CG2111A Engineering Principles and Practice

Semester 2 2022/2023

"Alex to the Rescue"

Final Report

Team: B04-6A

Name	Student #	Main Role
Zeng Zheqi	A0258700H	Coding and Software Implementation
Seenivasaragavan Praneet	A0255314L	Circuit Design and Navigation
Xia Cong	A0262355E	Robot Assembly and Motor Calibration
Zhang Wenyue	A0257964L	Sensors and Extra Functions (Colour Detection and Ultrasonic Sensor)

Section 1: Introduction

The main objective of this project is to enable our robot, named Alex, to navigate a maze with an unknown starting position, while utilizing a laser scanner called RPLIDAR to scan its surrounding environment and generate a 2D image of the overall layout of the maze. This will allow the robot to locate and rescue survivors in a disaster site that may be difficult for humans to enter. Additionally, we must prevent Alex from colliding with obstacles and walls throughout the entire navigation process. The maze where Alex is located is not a flat space. Alex needs sufficient power to be able to pass through unknown bumps located inside the maze. This is because the disaster site is never a smooth surface, Alex needs to be able to pass through places that are extremely uneven and covered with rubbles.

Furthermore, while navigating through the maze, it is important for Alex to differentiate between obstacles that are classified as "dummies" or "victims waiting to be rescued" and mark the position of victims on the 2D map of the maze. Obstacles colored in red and green are classified as victims, while all other obstacles are classified as dummies. To ensure that Alex is operating most effectively, it must accurately mark all of the victims in the maze while continuing to map and navigate the maze and avoid collisions as fast as possible within 8 minutes. This makes sure that the rescue team can perform rescue as fast as possible at the disaster site. Guided by SLAM mapping, our group will control Alex remotely throughout its navigation course by setting up a Virtual Network Computing (VNC) server on Alex and a VNC client on a laptop and transmitting commands to Alex via Secure Shell (SSH) protocol.

Section 2: Review of State of the Art

2.1 The SmokeBot Project

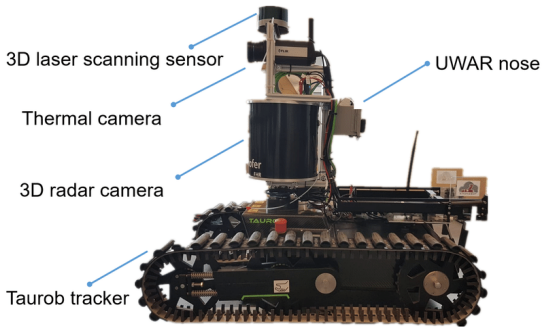

Figure 1: SmokeBot Project

The SmokeBot project has developed a robot system that can operate in low-visibility conditions, such as smoke, dust or fog, to support search and rescue missions in disaster sites. The system integrates hardware and software components, including gas sensors, a 3D radar camera and a stereo thermal camera, which work together to provide the robot with cognitive abilities to assess hazards, map airflow and gas distribution, and localize itself in unknown environments. The

system also includes algorithms that enable the robot to relate a pre-existing map of the environment to its own perception [1].

Strengths

(1) Thermal image: able to detect high temperature areas in a fire with low-visibility
(2) Continuous tracks: able to climb stairs in buildings hence possesses high mobility
(3) Map matching: able to incorporate hand-drawn pictures with its scanned results to improve mapping accuracy

Weaknesses

(1) In real-world emergency response scenarios, where air-flow, temperature and gas distribution could undergo a drastic change, the gas sensor may not be able to produce desirable results.
(2) The proposed gas discrimination module may not be able to reliably separate gasses when they are too close together, and the separation limits depend on various factors.
(3) Unable to work under high temperature, hence cannot enter the central area of fire [1].

2.2 The Spot® robot

Figure 2: The Spot® robot

The Spot® robot can handle up to 14 kg payloads and has a 3D vision system with SLAM and obstacle avoidance. The robot can be controlled remotely and also perform autonomous tasks through a web-based software, Scout. It has ingress protection IP54 and can operate in temperatures ranging from -20°C to 45°C [2].

Strengths

(1) It can walk omnidirectionally, climb and descend stairs.
(2) It is able to operate under wet conditions.
(3) It can mount on different payloads to perform different tasks when required [3].

Weaknesses

(1) It is unable to operate under extreme temperatures beyond -20°C or 45°C.
(2) It is not suitable to work in potentially explosive environments due to the use of lithium-ion batteries [4].

Section 3: System Architecture

3.1 Diagram of the components of ALEX

Alex consists of two sources of power, an Arduino Uno, a Raspberry Pi 4 (RPi), an RPLIDAR and various sensors. The figure below depicts how these various components are connected to one another:

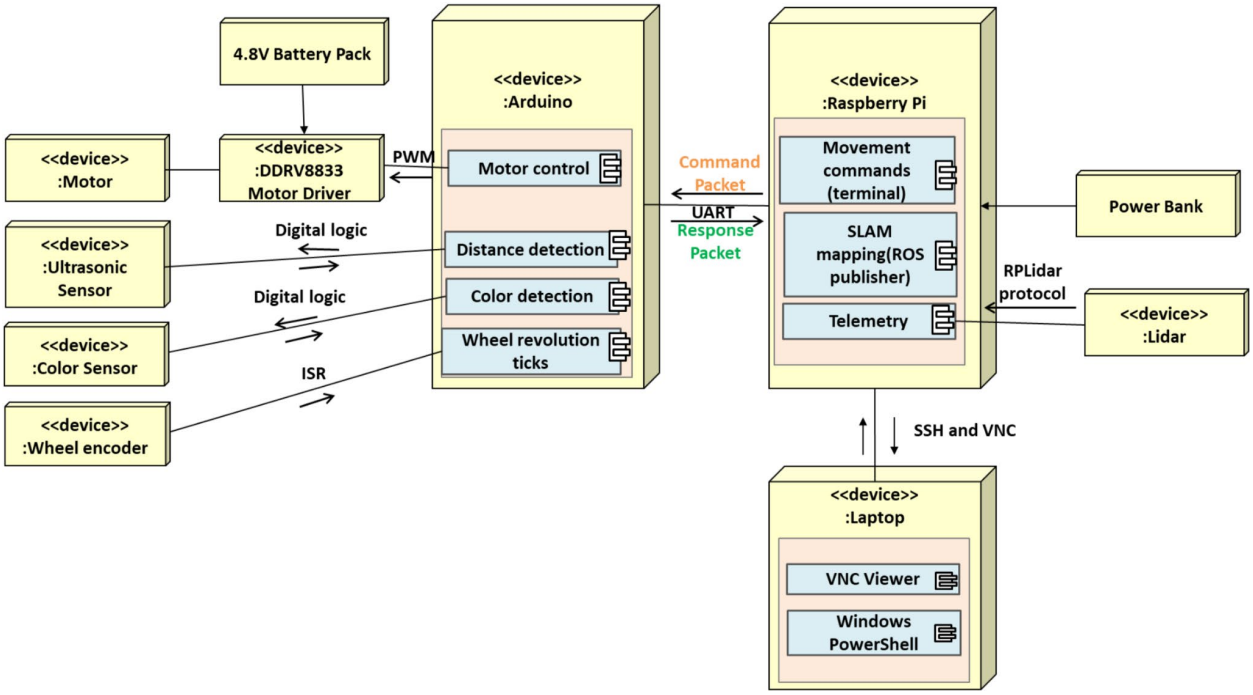

Figure 3: Overview of Alex's system architecture

3.2 Arduino Uno

The Alex incorporates two different microprocessors, the Arduino and the RPi. These microcontrollers are used in controlling the portions of the electronic system.The Arduino Uno acts as a bridge between the RPi and the hardware components of Alex (e.g motor, ultrasonic sensor, color sensor). It is connected to the hardware devices via jumper wires. It constantly receives the data detected by the sensors and the wheel encoders located on the motor of the ALEX. It connects with the RPi via a USB cable, so it is also powered by RPi. The movement commands and data which is sent from the laptop through the RPi enables the Arduino Uno to send a command to the motor driver. Ultimately, these commands determine the direction of movement of the ALEX.

3.3 Raspberry Pi 4 (RPi)

The RPi is considered the main "functioning brain" of this entire system. It is connected to the Arduino Uno and the RPLIDAR through USB cables. The laptop is connected to the RPi through

VNC(Virtual Network Computing), whereby the RPi can be accessed through the laptop remotely via mobile hotspot. The operator can send predefined commands to retrieve readings from various sensors connected to the Arduino Uno and control the motors of Alex. A RPlidar node is created in the ROS on RPi to publish Lidar readings and relevant nodes are used to get the RPLIDAR and the Hector SLAM algorithm to work on the ROS.

3.4 RPLIDAR

The RPLIDAR is a hardware device which is used to map its surroundings through the LIDAR (Light Detection and Ranging) method. This particular component is a 360-degree laser range scanner and makes use of the laser triangulation ranging principle and high-speed vision acquisition to scan the surroundings [5]. LIDAR is the method of finding out an object's surroundings by targeting the nearby surfaces with a laser and measuring the time taken for the reflected light to be received [6]. This device is connected to the RPi through the USB cable and is in turn powered by the RPi. The RPLIDAR sends over the data to the RPi and the data is processed by running Hector Simultaneous Localization and Mapping (SLAM) to map the surroundings. The data obtained would be plotted onto a map eventually to gain a better visualization of the mapped environment.

3.5 Colour Sensor

An TCS3200 color sensor is connected to the front of Alex to determine the colour of the obstacle in front of it when the distance between Alex and that obstacle is small. The LEDs will shine white light onto the obstacle and record the intensity of the light reflected. The photodiodes will convert the light signals received to different RGB values via the RGB filters inside the sensor. The RGB values will be sent to Arduino and the color of the obstacle will be determined based on the composition of the RGB values received by Arduino.

3.6 Laptop

The laptop acts as the primary remote platform to control Alex. It serves as a tool for communication with the RPi through a wireless connection. Commands for movements (turning left or right. moving forward and backwards) and sensors are being sent to Alex from the laptop remotely.

3.7 Power

The Alex is powered by two 5V power supplies which are the power bank and a battery pack containing 4 batteries. The power bank serves as a power supply for the RPi, which in turn is utilized to power the Arduino Uno and the LIDAR. The battery pack supplies power to the motors through the motor driver. The Arduino Uno powers the rest of the hardware components.

Section 4: Hardware Design

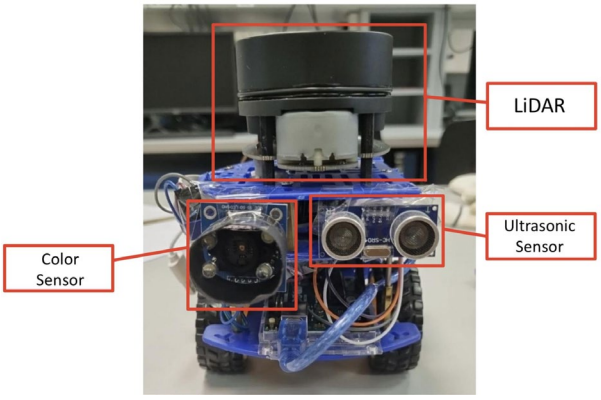

Figure 4: Front View of Alex

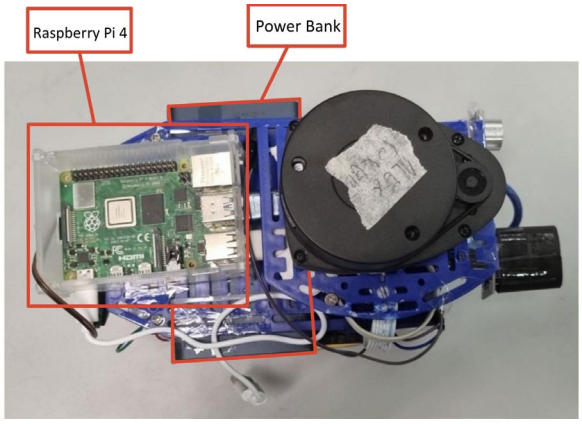

Figure 5: Top View of Alex

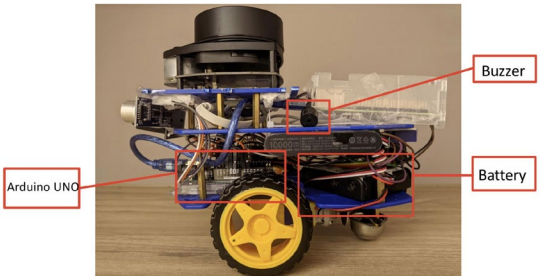

Figure 6: Side View of Alex

4.1 Layout

Alex consists of two and a half layers, with each component strategically placed to maximize functionality and minimize interference. The LIDAR sensor is situated on the topmost layer to ensure that it has an unobstructed view of the surroundings and is not blocked by other sensors like the ultrasonic sensor.

The ultrasonic sensor is located at the front of the second layer to provide an unimpeded detection of distances, and the RPi is also situated on the second layer for ease of access and frequent monitoring. The power bank is attached beneath the second layer at the back to balance the center of gravity as the LiDAR is situated in front. The bottom layer contains the battery pack and Arduino, while the motors and wheel encoders are beneath the bottom layer.

For connections of components, the wheel encoder, motors, buzzer, ultrasonic sensor, color sensor, and RPi are connected to the Arduino with the use of wires, USB cables, and breadboards. LIDAR is connected to the RPi. The RPi is powered by the power bank and the motors are powered by the battery pack.

We have made holes in the plastic layers for wires to go through. Moreover, as the wires are long, they are messy and easily tangled. Thus, we coil the wires around the long screws that are used to support the second layer, minimizing clutter and enhancing the neatness of the wiring system.

4.2 RPi Protection

As RPi is a critical component of Alex, it is imperative to take measures to ensure its protection. To prevent damage from physical impact or mishandling, we have included a sturdy protective case around the RPi. This safeguards the RPi against any potential breakage or crash.

4.3 Color Detection Improvement

To enhance the accuracy and reliability of color detection results, it is important to address the impact of ambient light. To mitigate this effect, a shielding made of black paper is introduced to surround the color sensor. This shielding effectively eliminates extraneous light sources, allowing only reflected light from the object to be detected.

4.4 Ultrasonic Sensor

During the movement of Alex, an ultrasonic sensor is utilized to obtain distance data from the front of the device to any obstacles such as walls or colored objects when needed. This is crucial in preventing Alex from hitting the obstacles. By utilizing the ultrasonic sensor to determine the optimal distance to objects when detecting colors, the accuracy of the detection process is significantly improved. Furthermore, the ultrasonic sensor is also employed while parking Alex to ensure that the device is not positioned too close to the front wall.

4.5 Buzzer

As an additional feature, we have incorporated a buzzer into the system to provide audio cues for color detection. When a red or orange color is detected, the buzzer will emit a single buzz to alert the user. Subsequently, if another object detected has a larger RGB sum than the previous object, it is detected to be red, the buzzer will emit three consecutive buzzes. Similarly, if a green color is detected, the buzzer will emit two consecutive buzzes. This added functionality greatly enhances our ability to identify colors accurately, providing an additional sensory feedback mechanism for improved color detection.

Section 5: Firmware Design

Arduino Uno is responsible for sending hardware information to RPi and receiving command packets from RPi via UART. It is also responsible for controlling various hardware components implemented in Alex such as motors, ultrasonic sensor, color sensor, and buzzer.

5.1 Communication protocol

Reading packets

A common packet structure is being defined between Arduino and RPi to ensure that the data transmitted and received are interpreted the same way, as shown below.

```
typedef struct {  // This packet has 1 + 1 + 2 + 32 + 16 * 4 = 100 bytes
  char packetType; // 1 byte
  char command;    // 1 byte
  char dummy[2];   // Padding to make up 4 bytes
  char data[MAX_STR_LEN]; // String data
  uint32_t params[16];
} TPacket;
```

The first two variables are followed by 2-byte padding because RPi interprets 4 bytes of data at a time. The last variable is an array of sixteen 4-byte parameters that store various data collected.

Interrupts will be triggered when the process of receiving the packet is complete or when the data register is empty. The receiver and transmitter will be enabled.

Function readPacket() is called continuously inside the main loop where it calls readSerial() and readSerial() function (Appendix) will read from a circular buffer and return whether the packet is complete or incomplete. If the packet is incomplete, function sendBadPacket() will be called where an error packet will be sent back to RPi indicating that the packet sent is a bad packet inside the response byte. The complete packet will be deserialized and checked for the magic number and checksum. If the checksum calculated by Arduino does not match the checksum inside the packet or the magic number is unmatching, an error packet will be sent as a response to RPi. If no errors occur, the complete packet will then be passed to handlePacket().

Sending packets from Arduino to RPi

Error packets will be serialized and sent from Arduino to RPi if there is an error detected as mentioned above. Response packets indicating the status of Alex where data detected by sensors are contained inside the parameters of the packet are being sent back to RPi when function sendStatus() or sendColor() are being called. Packets indicating OK when hello packets are received or commands have been confirmed will also be sent as mentioned above. Message packets will be sent when some string needs to be displayed on the screen to indicate status. All of these packets will be sent by calling sendresponse() function where all of the data inside each packet will be serialized and sent to RPi via the UART channel.

5.2 High-Level Algorithm on Arduino Uno

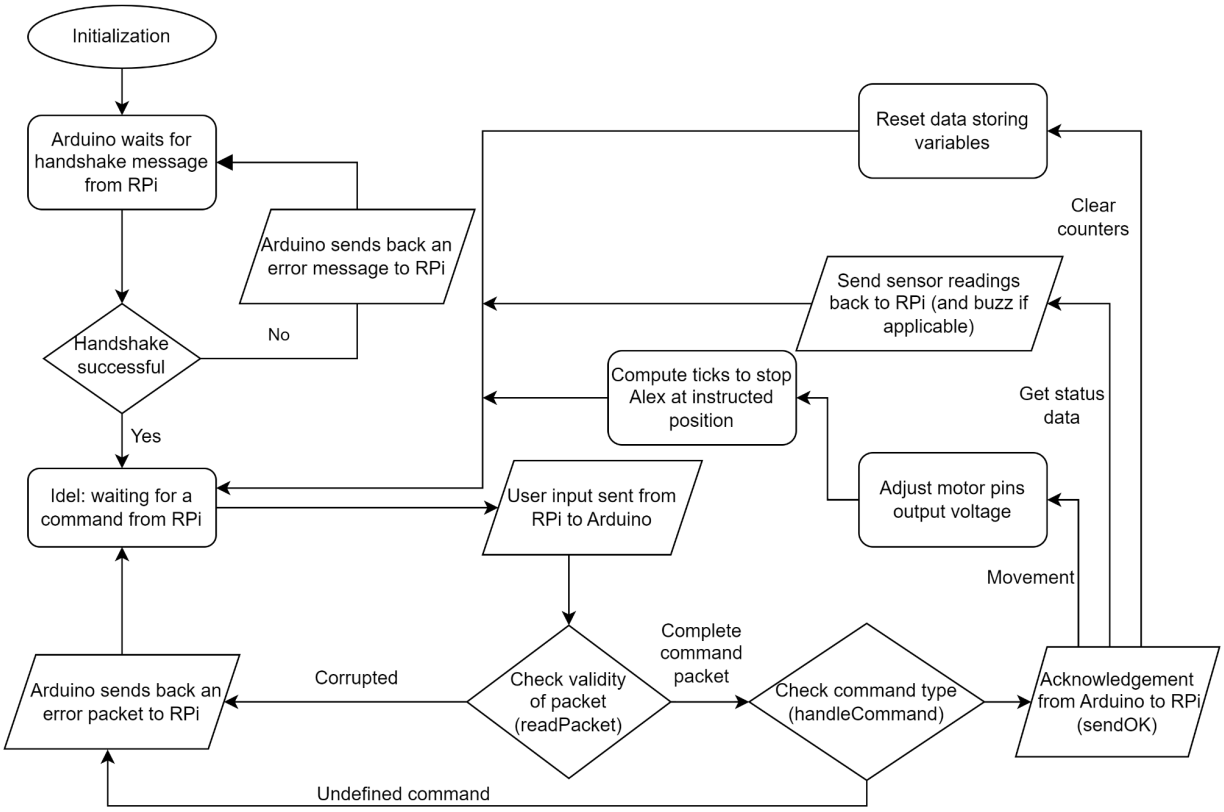

Figure 7: High-Level Algorithm on Arduino Uno

Executing different types of commands:
- Movement command
An 8-bit value converted from the "speed" parameter is directly assigned to the OCR0A, OCR0B, OCR2A, and OCR1BL registers using bare-metal code. (Appendix)

- Request status(and color) command:
The Arduino assigns odometry data, color sensor readings, ultrasonic sensor readings, and the color detected to an array of 16 params in a TPacket(refer to 5.1), and sends this status packet to RPi using sendResponse(). In addition, we have integrated the buzzer mechanism in the same function so that it buzzes at different times according to the color code.

- Clear command:
Arduino will set variables containing odometry data and sensor readings to 0.

Section 6: Software Design

6.1 Teleoperation

Alex is teleoperated by a program called "Alex-pi" running in the terminal of RPi, which is remotely accessed by a laptop via SSH. RPi and Arduino are connected via UART. In the program, RPi first tries to connect to Arduino by startSerial(). If successful, it will send a helloPacket to

Arduino as a handshake message. A receiver thread will then be created on RPi. This separate thread handles data received from Arduino in parallel with the main program so that it will not interfere with commands sent to Arduino and is more efficient.

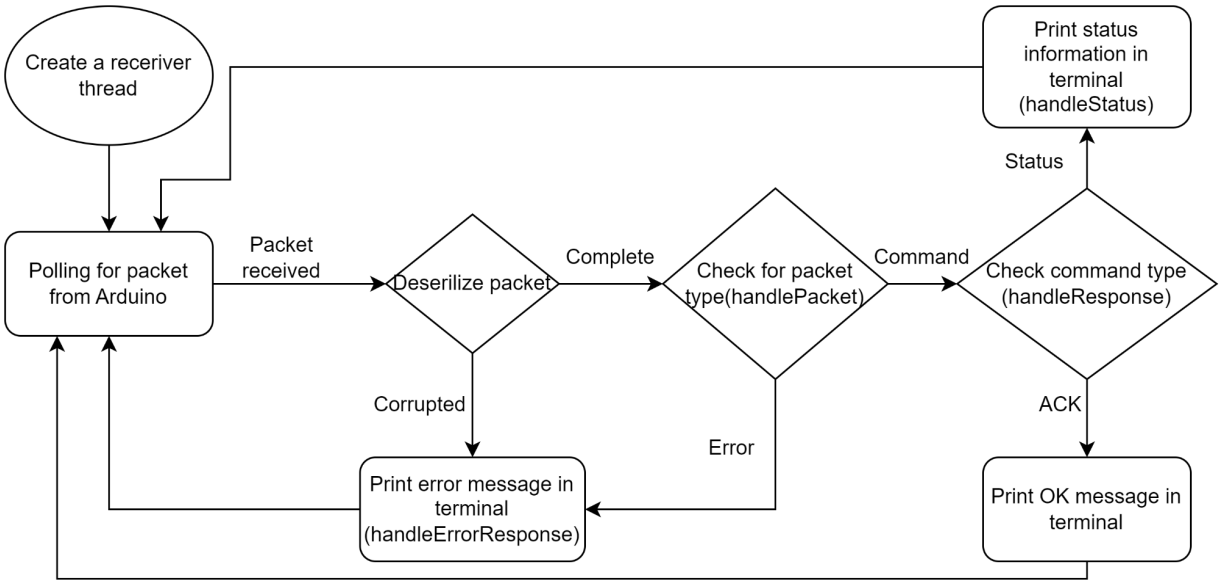

Figure 8: Receiver Thread (for RX)

The main program then enters a loop to poll for input characters from the terminal in parallel. Valid commands (i.e. in our predefined format such as ‘g’ to detect color and ‘d’ to turn right) will be sent to Arduino, otherwise ignored. The UART connection will be closed if the loop is terminated by an exitflag.

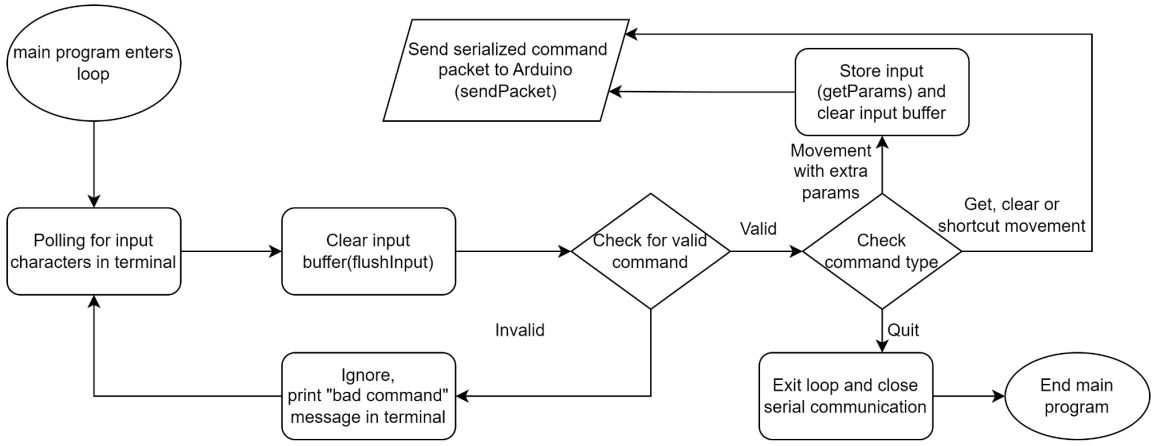

Figure 9: Main Thread (for TX)

6.2 Movement

Alex’s movement commands are modified(Appendix) to improve efficiency and user experience. For example, moving forward is now controlled by ‘w’(5 cm) or ‘W’(20 cm), with distance and speed integrated into one key. A new and simplified function “sendColor” to get the distance in front and perform our color detection function is created, controlled by ‘g’ or ‘G’, leaving the old “sendStatus()” function mainly for debugging purposes. To further enhance the user experience, the command keys can be mapped to a game-controller-style interface specially created for Alex’s

control via “Monect PC Remote” software on a smartphone (with receiver software installed on the laptop), using a wifi or Bluetooth connection.

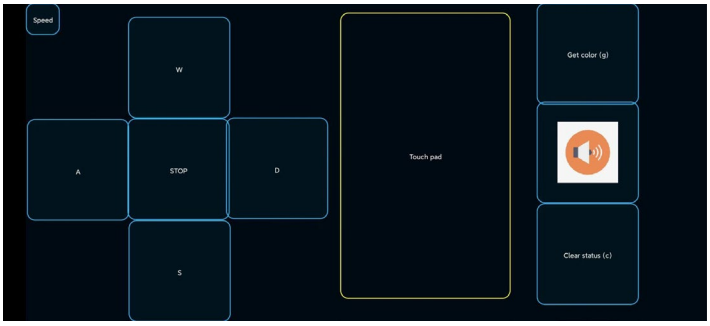

Figure 10: Remote Control Panel

6.3 SLAM mapping

An RPLidar node is first set up to publish the readings from the Lidar on a topic “/scan”. A Hector SLAM node is set up next. It subscribes to “/scan” to perform the Hector SLAM algorithm using the data and publishes the mapping information on another topic “/tf”. In addition, the Hector Slam algorithm also determines the odometry of the Lidar based on its initial position and publishes to a topic “/slam_out_pose”. Finally, an RViz node is set up and subscribes to the “/tf” topic to visualize the mapping information. The map can also show Alex’s odometry by enabling slam_out_pose and configuring its dimension shown by an arrow. We have also configured the size of the arrow to precisely match Alex’s real physical dimension through calibration.

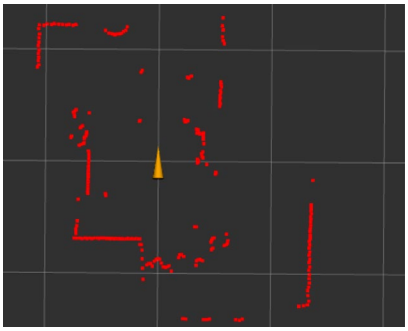

Figure 11: Map Produced by RViz

6.4 Color detection

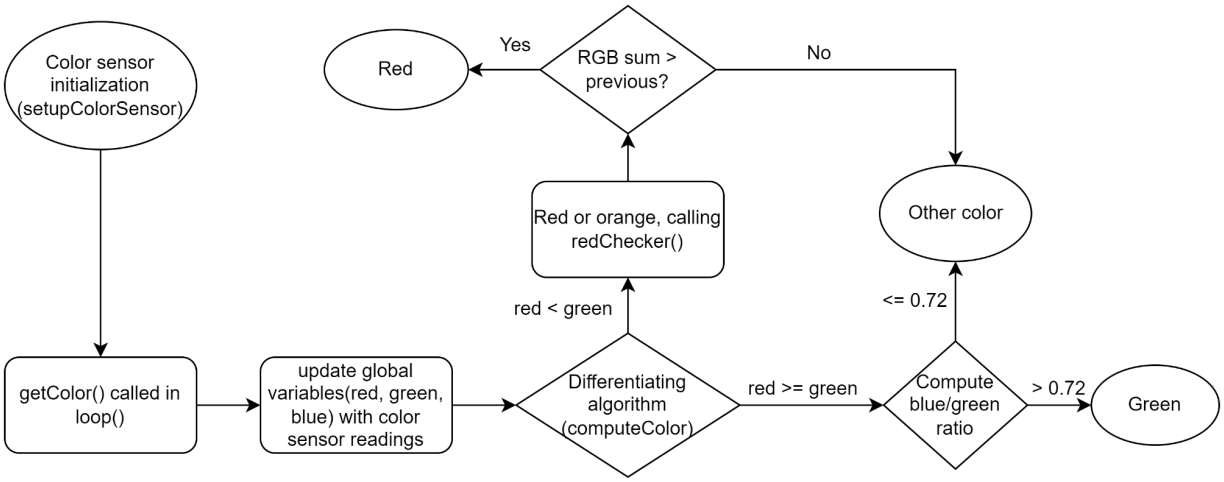

Figure 12: Color Differentiating Algorithm

There are three comparing algorithms for color detection:
i) Check if the color is a "warm" or "cold" color by comparing the "r" value with "g" value, by the setting of our color sensor(Appendix), a lower "red" value indicates more intense red in the color composition.
ii) If the color is "cold", simply take the "b/g" ratio to compare with a threshold value of 0.72. It is experimentally obtained that green objects have a "b/g" ratio > 0.72.
iii) If the color is "warm", the sum of "RGB" values will be stored in an array(Appendix) and will be compared against the next "suspected red" object's RGB sum. A red object will have a larger RGB sum than an orange object, as determined experimentally. Hence, this will confirm if the first warm-colored object encountered is red or not, as there is only one red object in the maze.

Section 7: Lessons Learnt - Conclusion

In conclusion, constructing this Alex and programming it in a way to follow the desired instructions, taught us that designing an operating robot from the start is definitely not an easy task. Throughout this journey, we faced a lot of obstacles and there were major takeaways from this project, which would be focused on in this section.

7.1 Two Mistakes we made

One of the major mistakes we made was that we were not consistent with the tasks assigned to us during the studio sessions and this made us fall behind a lot when it came to the construction of the Alex. We had to spend a lot of time outside lab hours, just to catch up on the implementations that we were supposed to do during the lab. In addition, we required additional time to troubleshoot to make sure that all basic movement issues of Alex were solved.

Another mistake is that we modified the code and changed the positions of components after calibration, which invalidated the previous parameters obtained. For example, the color sensor was initially calibrated without a black shield and with 20% frequency scaling. However, after some trials, we found that the readings are sometimes inconsistent, which could be a problem caused by

ambient light. To tackle this, we added a shield around the color sensor and this changed the intensity of ambient light, therefore we had to re-calibrate the color ranges.

7.2 Lessons Learnt

One of the most important lessons we learnt was that whenever we write our code, we should document it properly in a consistent manner. As we started writing the code for our Alex, the number of functions and the complexity of our source code multiplied constantly. We also referred to the various lines of code in the studio sessions and implemented them in our source code. As we progressively carried on with the project, the fact that there was incomplete documentation made us confused about what we wrote previously and it took up a lot of time for us to debug errors. Having proper documentation enables us to cross-check our code at any instance and makes it easier to find out any errors and also makes sure that it can be marked by our professors easily as they can better understand it.

Another lesson learnt was that we should plan beforehand, how to place the components in a hardware system so that they can be properly integrated during the movement phase. In the case of our Alex, we started adding various components to the main body without proper planning. The wirings were done in a chaotic manner and we did not think of how they would be eventually obstructed by the random positioning of the hardware components on the Alex. After the complete construction of the system, we noticed that the wires were tangled and an uneven weight distribution was present throughout the Alex. This hindered the general movement of the Alex and if we had to make any changes to our layout, various components had to be taken out each time, causing a lot of inconvenience. If we had properly planned the layout at the initial stages, the Alex would have been designed in an effective manner so that any hardware issues could have been solved without any layout hindrances.

References

[1] Achim J. Lilienthal et al. (February 2019) Towards Gas Discrimination and Mapping in Emergency Response Scenarios Using a Mobile Robot with an Electronic Nose. Retrieved on March 27, 2023, from https://www.mdpi.com/1424-8220/19/3/685?type=check_update&version=1

[2] *Spot - ROBOTS: Your Guide to the World of Robotics*. (n.d.). Spot - ROBOTS: Your Guide to the World of Robotics. Retrieved on March 28, 2023, from https://robots.ieee.org/robots/spotmini/

[3] *Spot® - The Agile Mobile Robot*. Boston Dynamics. (n.d.). Retrieved on March 28, 2023, from https://www.bostondynamics.com/products/spot

[4] Boston Dynamics Support Center. (n.d.). Retrieved on March 29, 2023, from https://support.bostondynamics.com/s/article/Introduction-to-the-Spot-battery-and-charger

[5] RPLIDAR A1-SLAMTEC. Retrieved on March 29, 2023, from https://www.slamtec.com/en/Lidar/A1

[6] Jie Shan and Charles K. Toth (2018). *Topographic Laser Ranging and Scanning: Principles and Processing* (2nd ed.). CRC Press.

Appendix

Section 5.1

```
// Reads in data from the serial port and
```

```
// deserializes it.Returns deserialized

// data in "packet".

TResult readPacket(TPacket *packet) {

  char buffer[PACKET_SIZE];

  int len;



  len = readSerial(buffer);



  if (len == 0)

    return PACKET_INCOMPLETE;

  else

    return deserialize(buffer, len, packet);

}



// Read the serial port. Returns the read character in

// ch if available. Also returns TRUE if ch is valid.

int readSerial(char *buffer) {

  int count = 0;

  TBufferResult result;

  while (result == BUFFER_OK) {

    result = readBuffer(&_recvBuffer, (unsigned char *)&buffer[count]);



    if (result == BUFFER_OK) {

      count++;

    }
```

```
  }

  return count;

}
```

Section 5.2

```
// Set up Alex's motors.

void setupMotors() {

  /* Our motor set up is:

   *    A1IN - Pin 5, PD5, OC0B

   *    A2IN - Pin 6, PD6, OC0A

   *    B1IN - Pin 10, PB2, OC1B

   *    B2In - Pin 11, PB3, OC2A

   */

  DDRD = 0b01100000;  // Pin 5, 6

  DDRB = 0b00001100;  // Pin 10, 11



  TCNT0 = 0;

  TCCR0A = 0b10100001;  // Set OC0A and OC0B mode: clear on compare

  TIMSK0 |= 0b110;      // Enable OCIEA and OCIEB



  TCNT1L = 0;

  TCCR1A = 0b00100001;  // Set OC1B mode: clear on compare

  TIMSK1 |= 0b100;      // Enable OCIEB
```

```
  TCNT2 = 0;

  TCCR2A = 0b10000001;  // Set OC2A mode: clear on compare

  TIMSK2 |= 0b010;      // Enable OCIEA



  OCR0A = 0;

  OCR0B = 0;

  OCR1BL = 0;

  OCR1BH = 0;

  OCR2A = 0;

}

// Bare-metal version of Arduino library's analogWrite()

void setAnalog(int pin, int val) {

  switch (pin) {

    case LF:

      OCR0A = val;

      break;



    case LR:

      OCR0B = val;

      break;



    case RF:

      OCR2A = val;

      break;
```

```
    case RR:
      OCR1BL = val;
      break;
  }
}
```

Section 6.2

```
void sendCommand(char command)
{
    TPacket commandPacket;

    commandPacket.packetType = PACKET_TYPE_COMMAND;

    switch(command)
    {
        // *WASD configuration*
        // lower case: 5cm forward/backward or 15° turn left/right
        // upper case: 20cm forward/backward or 90° turn left/right
        case 's':
            commandPacket.params[0] = 7;
            commandPacket.params[1] = 97;
            commandPacket.command = COMMAND_FORWARD;
            sendPacket(&commandPacket);
            break;
```

```
        case 'S':
            commandPacket.params[0] = 20;
            commandPacket.params[1] = 100;
            commandPacket.command = COMMAND_FORWARD;
            sendPacket(&commandPacket);
            break;

        case 'w':
            commandPacket.params[0] = 7;
            commandPacket.params[1] = 97;
            commandPacket.command = COMMAND_REVERSE;
            sendPacket(&commandPacket);
            break;

        case 'W':
            commandPacket.params[0] = 25;
            commandPacket.params[1] = 100;
            commandPacket.command = COMMAND_REVERSE;
            sendPacket(&commandPacket);
            break;

        case 'a':
            commandPacket.params[0] = 7;
            commandPacket.params[1] = 100;
            commandPacket.command = COMMAND_TURN_LEFT;
```

```
            sendPacket(&commandPacket);
            break;


        case 'A':
            commandPacket.params[0] = 90;
            commandPacket.params[1] = 100;
            commandPacket.command = COMMAND_TURN_LEFT;
            sendPacket(&commandPacket);
            break;


        case 'd':
            commandPacket.params[0] = 15;
            commandPacket.params[1] = 100;
            commandPacket.command = COMMAND_TURN_RIGHT;
            sendPacket(&commandPacket);
            break;


        case 'D':
            commandPacket.params[0] = 90;
            commandPacket.params[1] = 100;
            commandPacket.command = COMMAND_TURN_RIGHT;
            sendPacket(&commandPacket);
            break;


        case 'f':
```

```
        case 'F':
            getParams(&commandPacket);
            commandPacket.command = COMMAND_FORWARD;
            sendPacket(&commandPacket);
            break;

        case 'b':
        case 'B':
            getParams(&commandPacket);
            commandPacket.command = COMMAND_REVERSE;
            sendPacket(&commandPacket);
            break;

        case 'l':
        case 'L':
            getParams(&commandPacket);
            commandPacket.command = COMMAND_TURN_LEFT;
            sendPacket(&commandPacket);
            break;

        case 'r':
        case 'R':
            getParams(&commandPacket);
            commandPacket.command = COMMAND_TURN_RIGHT;
            sendPacket(&commandPacket);
```

```
            break;


        case 'z':
        case 'Z':
        case '\n': // double tap "Enter" for emergency stop
            commandPacket.command = COMMAND_STOP;
            sendPacket(&commandPacket);
            break;


        case 'c':
        case 'C':
            commandPacket.command = COMMAND_CLEAR_STATS;
            commandPacket.params[0] = 0;
            sendPacket(&commandPacket);
            break;


        case 'h':
        case 'H':
            commandPacket.command = COMMAND_GET_STATS;
            sendPacket(&commandPacket);
            break;


        case 'g':
        case 'G': // get both color and ultrasounic readings,
                  // also trggers buzzer
```

```
            commandPacket.command = COMMAND_GET_COLOR;

            sendPacket(&commandPacket);

            break;


        case 'q':

        case 'Q':

            exitFlag=1;

            break;


        default:

            printf("Bad command\n");


    }

}
```

Section 6.4

```
// On a scale of 0-500, lower value means higher color composition

void getColor() {

  setDigital(S2, LOW);

  setDigital(S3, LOW);

  red = pulseIn(COLOR_OUT, LOW);

  _delay_ms(20);

  setDigital(S3, HIGH);

  blue = pulseIn(COLOR_OUT, LOW);

  _delay_ms(20);
```

```
  setDigital(S2, HIGH);

  green = pulseIn(COLOR_OUT, LOW);

  _delay_ms(20);

}

// Futher check for red or orange

TColor redChecker(int RGB_sum) {

  if (colorCount > 10) {  // prevent buffer overflow

    colorCount = 0;

  }



  rgb_arr[colorCount] = RGB_sum;

  if (colorCount > 0 && rgb_arr[colorCount] > rgb_arr[colorCount - 1])
{  // red > orange

    colorCount++;

    return RED;

  }

  colorCount++;

  return OTHER;

}
```