t\t%d\n", packet->params[1]);
    printf("Left Reverse Ticks:\t\t%d\n", packet->params[2]);
    printf("Right Reverse Ticks:\t\t%d\n", packet->params[3]);
    printf("Left Forward Ticks Turns:\t%d\n", packet->params[4]);
    printf("Right Forward Ticks Turns:\t%d\n", packet->params[5]);
    printf("Left Reverse Ticks Turns:\t%d\n", packet->params[6]);
    printf("Right Reverse Ticks Turns:\t%d\n", packet->params[7]);
    printf("Forward Distance:\t\t%d\n", packet->params[8]);
    printf("Reverse Distance:\t\t%d\n", packet->params[9]);
    printf("\n---------------------------------------\n\n");
}
```

Appendix 6.1 – WASD Commands

```
TPacket commandPacket;

commandPacket.packetType = PACKET_TYPE_COMMAND;

switch(command)
{
    case 'w':
        setWASDParams(&commandPacket,TYPE_DISTANCE,1);
        commandPacket.command = COMMAND_FORWARD;
        sendPacket(&commandPacket);
        break;
    case 'W':
        setWASDParams(&commandPacket,TYPE_DISTANCE,2);
        commandPacket.command = COMMAND_FORWARD;
        sendPacket(&commandPacket);
        break;
    case 'a':
        setWASDParams(&commandPacket,TYPE_ANGLE,1);
        commandPacket.command = COMMAND_TURN_LEFT;
        sendPacket(&commandPacket);
        break;
    case 'A':
        setWASDParams(&commandPacket,TYPE_ANGLE,2);
        commandPacket.command = COMMAND_TURN_LEFT;
        sendPacket(&commandPacket);
        break;
    case 's':
        setWASDParams(&commandPacket,TYPE_DISTANCE,1);
        commandPacket.command = COMMAND_REVERSE;
        sendPacket(&commandPacket);
        break;
    case 'S':
        setWASDParams(&commandPacket,TYPE_DISTANCE,2);
        commandPacket.command = COMMAND_REVERSE;
        sendPacket(&commandPacket);
        break;
    case 'd':
        setWASDParams(&commandPacket,TYPE_ANGLE,1);
        commandPacket.command = COMMAND_TURN_RIGHT;
        sendPacket(&commandPacket);
        break;
    case 'D':
        setWASDParams(&commandPacket,TYPE_ANGLE,2);
        commandPacket.command = COMMAND_TURN_RIGHT;
        sendPacket(&commandPacket);
        break;
```

```
//type set as 0 for distance and type set as 1 for angles
void setWASDParams(TPacket *commandPacket,int type,int multiplier){
    if(type == TYPE_DISTANCE){
        commandPacket->params[0] = (DEFAULT_DISTANCE * multiplier);
    }else{
        commandPacket->params[0] = (DEFAULT_ANGLE * multiplier);
    }
    commandPacket->params[1] = DEFAULT_POWER;
}
```