CG2111A Engineering Principle and Practice II

Semester 2 2023/2024

“Alex to the Rescue”

Final Report

Team: B02-2A

Name	Student #	Main Role
Hua Zhenting	A0283162H	Member
Tram Minh Man	A0276034H	Member
Felix Yuen Pin Qi	A0272059B	Member
John Woo Yi Kai	A0272561E	Team Lead
Paul Tham Yong Le	A0271560J	Member

Section 1 Introduction

Our mission is to build a new robot called Alex with search and rescue functionalities.

Alex will be tele-operated (remote-controlled) from our laptop. As the robot moves, we can create an environment map relayed to the operator by using the information received from LiDAR. The operator is then able to utilise the map for manual navigation within the simulated environment.

We will interact with a master control program (MCP) on the Pi. This MCP will subsequently convert our instructions into actual movement commands for the connected Arduino board.

Furthermore, there will be 2 victim objects (with cube or cylinder shape coloured with red and green) and 1 dummy object coloured with white distributed across the room. Our team needs to identify those objects in our map, navigate Alex closer to these objects to figure out the colour remotely and the measured colour will be then relayed back to the operator. If the victim objects are detected, a message will be written on the LCD screen.

In the simulation, Alex is tasked to navigate the maze within 6 minutes, to correctly identify 2 victims and 1 dummy victim while minimising collisions with the walls and then park at the required location.

Alex's overall functionalities:

- Go straight/reverse (we can define how far/ how long, speed control)
- Go left/right (we can define the turning angle or the compass direction)
- Identify the victim objects and non-victim objects based on the colours. Finally, if victim objects are detected, Alex will display messages on the LCD screen.
- Accurately map out the layout of the room. The environment mapped out by Alex will then be noted down.

Section 2 Review of State of the Art

2.1 Deep Robotics' Jueying X20

Jueying X20[3] is a quadruped robot that is designed to assist emergency responses and locate lifeforms in any type of emergency situation. It can traverse rough terrains and detect hazards such as harmful gases and heat radiation. It can also conduct heavy-loaded tasks and can construct a 3D model of the environment.

Hardware Components	**Software Components**
Gas sensor that provides real-time concentration alarms Bi-Spectrum PTZ Camera Sound Pickup that receives sound and connects rescue calls to victims	Uses LIDAR and SLAM algorithms to model the terrain regardless of scenario AI algorithm for dynamic obstacle avoidance and rapid detection of hazard situations

Strengths: The X20 can carry heavy loads of up to 85kg, such as oxygen tanks and rescue supplies, which are vital for emergency operations. It can also traverse in extreme weather and hazardous conditions with IP66 protection. It is also able to take photos, and utilises LIDAR and SLAM to create accurate 3D models of surroundings.

Weaknesses: Its size restricts the robot from entering tight spots, hence restricting its effectiveness in going through rubble. The additional weight may also cause the battery life to deplete faster, hindering operations.

2.2 Dragon Runner

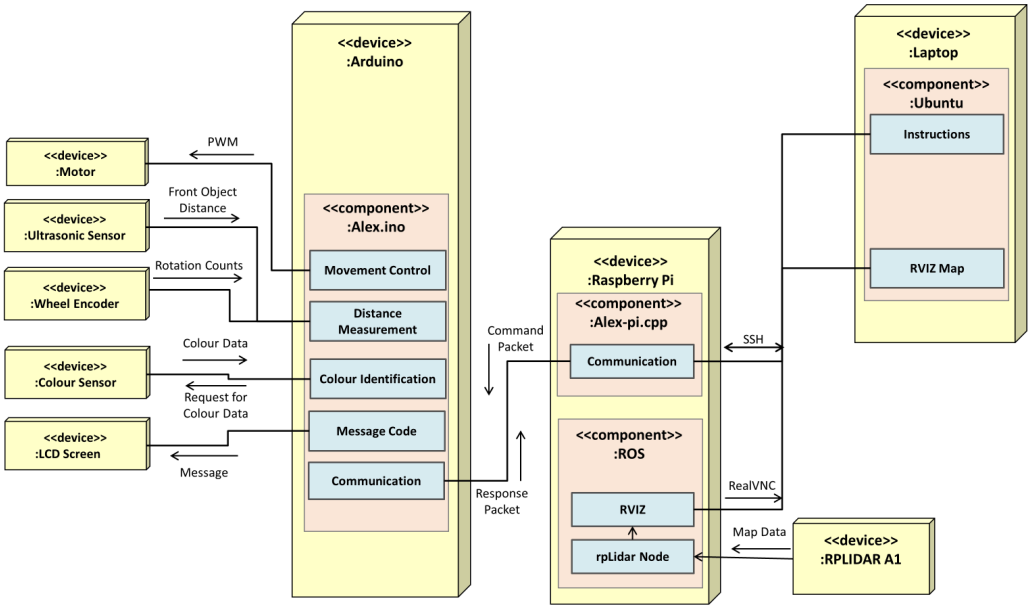
Dragon Runner is a small throwable robot developed by QinetiQ North America. It is designed for urban reconnaissance and surveillance missions, particularly suitable for navigating narrow spaces and complex environments, with a range of functions including searching, rescuing, and mapping.

Hardware Components	**Software Components**
Cameras for object recognition LiDAR sensors for creating 3D maps of its surroundings and obstacle detection Proximity sensors and motion detector for obstacle avoidance and collision detection Thermal imaging camera for locating survivors	SLAM algorithms to create detailed maps of the surrounding AI integration for autonomous navigation, object recognition, decision-making, and adaptation to different scenarios

Strengths: The Dragon Runner is able to work in hazardous environments and reach areas inaccessible to human rescuers due to its relatively small size. Its advanced sensor suite and path-planning capabilities also enables it to locate and rescue victims quickly.

Weaknesses: It has limited payload capacity due to its compact size and lightweight design. It also has relatively high acquisition cost and ongoing maintenance expenses

Section 3 System Architecture



Section 4 Hardware Design

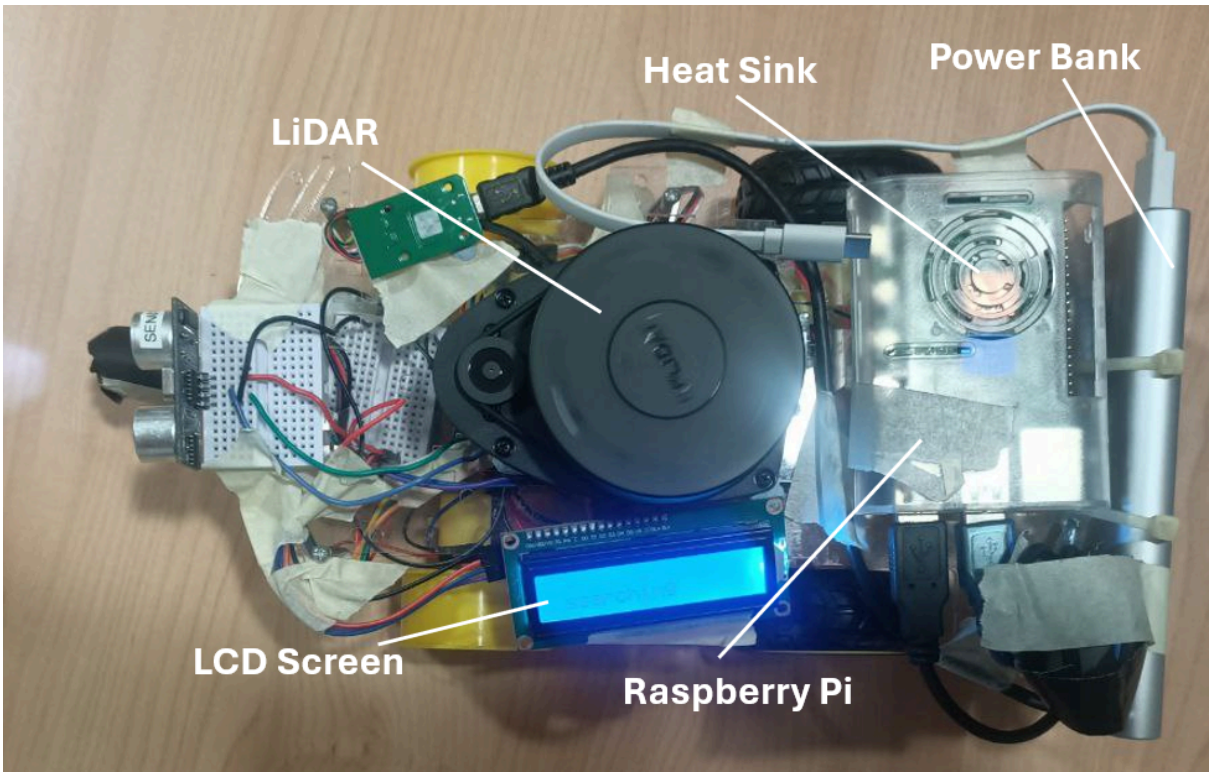


Fig 4.1. Top View of Robot

4.1 Weight Distribution

During the testing of the robot manoeuvrability, we had initially placed the power bank directly underneath the LiDAR, however we found out the weight is too concentrated on the chassis of the robot, and it is harder for the robot to make turns effectively. Hence we made the choice to put the power bank at the very end of the robot, as seen in Fig 4.1, to distribute the weight more evenly which makes it easier for the robot to make turns. We had also attached the batteries to the bottom of the robot, as seen in Fig 4.2, so that we have more space to work with on our robot.

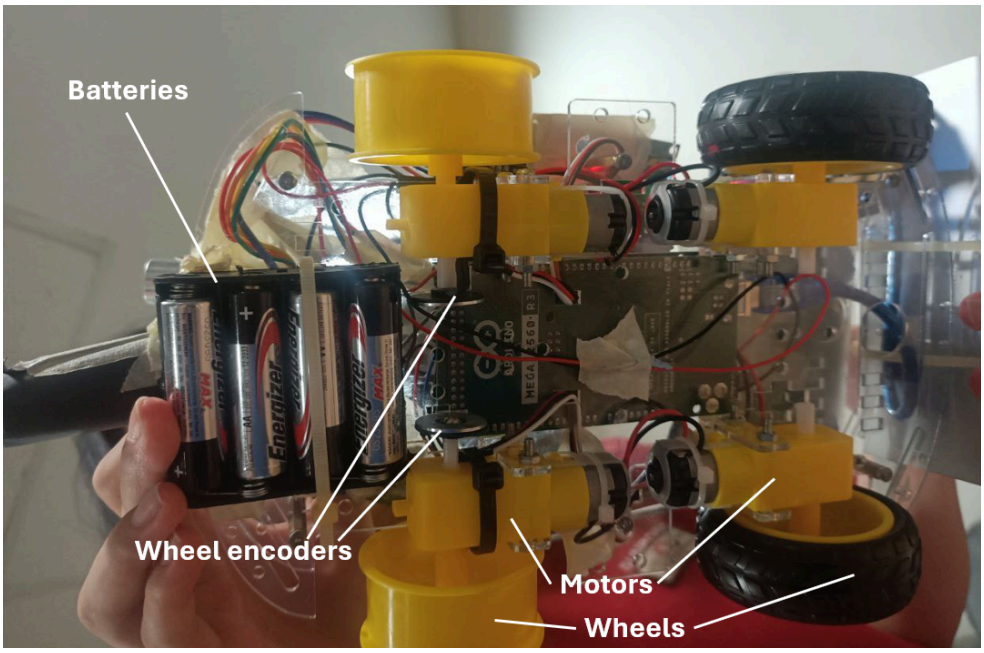


Fig 4.2. Bottom view of robot

4.2 Ultrasonic Sensor

The ultrasonic sensor is implemented at the front of the robot, allowing it to return the distance of the obstacle or walls in front of the robot. Moreover, the placement of the ultrasonic sensor is vertically aligned, so the distance measured corresponds to the distance between the colour sensor and the obstacle. This allows us to better track and control the distance of colour detection, enabling more precise detection.

4.3 Colour Sensor

The colour sensor is wrapped in black paper, to form a cylindrical shape with open ends.This is to prevent ambient light from affecting our results of colour detection, so that the colour sensor is more sensitive to the colour of the objects and can identify them more accurately. The length of the cylindrical paper was chosen to ensure we get a high accuracy of colours detected, while also not making it too long that it bumps into the obstacles.

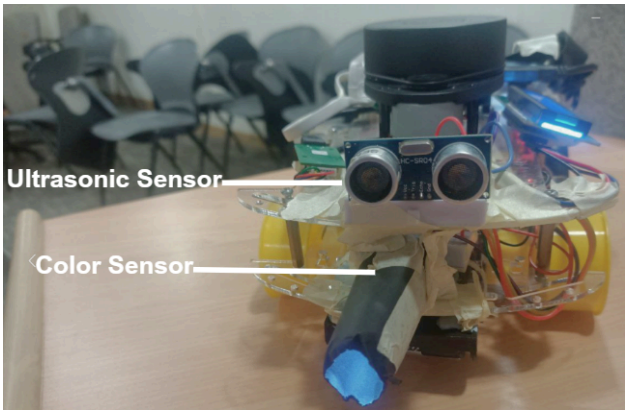


Fig 4.3. Front View of Robot

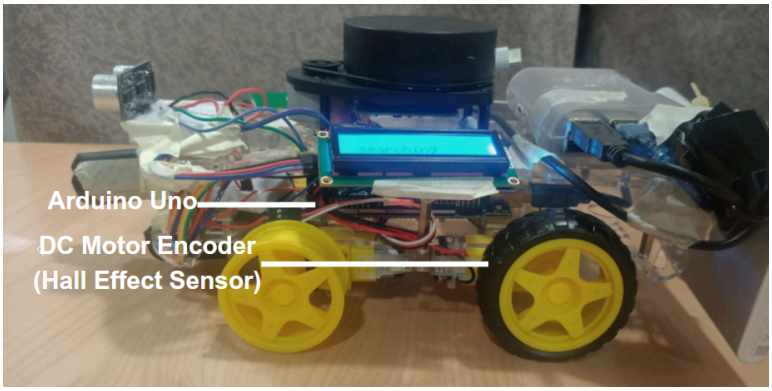


Fig 4.4. Side View of Robot

4.4 Additional Component (LCD Screen)

We have also included an LCD screen which can display any message that we typed in. For example, it can display “searching”, as seen in Fig 4.5, while searching for the objects.

Fig 4.5. Close-up on LCD Screen

Section 5 Firmware Design

5.1 High level algorithm on the Arduino Mega

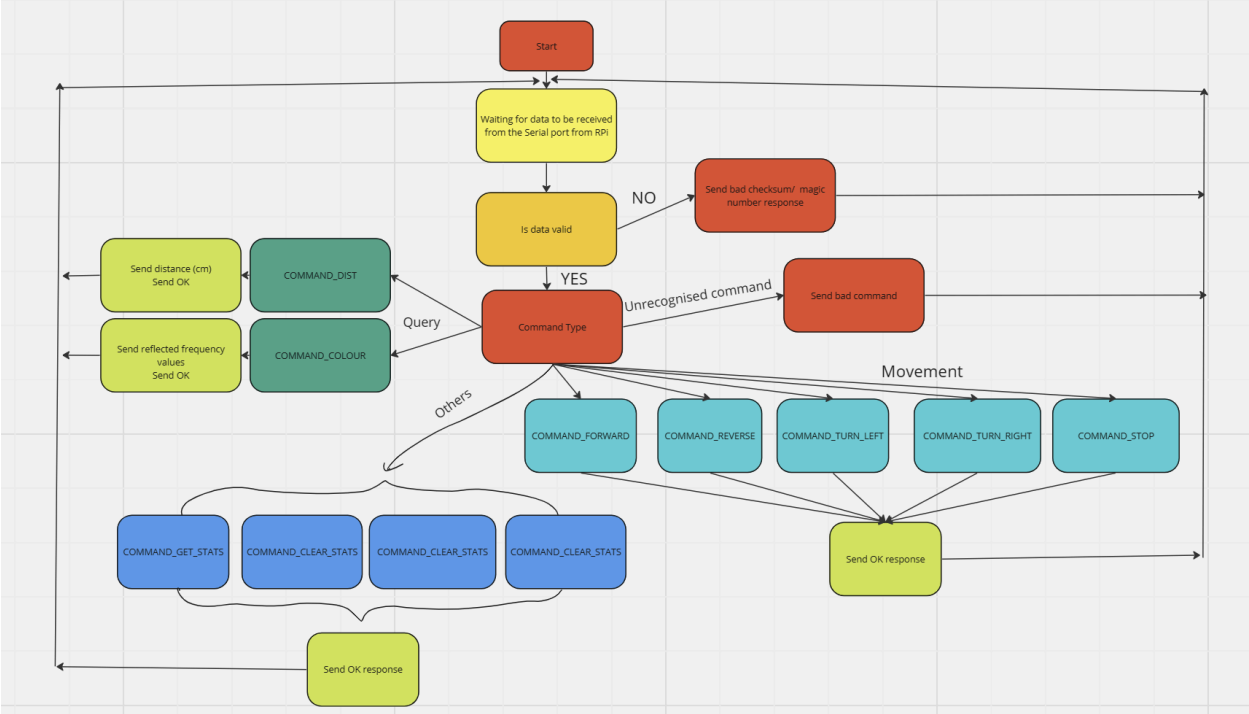


Fig 5. Flow chart of high level algorithm

5.2 Communication protocol

The Arduino and RPi communicate asynchronously via UART with the standard 8N1 frame format with a baud rate of 9600 bps. Each data packet is 100 bytes long. When Arduino receives data from RPi, it deserializes the data and stores it as TPacket type. Refer to Appendix 9.1, which describes the format of the TPacket packet. When the Arduino completes executing a command, it sends a response packet to the Pi. Refer to Appendix 9.2, which describes the different possible packets sent to Pi.

5.3 Additional noteworthy software-related features

5.3.1 Obtaining frequency values of the objects' colour

On the Arduino side, we stored the frequency values measured from the colour sensor in the params of TPacket and sent this packet via Serial port to RPi. On the RPi side, it will detect the command of this packet as 'RESP_COLOUR' and then display the frequency values on the client side.

5.3.2 Sending over messages

On the RPi side, messages are stored in the data bytes and sent out via Serial Port to the Arduino. On the Arduino side, it will detect this command of the packet as 'COMMAND_DISPLAY', and take the message stored in data bytes and display it on the LCD screen.

Section 6 Software Design

6.1 High level steps for Raspberry Pi

1. Initialise communication between Raspberry Pi (RPi), Arduino and PC
2. Generate mapping using RPLidar inputs and Hector SLAM
3. User issues commands based on data
4. Arduino processes commands
5. Repeat steps 2-4 until Alex reaches the parking spot.

6.2 Ultrasonic sensor

An ultrasonic sonic sensor was connected to the front of Alex, to give the driver information on how close Alex is to an object/wall in front of him. We connected the ultrasonic sensor to the Arduino Uno and sent the information received by the Arduino Mega back to the RPi.

1. When COMMAND_DIST is received by the Arduino Uno, the function sendDist() is run.
2. sendDist() runs another function getDistUltra() which writes the right signals to the ultrasonic sensor and returns the distance it calculated.
3. sendDist() then sends a packet containing the distance back to the RPi which will be displayed for the driver to see.

6.3 Colour detection

A GY-31 colour sensor was used to detect the colours red, green and white. This was done by connecting the GY-31 sensor to the Arduino Mega, getting the necessary information and then sending this information in packets back to the RPi to process.

1. When COMMAND_COLOUR is received by the Arduino Mega, the function getColour() is run, followed by the function sendColour().
2. The function getColour() sets the photodiodes to colours of red, green and blue and the average value of five readings of each colour is taken to be the colour frequency of the respective colour.
3. sendColour() then takes the value from getColour(), as well as the distance to the front using getUltraDist() and sends it in a packet to the RPi for processing.
4. The RPi handles the response type RESP_COLOUR by running the function handleColour().

5. This function first checks whether the distance to the front is small enough for our colour detection to be consistent, else it will inform the driver to move closer. On the other hand, when we are close enough, a simple check of the values of the different frequencies are done to determine which colour is in front of Alex. The colour detected is then displayed for the driver to see.

6.4 LCD Screen

A LCD1602 display was used to display messages sent over by the RPi. This was done by connecting the LCD screen to the Arduino Mega, and having the RPi send over a message packet to the Arduino Mega to process.

1. When COMMAND_DISPLAY is received by the Arduino Mega, the function write_message() is run.
2. The function write_message() sends over a pointer to the data bytes, which contain a string of 32 chars long.
3. write_message() then displays the string stored in the data array..
4. Arduino Mega sends a TPacket with "RESP_OK" to signal the message is displayed.

6.5 RPLidar and RViz features

When the RPLidar is subject to high turn rates, it tends to generate unwanted artefacts and map drifting as the map cannot keep up with the SLAM mapping. As we wanted to reduce the frequency of this occurring, we reduced the map sensitivity by increasing "map_update_distance_thresh" and "map_update_angle_thresh" in the launch file. To increase the quality of our map, the variable "map_resolution" was also decreased to increase map fidelity and aid in victim recognition. Boost mode was also selected on the RPLidar, increasing the scan speed of our RPLidar for better map accuracy.

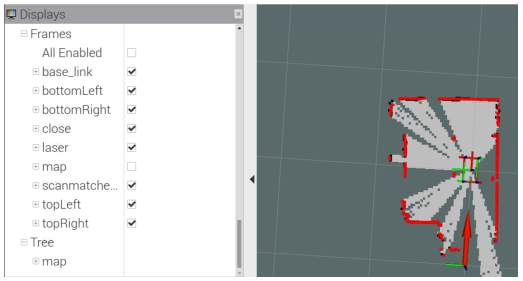
```
<param name="map_update_distance_thresh" value="0.04"/> <!-- Originally 0.02, now 0.04 -->
<param name="map_update_angle_thresh" value="0.3"/> <!-- Originally 0.1, now 0.3-->

<param name="map_resolution" value="0.025"/> <!-- Originally 0.05, now 0.025 -->
```

```
 <param name="scan_mode"           type="string"  value="Boost"/>
<!-- Boosts our sample scan speed by 4x versus standard mode -->
```

Figure 6.1: hectormapping.launch and rplidar.launch configurations

When it comes to representing Alex on RViz, we were initially given a simple arrow to display the pose on the map in RViz. However, the arrow is not an accurate representation of our robot on the map. Hence, additional markers were added that were positioned off the position of the LIDAR using the Transform Frame (TF) feature. Each marker is represented as 3 axes, which were rotated to represent the corners of Alex. The red axes from the RPLIDAR helped represent

the colour sensor. The following code snippet was added to view_slam.launch, and the display on RViz is shown below.




```
  <node pkg="tf2_ros" type="static_transform_publisher" name=
"marker1" args="0.13 -0.06 0 1.57 0 0 laser topRight" />
  <node pkg="tf2_ros" type="static_transform_publisher" name=
"marker2" args="0.14 0 0 1.57 0 0 topRight topLeft" />
  <node pkg="tf2_ros" type="static_transform_publisher" name=
"marker3" args="0.25 0 0 1.57 0 0 topLeft bottomRight" />
  <node pkg="tf2_ros" type="static_transform_publisher" name=
"marker4" args="0.14 0 0 1.57 0 0 bottomRight bottomLeft" />
  <node pkg="tf2_ros" type="static_transform_publisher" name=
"marker5" args="0.25 0 0 1.57 0 0 bottomLeft close" />
```

Figure 6.2: Transform Frame Markers Settings

Section 7 Lessons Learnt - Conclusion

7.1 Important Lessons

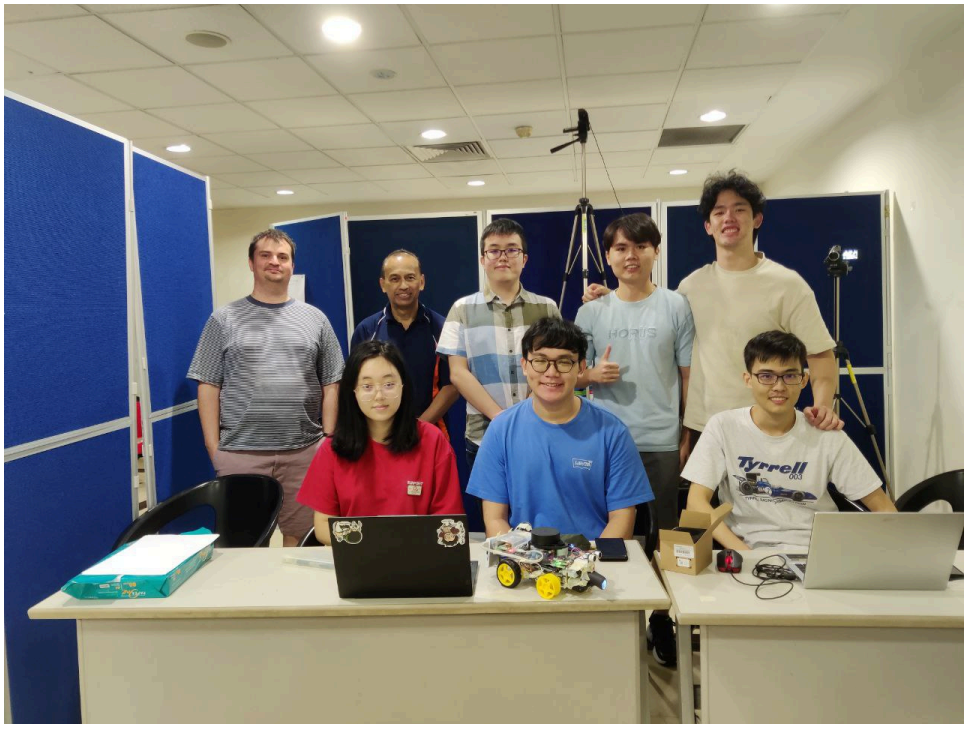
One lesson which we had learnt from this project, is the importance of weight distribution and how we should always apply the fundamentals of physics into our projects. When we were figuring out how to turn our robot with the newly added load of the powerbank and the LiDAR, we realised that we were having a lot of difficulty. Thankfully, we were able to deduce from our understanding of torque and force production in the wheels to see how that when our centre of gravity was in the middle of Alex, this led to greater friction in the front wheels as more weight were on them and the single motor of Alex could not overcome this force, hence Alex was not able to turn. Once we realised this, we instantly knew we had to place the powerbank at the back as a counterweight, so less friction would be experienced on the front wheels, which allowed it to turn more smoothly. Additionally, we reduced the friction of the front wheels by removing the black rubber and leaving only the yellow rims. This allowed the motors at the back to face less resistance and move about as required, making more effective turns as well. Through this we realised how basic and fundamental knowledge of physics could be applied to real life projects and how when mastered and applied properly, it can lead to the simplification of the process, as well as allowing us to engineer a solution more quickly.

Another lesson we had learnt from this project is how to be resourceful when making the robot. We had more freedom to add components to the robot as compared to the CG1111A project, and this gave us the chance to customise the robot to our liking to make it more efficient. For example, we noticed that the robot heats up quickly, so we included a heat sink into the Raspberry Pi so that the heat emitted does not affect the efficiency of the Raspberry Pi. We also utilised a LCD screen as said before in 4.4, This allowed us to not only think outside of the box in using components we never used before, but we are also forced to research and read up on the datasheets on different components. This helps us to prepare us for future projects when we have to utilise foreign components that we have never used before.

7.2 Greatest Mistakes

One mistake we committed during the construction of Alex was the improper management of the components, which had a trickle down effect and affected the rest of the process. It was the lack of forethought when constructing Alex that had led to this mistake. Initially, we were too focused on getting the functionalities of Alex to work. So much so that there was a disregard for the management of wires and the electrical components. Although we did manage to get all our functionalities working reasonably fast, the lack of management of the wires and electrical components began to cause us problems. Because our wires and components were all entangled in Alex, this made troubleshooting and adding new features extremely difficult and time consuming for us. Whenever we faced issues with the different components, such as the ultrasonic sensor as well as the colour sensor, it was very difficult to individually check which part of the wiring had gone wrong. This was a result of the entanglement of the wiring, so when we wanted to pull out a component to run tests, this would mean removing other components as well which led to a lot of unnecessary work. Eventually, we realised how counterproductive this was and began to tape up the wires of the same components together which help us to reduce time wastage when troubleshooting.

Another mistake that we made was with the colour sensor. Initially, the values of the RGB frequencies were unreliable when testing, and we tried all sorts of ways to reduce this uncertainty. We tried adjusting the position of the LED lights on the sensors, and even added black paper around the photodiode to prevent ambient light interference. Although the black paper did help to make the values more reliable, we were still having issues with calibrating the colour ranges. However, after putting the coloured objects at different locations did we realise that the distance was the biggest factor in determining the range. Thus, we calibrated our readings to the set distance of 5-8 cm as it gave us the most distinct and consistent colour ranges. We then used the ultrasonic sensor to ensure that our robot was in position before scanning for colours, thus ensuring we would always get the most accurate scan.



Alex and his creators after rescuing the victims.

Section 8 References

1. *Dragon Runner Reconnaissance Robot - Army Technology*. (2010, March 1). Army Technology. https://www.army-technology.com/projects/dragonrunnerrobots/?cf-view
2. Explosive Protective Equipment (EPE). (2023, October 11). *Dragon Runner 20 (DR-20) | EPE*. EPE. https://www.epequip.com/catalogue/uncrewed-systems/uncrewed-ground-vehicles/dragon-runner-20-dr-20/
3. *Uncover Myriad Uses of Robotics across varied industries- DEEP Robotics*. (n.d.). https://www.deeprobotics.cn/en/index/industry.html#part2
4. DEEPRobotics Co.,Ltd. (2023). *Hazard Detection & Rescue Solution*. https://deep-website.oss-cn-hangzhou.aliyuncs.com/file/X20%20Hazard%20Detection%20%26%20Rescue%20Solution.pdf

Section 9 Appendix

9.1 Format of TPacket packet:

Byte number	**Type of data**	**Inputs**
0	Packet type	PACKET_TYPE_COMMAND PACKET_TYPE_RESPONSE PACKET_TYPE_ERROR PACKET_TYPE_MESSAGE PACKET_TYPE_HELLO

1	Command	COMMAND_FORWARD COMMAND_REVERSE COMMAND_TURN_LEFT COMMAND_TURN_RIGHT COMMAND_STOP COMMAND_GET_STATS COMMAND_CLEAR_STATS COMMAND_COLOUR COMMAND_DISPLAY COMMAND_DIST
2-3	Dummy bytes for padding	
4-35	Data	Only used in sendMessage() function, which is used by dbprintf() (mainly used for debugging), and write_message() function (used for displaying messages on LCD)
36-99	Parameters	Integer values for example the distance to move as well as the speed, or the number of ticks counted, are stored here.

9.2 Response Packets

Response	**Details**	**Packet check status**
RESP_OK	When Arduino receives either a move or query command successfully, return an ok packet	Success
RESP_STATUS	When Arduino receives a status command to get wheel encoder information, return the information via sending the status packet	Success
RESP_BAD_PACKET	When Arduino gets a corrupted packet with the wrong magic number, return this response packet	Failure
RESP_BAD_CHECKSUM	When Arduino gets a corrupted packet that fail checksum tests, return this packet	Failure
RESP_BAD_COMMAND	When Arduino gets a corrupted packet that passes the bad magic number and checksum tests but fails to recognize the command as a valid one, return this packet	Failure

RESP_COLOUR	When Arduino receives a get colour query command successfully, returns the colour packet, and the ok packet	Success
RESP_DIST	When Arduino receives a get distance query command successfully, returns the distance packet, and the ok packet	Success