CG2111A Engineering Principles and Practice II

Semester 2 2023/2024

"Alex to the Rescue"

Final Report

Team: B03-1B

Name	**Student No**	**Sub-Team**	**Role**
Goh Yee Ern	A0282618X	Software	Lidar Implementation
Loh Yan Xun, Timothy	A0272149B	Software	Colour Calibration & Music
Li Mengyan	A0281893N	Software	Robot Movement
Kankanam Gamage Himeth Thewmika	A0287216Y	Hardware	Circuit Design

Table of Contents

Section 1: Introduction

As urban populations continue to grow rapidly each year, the potential impact of catastrophes such as war, earthquakes, fires, and tsunamis becomes increasingly severe. Rescue robots are involved in rescue teams for their ability to navigate and explore hazardous and unfamiliar terrain, utilising their sensors to collect data. In view of this issue, Alex is a robotic vehicle designed with search and rescue functionalities. The operator remotely controls Alex using Secure Shell protocol (SSH) via laptop to access and control the Pi's terminal over the network. Alex's motion is navigated by the operator through the master control program (MCP) on the Pi to constantly communicate with Alex by sending commands. Alex is implemented with a Light Detection and Ranging (LIDAR) unit which runs clockwise to perform a 360-degree omnidirectional laser range scanning for its surrounding environment and then generate an outline map for the environment. This 2D outline map generated by the LIDAR will be relayed to the operator throughout the operation process. Alex is also implemented with a colour sensor and will be able to detect the objects of different colours together with the LIDAR. Alex will be able to differentiate "Victims" (either red or green) from other objects by processing the identification command given by the operator.

Section 2: Review of State of the Art

<u>Snake Robot</u>

The Snake Robot, known also as serpentine robots, is designed to mimic the locomotion of snakes. It typically possesses multiple actuated joints thus multiple degrees of freedom. This enables superior ability to flex, reach, and approach a huge volume in its workspace with an infinite number of configurations. The robot is equipped with various sensors and cameras mounted on its body segments to provide situational awareness.

The robot is flexible and can navigate through tight spaces and rubble piles. It can access confined areas such as collapsed buildings more easily than wheeled or tracked robots. However, it is limited in payload capacity and manipulation capabilities compared to larger robots. Moreover, it is susceptible to entanglement or getting stuck in debris and the complex control systems may require skilled operators.

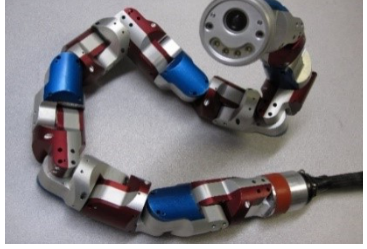


Fig 2.1 Snake Robot

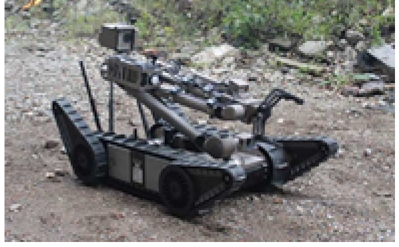


Figure 2.2 PackBot 510

PackBot 510

PackBot is a series of small, tracked robots developed by iRobot for various military and civilian applications, including search and rescue. It features a ruggedized chassis with tracks for mobility, along with a manipulator arm equipped with cameras, sensors, and tools for inspection and manipulation tasks. The PackBot 510 manipulator lifts to 20kg. It is deployable by one person in under two minutes.

The highly manoeuvrable design allows PackBot to traverse diverse terrains, including stairs, rubble, and rough terrain to relay real-time video, audio and sensor data while the operator stays at a safer, standoff distance. However, the endurance and battery life may be limited, necessitating frequent recharging or battery swaps during prolonged missions. Moreover, limited ground clearance may restrict mobility of the PackBot in extremely rough terrain or large debris piles.

Section 3: System Architecture

This section aims to explain the general architecture of Alex. The main components of Alex consist of 2 Microcontroller Units (MCUs), 1 colour sensor, 1 ultrasonic sensor, 1 speaker and 4 motors. We will explore the relationship between these components (Fig 3.1) and the way they cooperate with one another to accomplish the mission objectives.

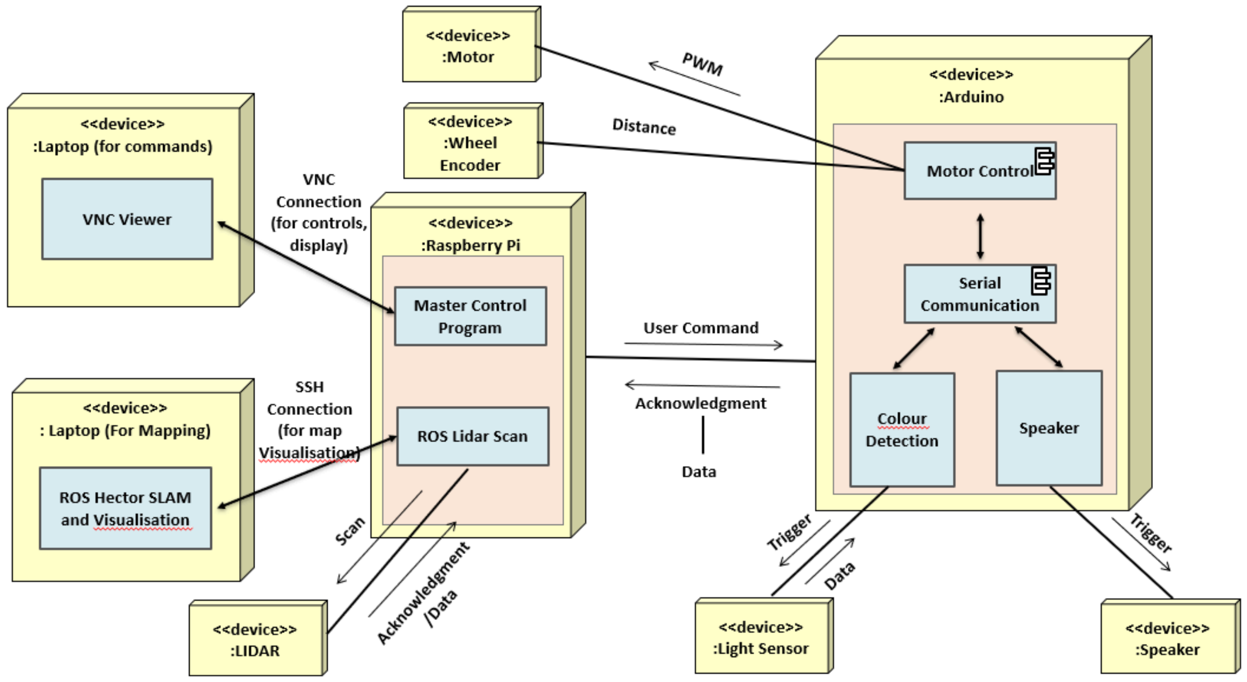


Fig 3.1 Overall System Architecture

Serial communication is established between the Arduino Mega and the Raspberry Pi (commonly known as Pi). The Pi acts as the central control unit, overseeing operation of Alex's system. The Arduino Mega, on the other hand, acts as the coordinating element, managing specific functions within Alex. The 2 Microcontroller Units work together to execute the search and locate functions effectively.

Raspberry Pi

The Pi is powered by a 3.7 power bank and is connected to the Arduino Mega and the Light Detection and Ranging (LIDAR) unit through UART (Universal Asynchronous Receiver / Transmitter). The Pi is accessed remotely by the operator's laptop using Secure Shell (SSH) Protocol and Virtual Network Computing (VNC) Protocol. The operator can access the Pi if the PC and the Pi are connected to the same network and will be able to send command packets to the Pi, thus supporting teleoperation.

The LIDAR sensor is directly connected to the Raspberry Pi, facilitating the transmission of live data for plotting and SLAM (Simultaneous Localisation and Mapping) algorithm processing. The sensor data is then published to a Robot Operating System (ROS) node, with the PC acting as the master. Subsequently, the data will be processed to create a 2D map. More information about the ROS network can be found in Section 6.

Arduino

The Arduino is connected to the motors, the colour sensor, the ultrasonic sensor, and the buzzer. Both the movement and the colour detection commands are integrated with the Master Control Program (MCP), the program which controls Alex. Commands from the operator are transmitted in the form of packets between the Pi and the Arduino.

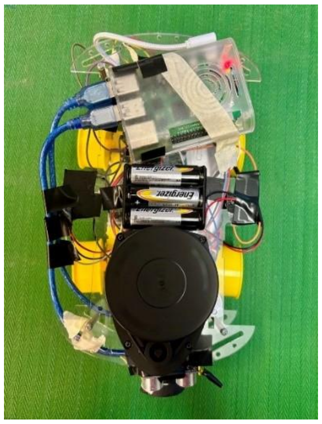
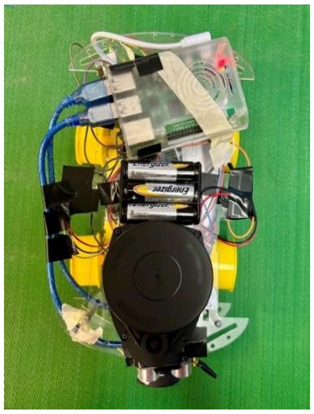
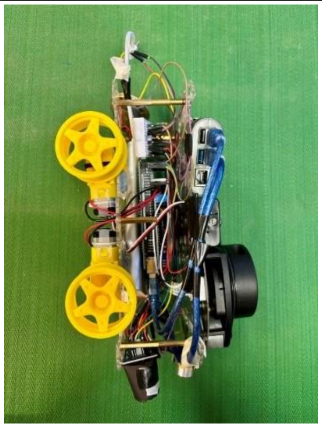
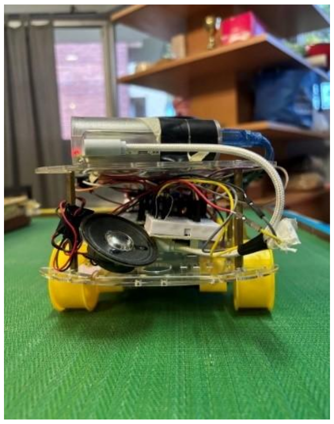
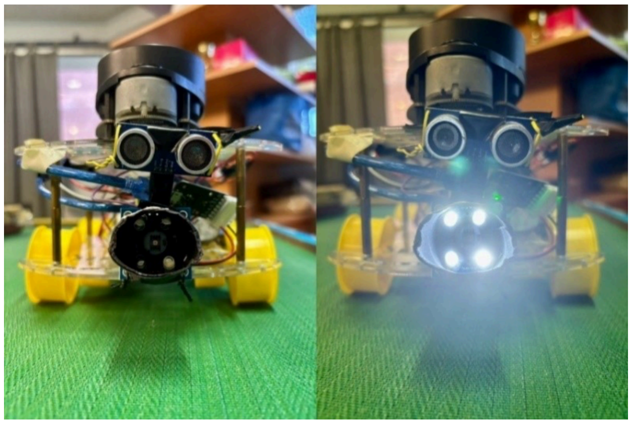
Colour detection is implemented using a Light Dependent Resistor (LDR) sensor connected to Arduino. Upon command, the LDR sensors perform colour detection of the obstacles in proximity before returning a message to Pi through Arduino, indicating the colour detected. After obtaining the colour, the corresponding music will play to signify a victim has been found.

Ultrasonic sensor measures the distance between front obstacles and Alex. It gives out the value of distance to the obstacle according to the command.

When the operator issues a movement command from the Pi, a command packet would be sent via serial communication to the Arduino which controls the motors. Once the Arduino finishes executing the command packets sent by Pi, it would send an acknowledgement packet to signal that it is ready for the next command.

Section 4: Hardware Design

This section describes the hardware design for Alex. The workings behind the respective hardware components and the reasons for incorporating them into Alex will be explained.

 *Fig 4.1 Top view of Alex*	 *Fig 4.2 Bottom view of Alex*
 *Fig 4.3 Side view of Alex*	 *Fig 4.4 Back view of Alex*
 *Fig 4.5 Front view of Alex*	

As shown in Fig. 4.1 to 4.5, we kept the shape of our Alex compact by ensuring most of the wires and components were kept within the confines of the chassis. This was an issue initially as the parts of Alex would affect the movement of Alex. We used a combination of zip ties, wires and tapes to ensure that all components were secured tightly and centralised within Alex.

We also increased Alex's height by using some screws to provide maximum space for including the Arduino, power bank, and RPi. This helps us evenly distribute the weight of Alex across the motors, ensuring smooth rotations. The table below tabulates the function and placement of each hardware component on Alex.

Name	**Function**	**Placement**
Arduino Mega	Receives instructions from the Raspberry Pi and controls motors, ultrasonic sensor, colour sensor, and buzzer when required.	Located on the lower deck of Alex, on top of the power bank, attached with tape.
Raspberry Pi	Sends information to the Arduino Mega for motor control, ultrasonic sensor, and colour sensor. It also transmits power to the LiDAR and Arduino Mega, and displays LiDAR map	Attached to the upper back of Alex using tape for easy access to plug in the power bank and HDMI cables.
LiDAR	Maps the surrounding environment and transmits data to the Raspberry Pi.	Secured to the upper front of Alex with 2 screws without blocking the sensor.
Power Bank	Provides power to the Raspberry Pi.	Placed on the lower deck of Alex to evenly distribute weight throughout.
2 Wheel Encoders	Measures the degrees of rotation and translates them into the distance travelled within the program.	Connected to the sides of the back wheels.
6 AA Batteries + Battery Holder	Provides power to the motors.	Top middle of the Alex secured with tape for easy accessibility when changing batteries
4 Motors	Creates a rotating motion that allows the wheels to spin, enabling movement.	Secured to the lower deck of Alex with screws.
4 Wheels	Allow for Alex’s movements.	Connected to the motors.
Colour Sensor	Provides colour sensing capabilities to allow Alex to detect the colour of the objects.	Connected to the lower deck and Alex's front, secured by wires and tapes.
Ultrasonic Sensor	Detect distance between Alex and the nearest front-facing obstacle.	Mounted to the upper deck and Alex's front, secured by wires and tapes.
Speaker	Provides a ringtone when detecting any of the victims	Connected to the back part of Alex.

Table 4.6 Function and Placement of Hardware Component

Section 5: Firmware Design

This section describes the firmware design of Alex, comprising 2 parts – the high-level algorithm on the Arduino Mega, and the communication protocol between the Arduino and the Pi.

<u>High-Level Algorithm on Arduino Mega</u>

Overall Program Flow

The Arduino receives commands sent from the Pi using a packet via serial communication, then checks for the packet's validity. If the packet is valid, the corresponding command is then carried out, else the corresponding error message is relayed back to the Pi. The overall process is shown in the flowchart below.

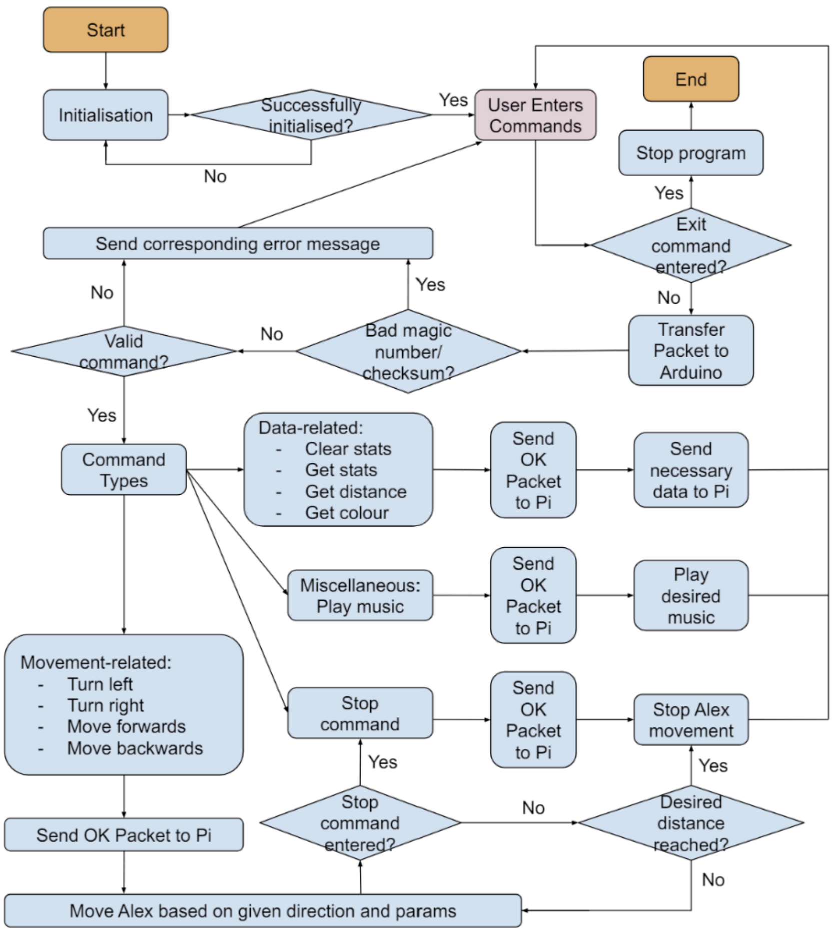


Fig. 5.1: Overall Program Flow

Outline of High-Level Algorithm

1. Initialisation
2. User Commands (Movement, Obtaining Data, Playing Music, Stopping)
3. Repeat step 2 until user enters exit command to end the program

Communication Protocol

Initialisation

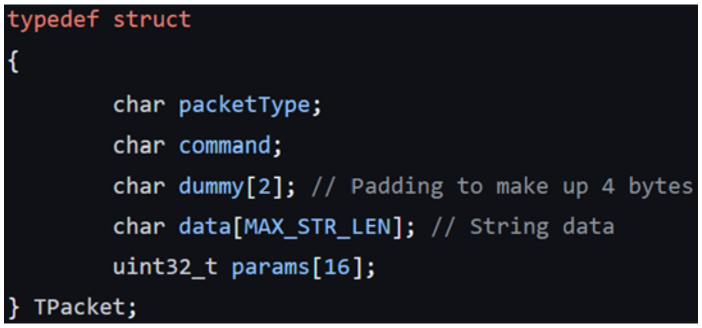
Upon booting up of the Pi, the other devices all get powered up, and the Arduino initialises all the necessary components, including the sensors, communication protocols, and outputs. Interrupts are also enabled.

To complete the initialisation process, the Pi does a handshake with the Arduino by sending it a "Hello" packet. The Arduino checks if the received "Hello" packet matches its own; if it matches, the Arduino sends an OK Packet back to the Pi, otherwise the Arduino returns an error message and the handshake process restarts.

Packet Transferring and Receiving

The communication is received by Arduino in a 8N1 data frame, with 1 start bit, 1 stop bit, no parity bits, and 8 data frames, at a Baud rate of 9600.

The packets have the following structure:



```
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;
```

Fig 5.2 Code for structure TPacket

"packetType" and "command" are characters describing the packet type and command respectively, as seen in constants.h; "dummy" then pads the above 2 bytes with 2 extra bytes to make up 4 bytes, ensuring the data transferred is not affected by Pi's word alignment. "params" is an array of 32-bit integers storing the parameters, differing for each command, such as direction and power for movement commands, and red, green and blue frequencies for colour commands. Finally, "data" contains the necessary string data to be sent to the Arduino.

After each packet is received, the Arduino checks for its validity via comparing its magic number, then doing a parity byte checksum. If either a bad magic number or bad checksum is received, the Arduino sends a bad packet or bad checksum error message respectively to the Pi to indicate that the data has been corrupted and the packet is invalid. If not, the packet is considered valid, and the Arduino executes the command received and sends an OK packet back to the Pi as acknowledgement.

Section 6: Software Design

Link to GitHub: https://github.com/yeeern27/CG2111A

Details of ROS Node Setup

The table below indicates the details regarding the setup of the ROS node between the Pi and the remote laptop. All the terminals are opened in Ubuntu 20.04 and source ~/cg2111a/devel/setup.bash has been run on all terminals.

First Terminal (PC)	Second Terminal (PI)	Third Terminal (PC)
Export ROS_MASTER_URI=http://PC_IP:11311 The command sets the URI (Uniform Resource Identifier) of the ROS Master Node. It notifies the ROS client the location and the port of the ROS master node.	Ssh pi@RPI_IP The command allows the user to access the Pi via SSH	roslaunch rplidar_ros view_slam.launch The command will launch Hector SLAM and RViz with the live laser scan data. This allows us to view the environment that Alex is in.
Export ROS_HOSTNAME=PC_IP The command sets the IP address of the PC in the ROS network. It tells the ROS client the IP address to use when communicating with other nodes in the ROS network.	Export ROS_MASTER_URI=http://PC_IP:11311 The command sets the URI (Uniform Resource Identifier) of the ROS Master Node. It notifies the ROS client the location and the port of the ROS master node.	
Roscore The command initialises the ROS master node. Roscore is run on the remote laptop as the laptop has higher computational power than the Pi. This allows smoother performance, enhancing the stability and reliability of the ROS network.	Export ROS_HOSTNAME=RPI_IP The command sets the IP address of the RPI in the ROS network. It tells the ROS client the IP address to use when communicating with other nodes in the ROS network.	
	roslaunch rplidar_ros rplidar.launch The command initialises the RPLidar node and publishes the scan data on the ROS network.	

Table 6.1 Details regarding ROS node setup between Pi and Remote Laptop

For teleoperation, modifications were made to the given code from the lab. The movement command is given by the user within the terminal of Pi, and the command will be sent to Arduino for execution through a high level algorithm. A “Command OK” command is sent to the Pi once the Arduino acknowledges the command. The table below tabulates the command that controls the movement of Alex.

Command Input	**Movement Done**	**Remarks**
‘f’	Move backwards	Desired distance and power for the motors to move are inputted after the command.
‘b’	Move forwards	
‘l’	Turn right	Desired angle and power for the motors to move are inputted after the command.
‘r’	Turn left	
‘s’	Stop the robot	

Table 6.2 Details regarding movement command

To determine the distance and the colour of the obstacle, an ultrasonic sensor and a colour sensor was implemented respectively. The following table tabulates the commands to retrieve the telemetry of the sensors, i.e. the ultrasonic sensor, the colour sensor and the hall encoder.

Command Input	Data Retrieved
‘g’	Wheel ticks that encoder recorded and the distance moved
‘z’	Distance of the obstacle in front of the ultrasonic sensor
‘a’	Colour of the obstacle
‘m’	Play music based on the colour detected
‘c’	Clear all stored information on movement

Table 6.3 Details regarding other commands

Details of Colour Sensor Detection

For the Pi to receive the distance and the colour read by the sensors, we created a handleDistance packet and a handleColour packet, as well as making edits to the constants.h file to include the new cases. The following table (table 6.4) outlines the process when a colour test command is sent from the Pi to the Arduino.

Command	Pi	Arduino
User types ‘a’ in the pi terminal	A colour command packet is sent from the Pi to the Arduino	Arduino unpacks the packet and verifies the magic number, followed by checksum. If it obtains a bad magic number, it will send a bad magic packet to the Pi. If the checksum of the packet received does not match the checksum of the packet sent, it will send a bad checksum packet to the Pi. When the Arduino receives the proper colour command, it will send the ‘Command OK’ packet to the Pi, and run the colour sensing function.
Arduino runs colour sensing function	Pi waits for the colour packet from the Arduino.	5 readings are taken for each for Red, Green and Blue frequencies and averaged out. This average value is fed into the determineColour() function to check against the calibrated value. If the frequency obtained matches one of the calibrated conditions, it will return the respective colour.
Arduino sends the colour obtained back to the Pi	Pi receives the colour of the object.	Arduino sends the colour to the Pi through the sendColour() function.

Table 6.4 Details regarding colour detection commands

Process of sending distance that is detected by the ultrasonic sensor is similar to the above table. Furthermore, a speaker is implemented on Alex. Music will be played after detecting the colour, and different music has been coded for the ‘red’ and ‘green’ victim. If a dummy is detected, no music will be played. A speaker instead of a buzzer is used as the speaker is louder than a buzzer, which can attract attention to notify people that a victim is detected.

Section 7: Limitations and Improvement

Lesson 1 – Time Management

Throughout the whole project, the most important lesson we have learnt is to adopt better time management methods. There were many unexpected bugs we encountered which greatly hindered our process as we spent too much time on debugging instead of moving on to settle down key features before we make any refinement. Moreover, we could have sought help from experienced individuals like our professor and TAs to reduce time wasted on repeated try and error. For example, we faced issues related to automatic port changing which took us a lot of time to figure out the root cause, but this was easily solved after we consulted our senior.

Lesson 2 – Strategic Planning

We could have improved our efficiency by avoiding repeated but meaningless testing with a better project management plan. This is especially true for our colour sensor calibration and robot movement testing. We failed to calibrate under the actual maze environment to get the closest set of readings at the start of implementation of the colour sensor which led to repeated colour calibration under different lighting conditions. This was like our movement testing, as we did not manage to simulate the closest smoothness of the maze floor for testing, we wasted time unnecessarily on determining whether to keep the wheel tyres or not for several rounds.

Mistake 1 - Distance Detection

We have faced difficulties in interpreting the SLAM mapping correctly as the distance was not to the correct scale shown on the map compared to the actual location of the robot. As shown in Figure 7.1, blue boxes indicate the corresponding location of Alex. It is significantly shown that the location of Alex indicated by the green arrow on the SLAM is inaccurate. To improve this, we firstly adjusted the shaft length and head length of the indicator to better simulate the location of Alex by taking the base of the arrowhead as a reference to the location of lidar. Moreover, we implemented an ultrasonic sensor at the front of the robot for distance detection to ensure consistency in colour detection as we made sure Alex was stopped within the most accurate range of detection from the object.

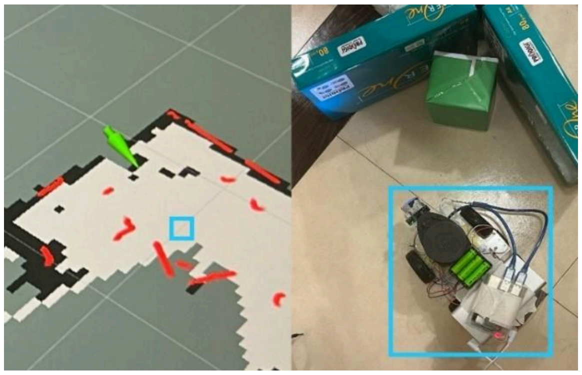


Fig 7.1: Comparing the mapped and actual location of Alex

<u>Mistake 2 – Weight Balance</u>

Another mistake we have made was failing to ensure even weight distribution to all four wheels. We initially placed the power bank at the back of the top layer of Alex (Fig 7.2), which was highly pressuring the back wheels, affecting the performance of the robot when carrying out turning movements. To tackle this, we tested and relocated the power bank to its optimal position which was the ground layer (Fig 7.2). This ensured that the weight of the power bank was evenly distributed to all four wheels and significantly improved the consistency and accuracy of the movement.

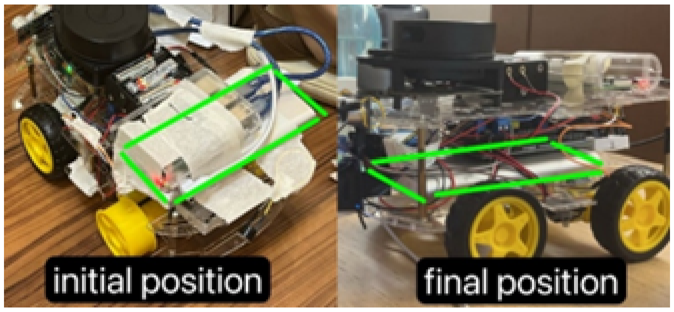


Fig 7.2: Change in position of power bank

References

“Rplidar A1M8 - 360 Degree Laser Scanner Development Kit.” *Farnell*, www.farnell.com/datasheets/3176118.pdf. Accessed 27 Mar. 2024.

“Modular Snake Robots.” *Modular Snake Robots - CMU Biorobotics*, biorobotics.ri.cmu.edu/projects/modsnake/. Accessed 29 Mar. 2024.

“PackBot® 510.” *PackBot® 510 | Teledyne FLIR*, www.flir.com/products/packbot/?vertical=ugs&segment=uis. Accessed 29 Mar. 2024.

(CMU), Carnegie Mellon University. “Snakebots.” *CMU*, www.cmu.edu/homepage/collaboration/2008/summer/snakebots.shtml. Accessed 29 Mar. 2024.

“Snake Robot Design.” *Robotics Institute Carnegie Mellon University*, www.ri.cmu.edu/project/snake-robot-design/#:~:text=Snake%20robots%20are%20a%20new,with%20infinte%20number%20of%20configurations. Accessed 29 Mar. 2024.

Norouzi, Freek De Bruijn, and Jaime Valls Mir´o, Mohammad. “(PDF) Planning Stable Paths for Urban Search and Rescue Robots.” *ResearchGate*, 2007, www.researchgate.net/publication/262363766_Planning_Stable_Paths_for_Urban_Search_and_Rescue_Robots. Accessed 29 Mar. 2024.

Chitikena, Hareesh, et al. “Robotics in Search and Rescue (SAR) Operations: An Ethical and Design Perspective Framework for Response Phase.” *MDPI*, Multidisciplinary Digital Publishing Institute, 30 Jan. 2023, www.mdpi.com/2076-3417/13/3/1800#:~:text=Robots%20are%20seen%20as%20part,centre%20to%20perform%20effective%20SAR. Accessed 08 Apr. 2024.