CG2111A Engineering Principle and Practice

Semester 2 2023/2024

"Alex to the Rescue"

Final Report

Team: B02-05-A

Name	Student #	Main Role
Chittazhi Nivedit Nandakumar		Motor Control and Display
Dwivedi Shyaamal		Circuitry and Organisation
Glenn Chew		ROS and LIDAR
Goh Yik Bing		Colour Sensor and Display
Govindaraj Roshni Daksha		Code Implementation

Acknowledgement

We express our sincere thanks to the CG2111A teaching team - would like to thank the professors of CG2111A, especially Dr Boyd Anderson, TAs, and the lab assistant for helping us throughout the module. This project could not have been done without their presence.

Table of Contents

Section 1 Introduction

The main objective that Alex is supposed to achieve is to be remotely controlled to move through and map out a room filled with various obstacles and correctly identify victims who are either healthy(green) or injured(red) within a 6 minute window. The map will be sent back to the operator in real time, allowing them to decide the most efficient path for Alex to identify victims and map out the area. This is to simulate disaster situations where rescuers have to race against time through various obstacles such as debris or narrow areas to save people. To achieve this, Alex needs the following functionalities,

1. Able to be tele-operated
2. Accurately map out its surroundings back to the operator
3. Correctly identify whether the target is injured, healthy and relay the information back to the operator
4. Able to move forwards, backwards at the operator's desired speed and distance
5. Able to turn left and right at the desired speed and angle
6. Able to show Alex's location on the map sent back to the operator accurately

Section 2 Review of State of the Art

2.1 Clearpath Robotics' Husky UGV

The Husky, developed by Clearpath Robotics, is a rugged, all-terrain UGV (Unmanned Ground Vehicle) which is capable of traversing various terrains, including rough, uneven, or hazardous environments.

Husky can be integrated with existing and new technologies as it uses an open source serial protocol and its ROS is API supported. Its general hardware components include large lug-tread tires for all-terrain mobility, high resolution encoders for improved state estimation and a finely tuned and user adjustable controller for smooth movement even at low speeds. Its overall structure is elegantly built using durable materials with few moving parts. It also contains a LiDAR and Duro GPS. Clearpath Robotics also provides software development kits, libraries and ROS to users so that they can develop custom applications and algorithms using the tools (*Husky UGV - Outdoor Field Research Robot by Clearpath*, n.d.).

The advantages of this robot include versatility and adaptability to a wide range of applications, maneuverability through all kinds of terrains and the ability to integrate different sensors and cameras. However, this increases its complexity and the cost of maintenance. Its size is also comparatively large due to its robustness.

2.2 ANYbotics' ANYmal

ANYmal is a quadrupedal and autonomous robot that is tele-operated for various tasks such as inspection, mapping, surveillance and search and rescue operations. Its special mechanical design enables it to traverse any terrain including stairs and obstacles with ease.

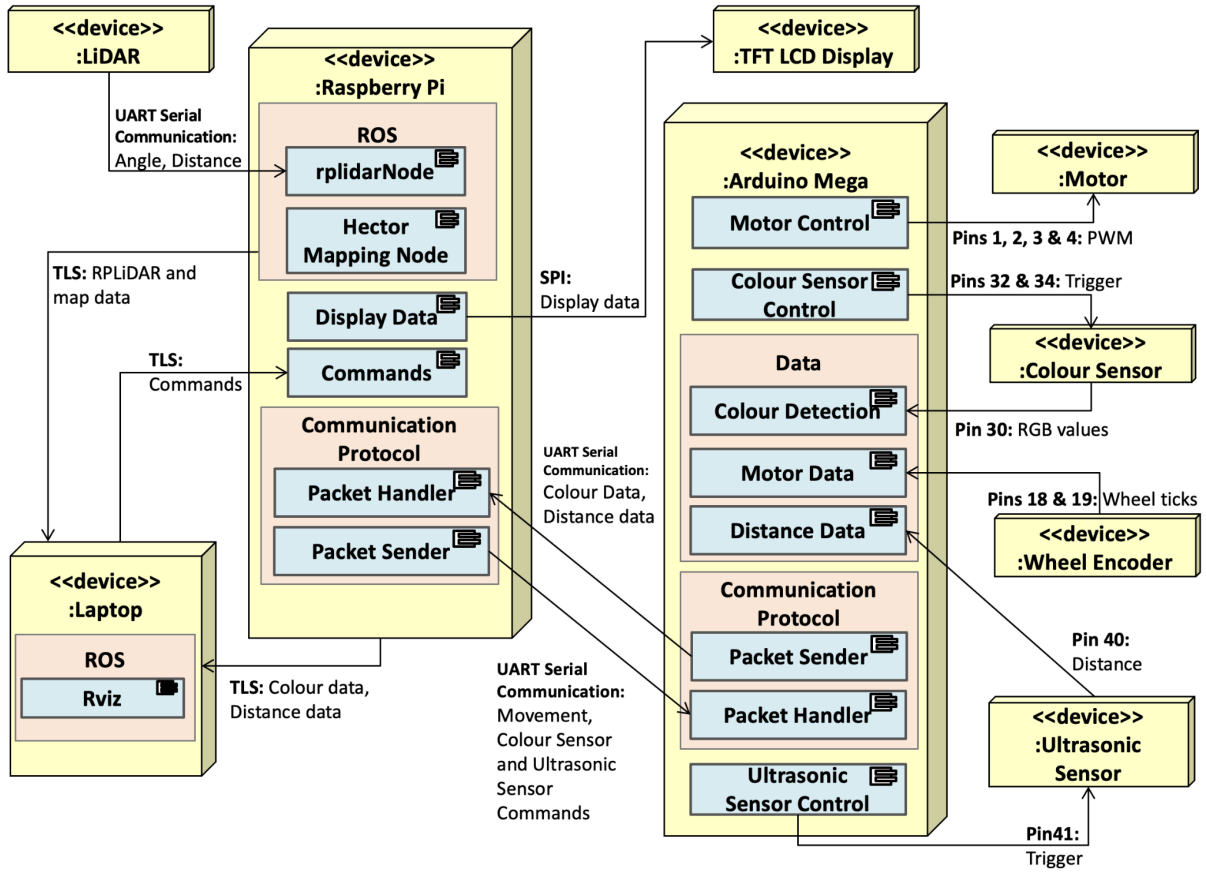
This robot is equipped with 2 cameras: one visual camera with 20x optical zoom for clear long distance images and one thermal camera for temperature reading (-40 to 550 degree celsius) without physical interaction. Other hardware features include an ultrasonic microphone to record acoustic measurements and a LiDAR scanner .

The software aspects have AI incorporated along with a customisable ROS. It also has inbuilt WiFi and 2 × Intel i7 Core computers. AI-based mobility enables it to navigate complex multi-floor areas too. It can perform its defined inspection missions without human intervention or an internet connection as it continuously stores data it receives. It can make smart decisions at low battery status and automatically connects to a docking station to recharge. Its balance between automation and tele-operation is its best advantage (*ANYmal - Autonomous Robotic Inspection Solution - ANYbotic*, n.d.).

Due to the complexity and presence of 2 Intel i7 computers and other sensors, its battery run time is short (about 90 minutes). Initial investment and cost for maintenance is likely to be high for the same reason.

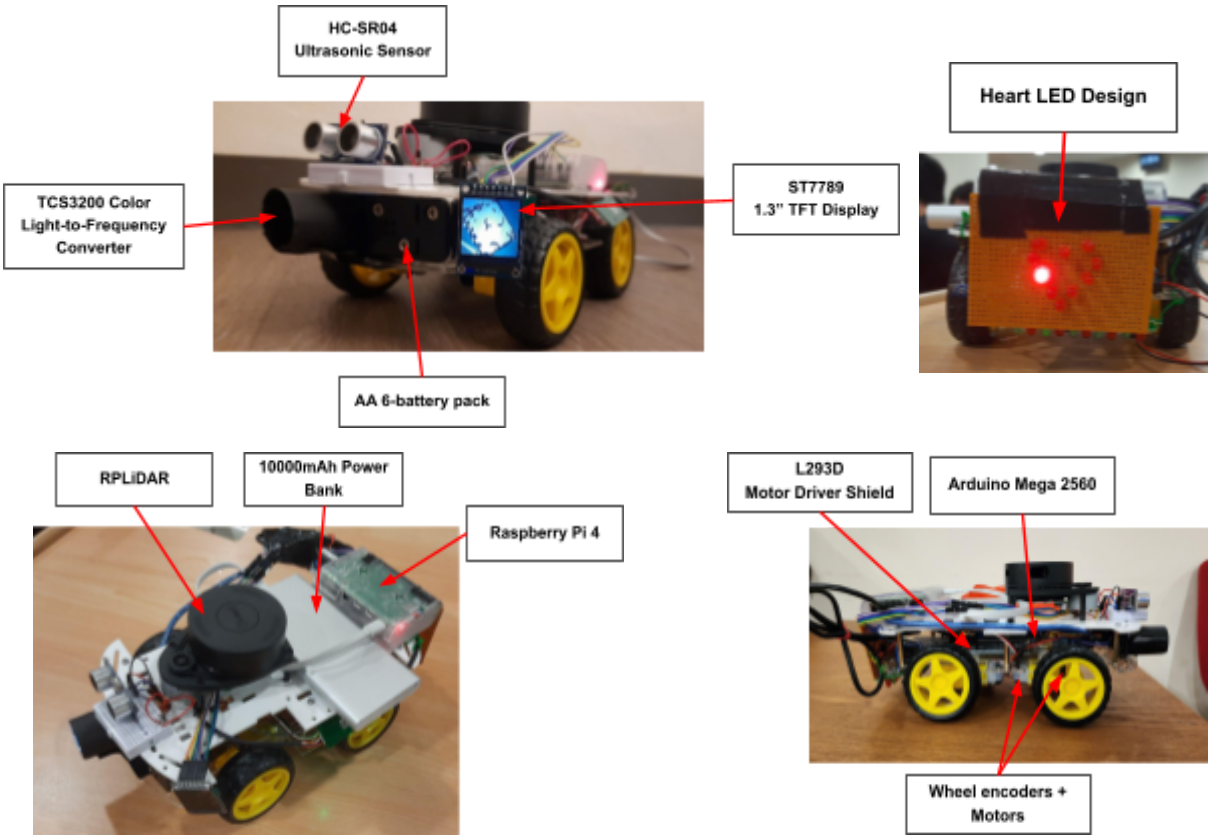
Section 3 System Architecture

The diagram below depicts the UML deployment diagram for Alex. Alex consists of 9 main hardware components: Arduino Mega, Motors, Colour Sensor, Wheel Encoders, Ultrasonic Sensor, Raspberry Pi, LiDAR, TFT LCD Display and a Laptop.



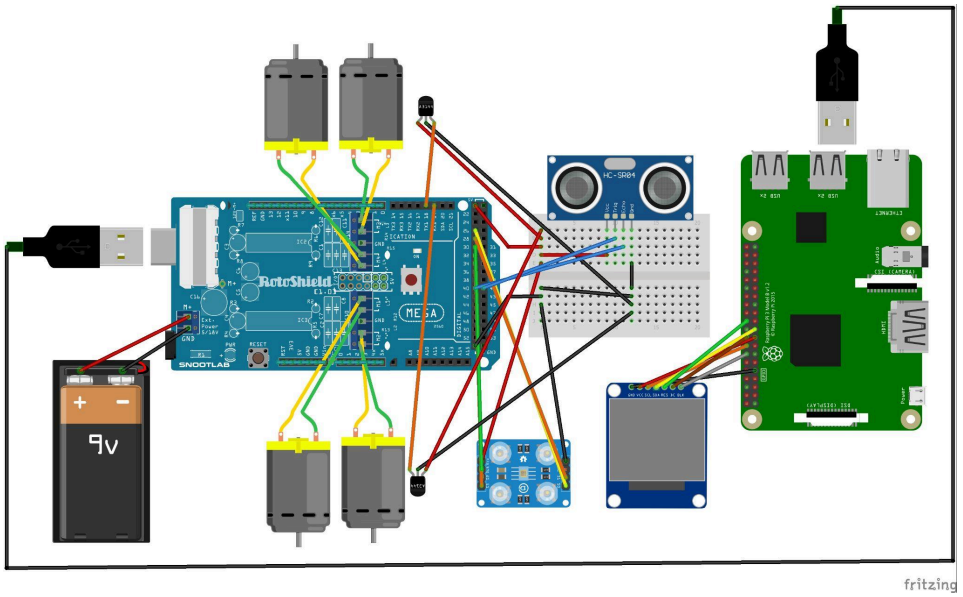
Section 4 Hardware Design

4.1 Pictures of Alex



4.2 Hardware components

Alex is built using a 2-layer 4-wheel drive robotic chassis. All the four wheels are connected to the motor driver shield and powered using 9V. This motor shield is then placed above Arduino Mega (referred to as Arduino), reducing the connections made using wires. Furthermore, the Arduino is connected to a tiny computer Raspberry Pi 4 model B (referred to as RPi). The schematic diagram for Alex is given below. Refer to Appendix 1 for detailed explanation of connections.



4.2.1 L293D Motor Driver Shield

The motor driver shield features two L293D motor driver ICs and one 74HC595 eight-bit shift register chip. With this shield and by using the Motor Shield Library by Adafruit, we can effectively control 4 DC motors easily. The initial 4-battery pack was not enough to drive the wheels as the power bank and LIDAR were too heavy. Hence we sourced a 6-battery pack from the lab. Now the shield is powered by 9V (6 * 1.5V) and the wheels move at a faster rate. Even then, the wheels had to be supplied with 100% power while turning, else the movement becomes largely unpredictable.

4.2.2 HC-SR04 Ultrasonic Sensor

The HC-SR04 Ultrasonic Sensor was placed directly above the TCS3200 Color Light-to-Frequency Converter to ensure that the distance it measured was relative to the color sensor and that the distance is not too far from the object such that the data collected by the TCS3200 was inaccurate due to the disruption by ambient light. The Ultrasonic sensor was also placed further inside on the chassis to ensure that its position and distance measured would not change in the case where we collide into anything.

4.2.3 TCS3200 Color Light-to-Frequency Converter

The TCS3200 Color Light-to-Frequency Converter was attached to the front of Alex so that we can move Alex directly in front of the target to identify the color. To secure the color sensor, we slotted it in one of the holes present on the bottom chassis and taped it down with masking tape at the back. We also made a cover for the TCS3200 to ensure that the data collected by the photodiodes had less interference from the ambient lighting, giving us more reliable color readings.

4.2.4 Wheel Encoders

The two front wheels of Alex have been connected to two wheel encoders to keep count of the distance moved by each wheel. The disk magnets on the wheel encoders have 8 magnetic poles. When a wheel makes one full rotation, the count (say ticks) is increased by 4 times. Therefore, the circumference of the wheels are noted and depending upon the distance to travel, a target variable (targetTicks) is set. So when the current count (currentTicks) reaches the target amount, the wheels stop moving, signifying that the distance is reached.

4.2.5 Laser Cut top Chassis

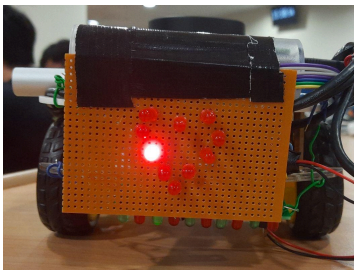
To secure the LiDAR on the top of the chassis, we measured the distance between the legs of the LiDAR and added additional holes (Appendix 5) to fit the LIDAR securely on the top chassis, ensuring that it would not move around and disrupt the mapping when it spins to collect data because it was not secure enough.

4.3 Additional Hardware features implemented

4.3.1 ST7789 1.3” TFT Display

Alex has a 1.3” ST7789 based TFT display. The display is intended as a channel for communicating with the victims and relaying important information to them. Safe escape routes and closest emergency exits are displayed to victims that are not injured. It also features motivational quotes to upkeep the spirit of the victims. The display works through an SPI communication protocol and is connected to the RPi. Refer to appendix for implementation details.

4.3.2 LED design in soldered protoboard

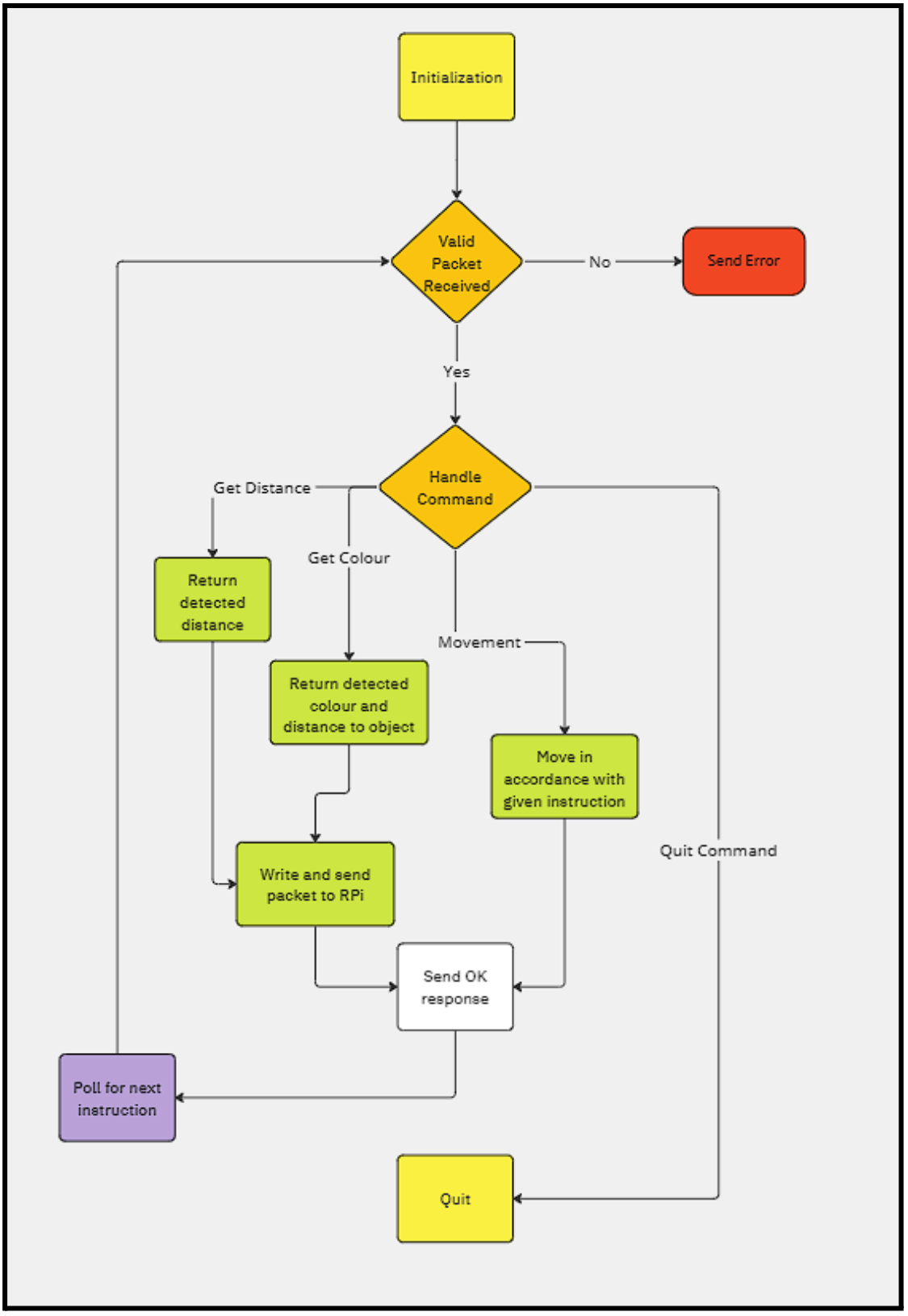


A cool feature that Alex has is a soldered protoboard which has red LEDs in the shape of a heart. The purpose of this heart shaped LED board is for any survivors that Alex might have missed out on to notice that Alex is here and call out to us. The heart shape can also show them that we care about them and will go back and rescue them. The LED design is an extension to the circuit designed for the studio related to soldering. It makes use of a 555 timer and a decade counter in a similar manner with a different arrangement for the LEDs. Refer to appendix for circuit design.

Section 5 Firmware Design

5.1 High level algorithm on Arduino

While the serial communication between Arduino and RPi is active, the Arduino repeatedly polls for incoming data packets and performs actions based on the flowchart below.



5.2 Communication between RPi and PC

Communication between RPi and PC is established using TLS (Transport Layer Security) Protocol, where data is sent over the internet after encryption. We are using the Rivest-Shamir-Adleman (RSA) encryption algorithm to generate the keys and certificates. This algorithm uses the public key for encryption and private key for decryption (What Is RSA?, 2024). There are 3 types of keys and certificates used for the project:

1. Certifying Authority (CA) key and certificate:

   This is the signing key used to sign all the other keys. The signing certificate from the key is generated using Secure Hash Algorithm (SHA256). This certificate is the public key and must be copied over to all the clients.
2. Alex's private key is created in a similar way. Now, using the CA key, a certificate signing request (CSR) and certificate is created.
3. The PC's private key, CSR and certificate are created in the same way as Alex's.

During the handshake process, Alex's certificate is sent over to the PC and the PC's certificate is sent over to Alex for authentication. This way only authenticated users can access Alex, preventing anyone from "kidnapping" Alex.

5.3 Communication between Arduino and RPi

The Arduino and RPi communicate via UART asynchronously.

```
startSerial(PORT_NAME, BAUD_RATE, 8, 'N', 1, 5);
```

The above line in alex-pi.cpp sets the byte size as 8, no check for parity and 1 stop bit (frame format of 8N1). Each data packet, TPacket defined in packet.h has a size of 100 bytes. After appending the magic number, data size and checksum, the final size of the packet for serializing is 140 bytes. Refer to Appendix 4 for TPacket and TComms structure. Since the baud rate is set to be 9600 bps, the time taken for RPi to send a command to Arduino is 140 * (10 / 9600) = 0.15s. Therefore, if the user sends commands with a time interval less than 0.15s (~ pressing key 7 times in one second), the sequence of serializing and deserializing could be altered causing "bad magic number" errors. This is discussed more in section 7.1.

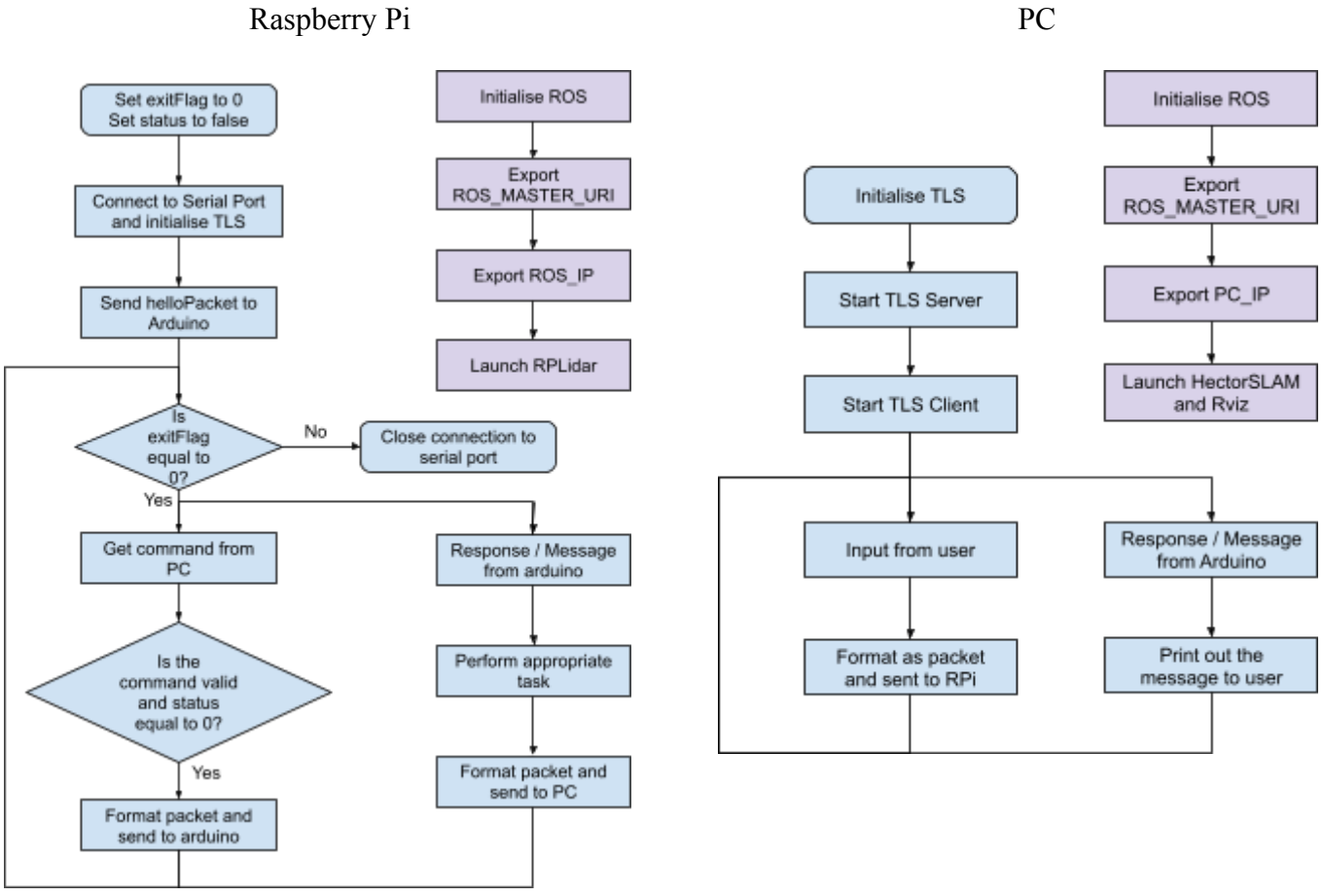
After executing each command, the arduino sends a response packet back to RPi. Depending on the type, these responses are further classified and stored in packetType.

Response	Description	PacketType
RESP_OK	When movement and other commands are received successfully	PACKET_TYPE_RESPONSE
RESP_STATUS	Sent in response to get stats command sent by RPi.	PACKET_TYPE_RESPONSE
RESP_BAD_PACKET	When Arduino gets a corrupted packet with the wrong magic number.	PACKET_TYPE_ERROR
RESP_BAD_CHECKSUM	When Arduino gets a corrupted packet with the wrong checksum.	PACKET_TYPE_ERROR
RESP_BAD_COMMAND	When the command received is not valid, but passes both the magic number and checksum.	PACKET_TYPE_ERROR

RESP_BAD_RESPONSE	When the response type is not valid	PACKET_TYPE_ERROR
RESP_COLOR	Sent in response to get color command by RPi.	PACKET_TYPE_RESPONSE
RESP_DIST	Sent in response to get distance command by RPi.	PACKET_TYPE_RESPONSE
The other packet types not related to response are: PACKET_TYPE_HELLO - to signify successful connection between Arduino and RPi PACKET_TYPE_MESSAGE - to send messages from Arduino to RPi PACKET_TYPE_COMMAND - to perform the movement and other commands		

Section 6 Software Design

6.1 High level Algorithm on Pi



6.2 Teleoperation

Teleoperation of Alex is done via TLS (Transport Layer Security) and has two modes: Auto and Manual. In Auto mode, each command will be recognised immediately without the need to press enter, while the manual mode requires confirmation in the form of enter before execution. The auto mode is implemented using getch() function. Refer to Appendix 3 for its working and benefits and section 7.1 for the problems faced during its implementation.

The complete list of commands used are described in the table below:

<table>
<tr><th>Command</th><th>Key</th><th colspan="3">Description</th></tr>
<tr><td>Forward</td><td>W</td><td colspan="2">Auto Mode</td><td>Manual Mode</td></tr>
<tr><td>Reverse</td><td>S</td><td rowspan="3">Low
5cm - 40%
10 deg - 100%</td><td rowspan="3">High
5cm - 50%</td><td rowspan="3">Parameters obtained from user:
- Distance or Angle
- Speed</td></tr>
<tr><td>Left</td><td>A</td></tr>
<tr><td>Right</td><td>D</td></tr>
<tr><td>Stop</td><td>R</td><td colspan="3">Stop the bot</td></tr>
<tr><td>Clear Data</td><td>V</td><td colspan="3">All global variables storing odometry data will be reset to 0</td></tr>
<tr><td>Get Stats</td><td>G</td><td colspan="3">Retrieve the number of turns made and distance moved by each wheel.</td></tr>
<tr><td>Get Color</td><td>C</td><td colspan="3">Retrieve R, G, B frequency values, ultrasonic distance and detected color based on color detection algorithm.</td></tr>
<tr><td>Get Distance</td><td>Z</td><td colspan="3">Retrieve Ultrasonic Distance</td></tr>
<tr><td>Toggle</td><td>M</td><td colspan="3">Toggle between auto and manual modes</td></tr>
<tr><td>Exit</td><td>Q</td><td colspan="3">Closes connection with Arduino by setting exitFlag to 1</td></tr>
</table>

6.3 Color Detection

Since Alex is a rescue robot, it should be capable of detecting the colors of objects before it. To implement this functionality without using a camera, we used TCS3200 Color Light-to-Frequency converter. This converter returns a square wave with frequency directly proportional to light intensity. Hence the frequency of light intensity when red, green and blue lights are radiated is noted and a color detection algorithm is framed. The main colors to be detected are white, red and green.

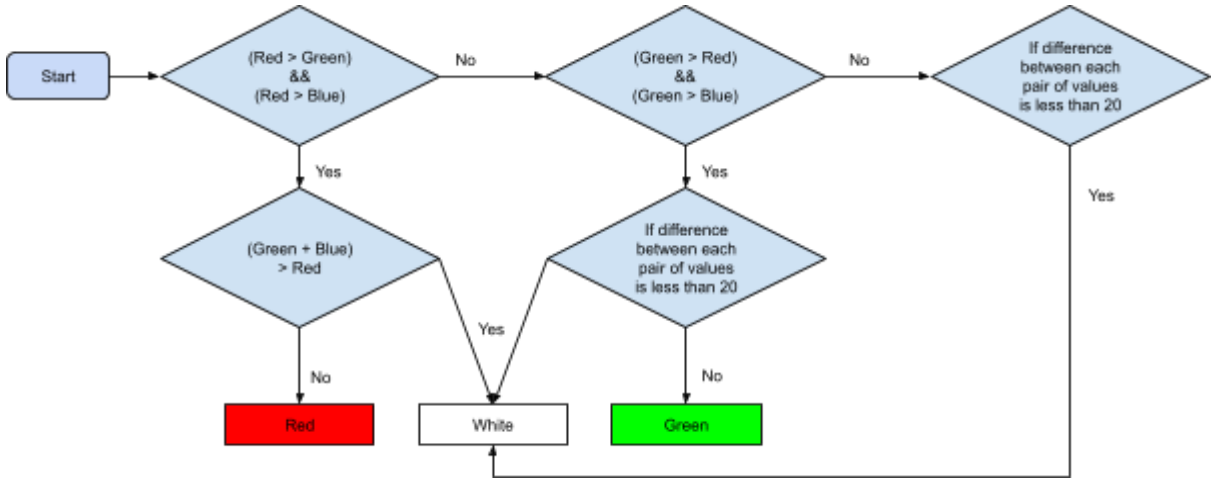


Fig 6.3.1 Color detection algorithm

The frequency of reflected light varied with distance between the sensor and object. The values were unreliable when the distance was greater than 15 cm. Hence the color packet would send the ultrasonic distance along with it and the user would decide whether to use the information or not. We also covered the sensor with black paper for effective shielding.

6.4 Mapping

In order to navigate through the room, mapping and localization is required to create a floor plan and identify the position of Alex while it maneuvers around the obstacles. Hence, Simultaneous Localization and Mapping (SLAM) is used to achieve the navigation, particularly Hector SLAM as odometry data is not required for the localization.

Three ROS nodes were created. The first node (rplidar) was required to broadcast the readings obtained from LiDAR, the second node to subscribe to the broadcasted topic to receive the readings, and the third node to render the map.

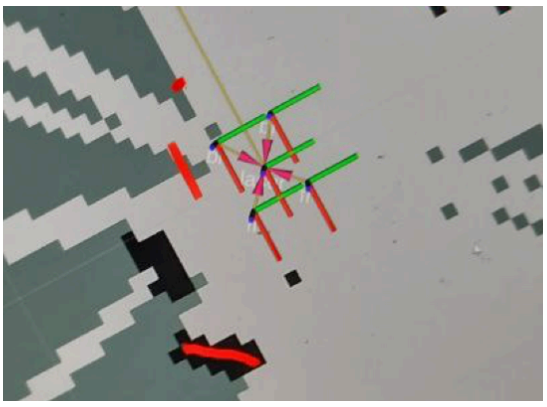
In order to offload the map rendering to the PC, only the rplidar node would be launched on the RPi. The PC would be configured as the master node, to receive the LiDAR data that is broadcasted over port 11311 after the rplidar node is started. The final node would then be launched on the PC as well for the hector mapping node to perform the SLAM algorithm and render the map on Rviz. This would reduce the computational load on the RPi.

6.5 Additional Software features

One additional feature that was added was a model of Alex's dimensions on Rviz. The arrow created by "pose" was able to indicate the direction that Alex was facing, but was not able to show the size of the robot. Instead, the dimension of Alex was implemented using four additional transform frames. The distance of Alex's four corners were measured from the LiDAR in the x- and y-direction. Each transform frame was created with an x and y offset with reference to the LiDAR's frame, with the frame ID "laser", using the values measured previously. The code is added to hectormapping.launch as shown below.

```
 <node pkg="tf" type="static_transform_publisher"
name="link2_broadcaster" args="0.12 -0.11 0 0 0 0 laser fl 100" />
 <node pkg="tf" type="static_transform_publisher"
name="link3_broadcaster" args="0.12 0.11 0 0 0 0 laser fr 100" />
 <node pkg="tf" type="static_transform_publisher"
name="link4_broadcaster" args="-0.16 -0.11 0 0 0 0 laser bl 100" />
 <node pkg="tf" type="static_transform_publisher"
name="link5_broadcaster" args="-0.16 0.11 0 0 0 0 laser br 100" />
```

This results in the model of Alex on Rviz as shown in the image below.



Once an Rviz map was created with the added model, the configuration is then saved in an Rviz file, named setup.rviz. The file view_slam.launch is edited such that the map would be launched with the model automatically added for us. This allows us to save time when setting up the map, as we do not need to configure and toggle the transform frames manually. The edited view_slam.launch is included below.

```
<launch>
  <include file="$(find rplidar_ros)/launch/hectormapping.launch" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
rplidar_ros)/rviz/setup.rviz" />
</launch>
```

Section 7 Conclusion

7.1 Lessons Learnt

7.1.1 “Bad Magic Number”

With the usage of getch() instead of scanf(), the packet pieces are sent at a much faster rate now. Hence, when a packet piece reaches the buffer when it is full, the new packet piece is dropped without any error message. This loss of intermediate packet pieces causes the final packet after rearranging to become corrupted and returns “bad magic number.” Subsequent packets would also give the same error due to the dropped piece, and the connection has to be forcefully stopped using ^ Ctrl C to resolve the issue. This problem is faced when the keys are pressed with a high frequency. Forcefully stopping the connection is concerning as the time taken to reconnect to the serial port and arduino is high and should not be wasted.

To solve this issue in a friendlier way and reduce the time taken to repeatedly establish connection to arduino, a global variable “status” is maintained. This variable is set to true when a command is sent. It is set back to false only after RPi receives an OK packet from Arduino. Moreover, new commands are sent to the arduino only when this status variable is false. After implementing this comparison, now we can even hold onto any key without causing any error. Fortunately, we identified this problem before our trial run and we were able to solve it before the test.

7.1.2 Unable to communicate with LIDAR

Rendering a SLAM map from LIDAR data requires extensive calculations and may slow down the RPi. To solve this issue, the tutorials in the labs taught us how to offload the LIDAR data processing to the PC and allow the PC to render the map (Chen Guoyi - CG2111A Tutorial, n.d.). To do this, we need a windows subsystem on a linux system. So we created a virtual machine for the same purpose. But even after following the tutorial steps, we could not get the LIDAR data to be displayed. When we consulted our professor, he asked us to change the networking type from NAT (Network Address Translation), the default setup, to Bridged Network.

In NAT, the virtual machine (VM) is connected independently to a private network. This enables the VM to access external networks via the host IP address. Hence the VM is not connected to the same internet connection as LIDAR (Khan, 2023). This caused the problem of not being able to send the data to the VM. Our professor explained the scenario by printing

out the IP addresses of RPi and VM. Comparing both the addresses, their network ID was not the same. Hence we changed the settings to enable bridged networking. In bridged networking, the VM has direct access to the network.

7.2 Mistakes Made

7.2.1 “Angle change too large”

At the initial stages of the project, we often receive the message “Angle change too large” in the LiDAR readings. This caused the map on Rviz to appear distorted and thus unreliable, as we were no longer sure of Alex’s orientation and position on the map .

From our observation, this was due to two main factors. First, the power that was used when turning the motor was at 100%, which caused Alex to turn at a very fast speed in Auto mode. This change in surroundings is too abrupt for Hector SLAM. Hence, we tried to reduce the fixed speed, so that Alex will turn gradually. However, Alex is not able to turn reliably with a lower power, as the motors are not able to overcome the friction torque and start spinning. This meant that we could not lower the power to turn Alex, which leads to the next factor that caused the message we received.

Second, the message was also due to the robot turning at a large angle. The angle we initially fixed at Auto mode was 90 degrees, which could mean that the robot was turning at an angle that was too large. Since, we could not change the speed, we could only adjust the angle to 10 degrees, so that the turn would not be too sudden a change for the mapping. Despite the short, jerky movements from the high power, the small turning angle was sufficient to ensure Alex turned without compromising the quality of the map on Rviz.

7.2.2 Bad wire management

While preparing Alex, we had failed to think of what we wanted to do for wire management beforehand. Our only consideration in the beginning was the wires blocking the view of the LIDAR and hence, the only thing we did for that was to push the wires down such that it did not block the view of the LIDAR. By doing so, we ended up with some wires protruding out of Alex’s chassis by a lot, which resulted in the overall width increasing. This made Alex more likely to hit the obstacles placed in the maze. The dimensions for Alex we put in the Rviz were also less reliable since Alex’s size was larger than we had measured it to be initially.

This issue was mainly caused by insufficient planning of the positions of the components we had. We also did not take into account the length of the wires we were given for Alex and did not think of how we could place the wires within the chassis.

Looking back, we should have planned out where we wanted to place every component before we assembled Alex to maximize the use of all the space between the top and bottom chassis for wire management so that we would not have to resort to last minute solutions such as duct tape to secure the wires in place.

References

ANYmal—Autonomous Robotic Inspection Solution—ANYbotic. (n.d.). Retrieved March 28, 2024, from https://www.anybotics.com/robotics/anymal/

Husky UGV - Outdoor Field Research Robot by Clearpath. (n.d.). Retrieved March 28, 2024, from https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/

What is RSA? How does an RSA work? | Encryption Consulting. (2024, March 4). https://www.encryptionconsulting.com/education-center/what-is-rsa/

Answer to "Capture characters from standard input without waiting for enter to be pressed." (2009, May 26). Stack Overflow. https://stackoverflow.com/a/912796

Khan, N. (2023, June 17). *VMware Bridged vs. NAT: Choosing the Right Network Configuration*. NextdoorSEC - Penetration Testing Worldwide. https://nextdoorsec.com/vmware-bridged-vs-nat/

What is RSA? How does an RSA work? | Encryption Consulting. (2024, March 4). https://www.encryptionconsulting.com/education-center/what-is-rsa/

Appendix

1. Alex Connections

First, the L293D Motor driver shield is placed directly above the Arduino Mega such that all the connections are in place. Other pin wiring for the hardware components used are given below.

Colour Sensor	
TCS3200 pins	**Connections**
OUT	Arduino pin 30
S0	Arduino pin 26
S1	Arduino pin 28
S2	Arduino pin 32
S3	Arduino pin 34

Ultrasonic Sensor	
HC-SR04 pins	**Connections**
VCC	Arduino +5V
GND	Arduino GND
TRIG	Arduino pin 41
ECHO	Arduino pin 40

Wheel Encoders	
Hall effect pins	**Connections**
VCC	Arduino +5V
GND	Arduino GND
Right Encoder	Arduino pin 19
Left Encoder	Arduino pin 18

TFT Display	
ST7789 pins	**Connections**
VCC	Pi +3.3V
GND	Pi GND
SCL	Pi pin 23 (SCLK)
SDA	Pi pin 19 (MOSI)
RES	Pi pin 22 (GPIO 25)
DC	Pi pin 18 (GPIO 24)
BLK	Pi pin 13 (GPIO 27)

2. Code

The code base is uploaded on GitHub and can be accessed with the following link: https://github.com/roshnidaksha/CG2111A-Alex.git

The repository is organized into directories. All the files under RPi have to be stored under the same folder in RPi. The files under Alex are .ino files. These have to be compiled together, and ensured that they are recognised as tabs for the program to compile successfully. With respect to the TLS directories, the laptop or PC used is the client and RPi is the server.

3. ST7789 1.3” TFT Display without CS pin

Library used : https://github.com/solinnovay/Python_ST7789

Datasheet : https://www.mouser.com/datasheet/2/744/ST7789VW-2320339.pdf

Connections : Refer Alex Connections

Code : Refer Code

4. getch()

getch() is a non-standard function from the conio.h header file. It is similar to reading a single character using scanf() but does not use any buffer. So the input character is immediately returned without waiting for an enter key.

Implementation: (*Answer to "Capture Characters from Standard Input without Waiting for Enter to Be Pressed,"* 2009)

```
#include <termios.h>
char getch() {
      char buf = 0;
      struct termios old = {0};
      if (tcgetattr(0, &old) < 0)
                perror("tcsetattr()");
      old.c_lflag &= ~ICANON;
      old.c_lflag &= ~ECHO;
      old.c_cc[VMIN] = 1;
      old.c_cc[VTIME] = 0;
      if (tcsetattr(0, TCSANOW, &old) < 0)
                perror("tcsetattr ICANON");
      if (read(0, &buf, 1) < 0)
                perror ("read()");
      old.c_lflag |= ICANON;
      old.c_lflag |= ECHO;
      if (tcsetattr(0, TCSADRAIN, &old) < 0)
                perror ("tcsetattr ~ICANON");
      return (buf);
}
```

Advantages: The runtime is increased to some extent as the user does not have to move his hands to the enter key to execute the commands.

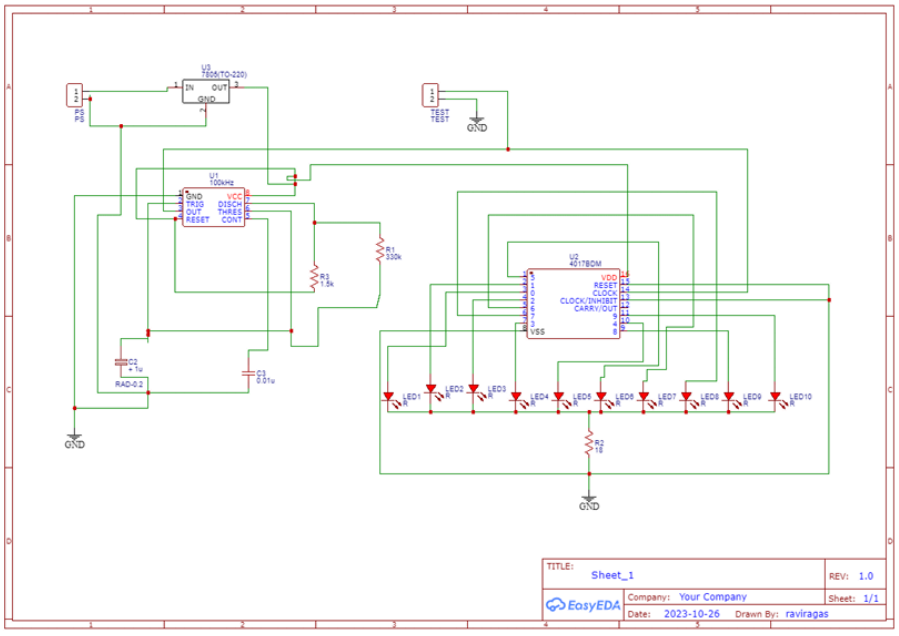
5. TPacket and TComms structure

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

```
#define MAX_DATA_SIZE  128
// This packet has 4 + 4 + 128 + 1 + 3 = 140 bytes
typedef struct comms
{
     uint32_t magic;
     uint32_t dataSize;
     char buffer[MAX_DATA_SIZE];
     unsigned char checksum;
     char dummy[3];
} TComms;
```

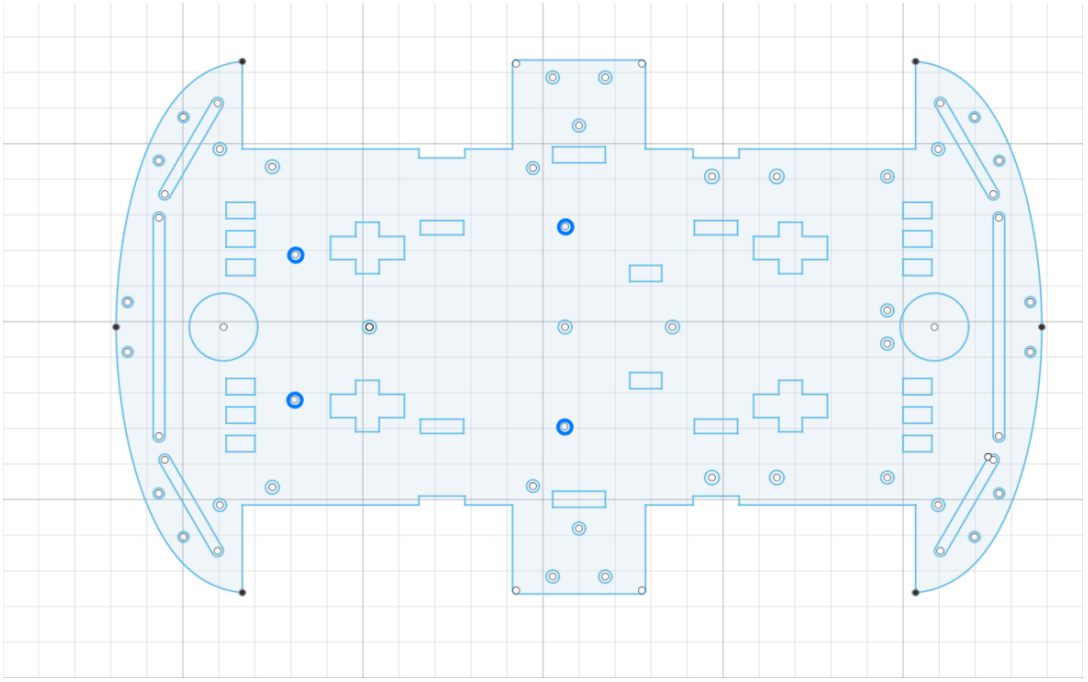
6. Circuit design for LED Pattern

Credit - CG2111A teaching team, Week 5 Studio 2 handout



Schematic Design of Circuit

7. CAD Model of Chassis



CAD model of original chassis provided by the teaching team + additional dark blue round holes to fit LIDAR