
**CG2111A Engineering Principle and Practice**

Semester 2 2024/2025

# “Alex to the Rescue”

# Final Report

# Team: B03-5B

| Name | Student Number | Main Role |
|---|---|---|
| Poh Wei Hao | A0299839X | Serial Comms, Motor/Servo Control, Interrupts |
| Juven Lim Heng Yi | A0273298N | Command Logic, Network Comms, Servo Control |
| Melvin Tan Jun Hao | A0306772B | Control Interface, SLAM & LIDAR Processing, Circuit Wiring |
| Russell Ng Jun Heng | A0299838Y | Colour Sensing, Control Interface, Serial Comms |

# Table of Contents

## 1. Introduction

In many real-world emergencies, human first responders are often faced with environments that are too hazardous or inaccessible. Delays in navigation or communication in such conditions can result in significant loss of life and impede rescue operations. In response to this challenge, “Alex” is designed as a teleoperated robot capable of executing critical search and rescue tasks while minimising risk to human operators.

Alex is designed to be compact, robust, and modular, capable of operating in a constrained environment such as mazes or simulated disaster zones. Alex’s physical form and movement system take inspiration from a scorpion, a creature known for its agility, precision and adaptability. The claw mechanism, mounted at the front, is used to rescue the red astronaut by physically gripping and dragging it to safety. Additionally, the rear-mounted servo arm mimics a scorpion’s tail, used to deliver the medpak to the green astronaut.

The robot integrates remote teleoperation, environment mapping, object manipulation, and colour-based victim identification. At its core, Alex comprises an Arduino-based control system, equipped with a Raspberry Pi for high-level coordination and data processing. Alex is also equipped with a LIDAR sensor that enables real-time 2D mapping through SLAM, allowing the remote operator to navigate and make decisions with spatial awareness. A wheel encoder system combined with interrupt-driven firmware ensures precise tracking of movement. The robot’s colour sensor guides the appropriate rescue response based on colour recognition.

Overall, Alex offers a practical and scalable solution for remote rescue missions in hazardous settings. It underscores the potential of biometric design in robotics, where nature-inspired forms enhance both function and versatility.

## 2. Review of State of the Art

**PackBot 510 by iRobot**

**System Description:**

The PackBot 510 is a tele-operated versatile, rugged, search and rescue robot designed mainly for military, law enforcement, and hazardous material handling operations.

**Key Functionalities:**

Explosive Ordnance Disposal, Reconnaissance and Surveillance, Hazardous material handling, Search and Rescue missions.

**Hardware Components:**

- Tracked mobility system with articulated flippers
- Modular design allows for quick attachment of various payloads
- Infrared cameras, pan-tilt-zoom cameras, and high-resolution colour cameras
- Manipulator arm with multiple degrees of freedom to handle objects
- Uses secure wireless communication for remote operation

**Software Components:**

- Remote Operator Control Unit with touchscreen or laptop interface
- Real-time video feed and telemetry data display

| |
|---|
| ● Sensor data integration software for chemical, radiological, and other hazardous material detection<br>● Navigation assistance through control algorithms and semi-autonomous functions |

| Strengths | Weaknesses |
|---|---|
| Rugged and reliable in extreme conditions | Relatively heavy, not much portability |
| Excellent maneuverability with flippers and tracks | Struggles in soft or loose terrains |
| Reduces human risk in dangerous environment | Expensive to purchase and maintain |

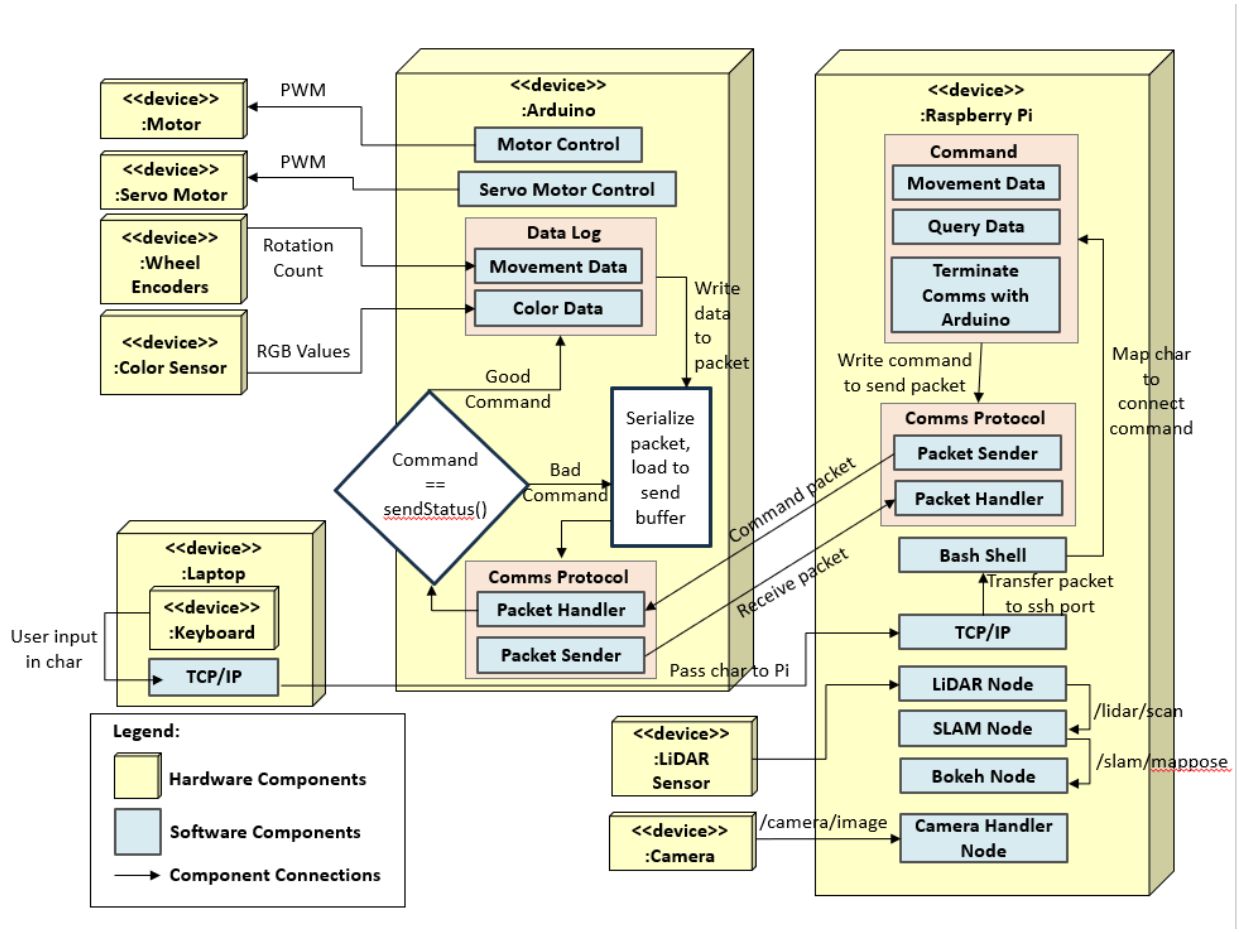
| **SuperDroid LT2 Tracked ATR** |
|---|
| **System Description:**<br>It is a compact, rugged, and fully customisable tracked robot platform designed mainly for military, law enforcement, and research institutions for remote inspection in hazardous or hard-to-access areas<br>**Key Functionalities:**<br>Remote surveillance and reconnaissance, hazardous area inspection, obstacle navigation, and payload customization.<br>**Hardware Components:**<br>● Tracked mobility system with robust treads<br>● Lightweight aluminium chassis with protective panels<br>● High-torque motors with regenerative braking and gearing<br>● Infrared cameras, thermal cameras<br>● Uses secure wireless communication for remote operation<br>**Software Components:**<br>● Remotely controlled system via a wireless controller or laptop<br>● Real-time video feed transmission<br>● Semi-autonomous navigation modules<br>● Navigation assistance through control algorithms and semi-autonomous functions |

| Strengths | Weaknesses |
|---|---|
| Excellent all-terrain capability | Not standardized and requires configuration |
| Easy for field repair and maintenance | Speed and agility depend on the configuration |
| Highly customizable platform | Price increases with custom features |

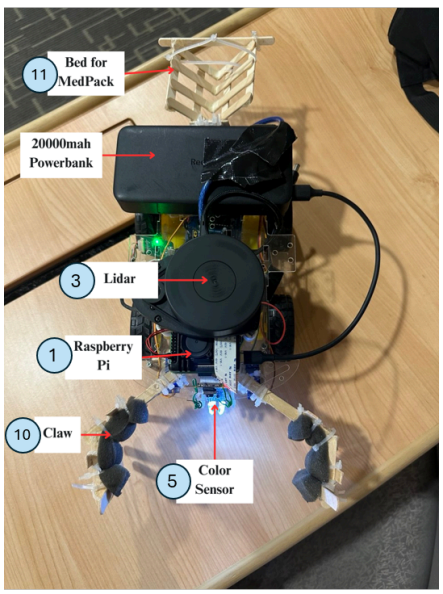
## 3. System Architecture

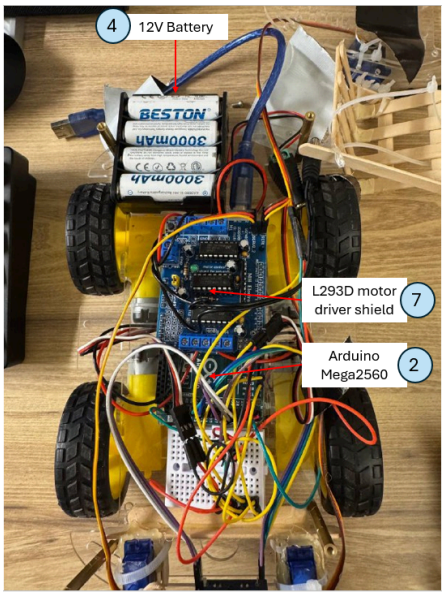
This section shows the various components in ALex and how each component communicates with each other. Alex comprises 9 devices, namely the motors, servo motor, wheel encoders, colour

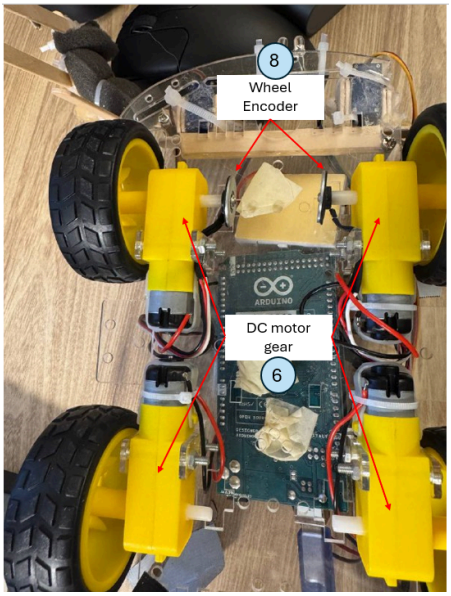
sensor, Arduino, Raspberry Pi, LiDAR, camera and a laptop. The figure below shows the system architecture of Alex showing all components of Alex and its connections.




## 4. Hardware Design



## 4.1 Hardware Components

| Component | Purpose | Considerations |
|---|---|---|
| Raspberry Pi | The Raspberry Pi is the main controller of Alex. It runs the main robot control software. It sends instructions to the Arduino from the client side via USART. It receives instructions from the client side wirelessly through wifi. It receives map data from the RPLidar and uses it to visually map out the surroundings on Bokeh using the SLAM algorithm. | With its cables connecting to the RPlidar and the Arduino. We taped the cable down to ensure the cables did not interfere with the Lidar. Place under the Lidar such that we have more space on top for other components as well as ensure that connecting the wires from the lidar to the Raspberry Pi is easier. |
| Arduino Mega2560 | The Arduino Mega2560 reads data from sensors (eg. Colour sensor and wheel encoders) and sends data to Raspberry Pi. It controls the movement of Alex like the DC Motors and the servos. | Wires connected to the Arduino were inserted through the holes of the body to ensure that the wires would not tangle with the motors and disconnect. |
| RPLidar A1M8 | Lidar continuously scans 360 around Alex in a 6m range and sends the data to the Raspberry Pi to map the surroundings using the SLAM algorithm. | We positioned the Lidar to be the highest point of Alex to minimise obstructions from other components. |
| 12V battery | Provides power to the Arduino Mega2560 and the Motors | Placed at the back of the Alex Robot |
| TCS3200 Colour Sensor | The colour sensor measures the intensity of reflected light in red, green, and blue (RGB) wavelengths. It then sends the intensity of the RGB light data to the Raspberry Pi to differentiate the colour of the object placed in front of it. | Ensured that the astronaut was grabbed first so that it was as close as possible to the colour sensor to ensure accurate reading and data |
| DC Motor | The motors are controlled by the Arduino Mega2560 and are configured to be able to move forward, backwards left and right with varying speeds and distance or angle. | Attached inwards such that wires can be attached more compactly. |
| L293D Motor Driver Shield | The L293D motor driver shield is used to control DC motors and servos from a microcontroller like the Arduino. It acts as a bridge between low-power control signals (from the Arduino) and the higher-power requirements of motors. | We had to download the <AFMotor.h> library dependency. |
| Wheels Encoder | The wheel encoders are attached to the front wheels of Alex to measure the distance and speed at which Alex is travelling. This is done by using | Needs to be as close as possible to the magnet and as much as possible almost parallel to the magnet. This is to ensure the hallmark sensor can |

| | | |
|---|---|---|
| | magnetized wheels and taking advantage of eddy currents which send a pulse to the Arduino after each rotation. These pulses are then sent to the Arduino to calculate the speed and distance. | correctly detect the change in magnetic field from the magnet when the wheel is rotating. |
| Heat sink | Used to dissipate heat from key components, mainly the CPU, GPU, and RAM, to prevent them from overheating. | Raspberry Pis don't have active cooling by default. So under heavy load, the temperature can rise quickly causing it to overheat and throttle performance |
| Claw | Attached to servos to grab Astronaut | Added foam cushion to ensure that the astronaut is held firmly in place as well as a stick wall to ensure the astronauts do not fall out as Alex moves around. |
| MedPak bed | Attached to a servo to hold and drop medpak | Must be placed in the mid-section of Alex so that when medpak is placed it will not block the lidar. |

**4.2 Additional Components**

| Component | Purpose | Considerations |
|---|---|---|
| 20000mah Powerbank | Reliable Power Source for the Raspberry PI | Must be able to deliver sufficient voltage to Raspberry PI to avoid low voltage warning. |
| Lidar Mount | Holder lidar in place and provide a flat bottom for easy mounting onto the body of Alex | There needs to be ample space before the lidar for Raspberry PI to fit so that we can have more space for a bigger battery at the back. |

## 5. Firmware Design

### 5.1. High-Level Algorithm on the Arduino Uno

The Arduino Uno acts as the firmware control unit for the robot, executing commands received from the RPi, collecting sensor data, and controlling motors and servos. The Arduino control flow is as such:

#### 5.1.1 Initialization

1. Set up serial communication with the RPi
2. Initialize hardware peripherals such as the motors, servos, TCS3200 colour sensor and wheel encoders
3. Configure interrupt routines for wheel encoder tick counting

### 5.1.2 Main Execution Loop

1. Listen for commands by continuously checking the serial port for incoming command packets
2. Deserialize incoming packets using a checksum-based validation process
3. Execute Command based on Type:
    a. **Movement:** Control DC motors with PWM signals and use encoder feedback to maintain precise distances and angles
    b. **Turning:** Rotate Alex using predefined tick counts for angles, tracked by encoder interrupts
    c. **Claw Control:** Open/close claw using servo motors for interacting with red astronauts
    d. **Tail Medpak Delivery:** Extend/retract servo mechanism to simulate medpak delivery to green astronauts
    e. **Colour Scan:** Activate TCS3200, cycle through RGB LEDs, and measure the light frequency reflected. Return the result via packet.
4. Respond with feedback by sending back telemetry data (e.g. tick counts, distances, detected colours) in TPacket response packets to the RPi.
5. Repeat and await the next command

Our interrupt routines ensure encoder ticks are tracked with minimal delay and without blocking the main loop, allowing for precise and real-time motion tracking.

## 5.2. Communication Protocol (Format of Messages and Responses)

Communication between the Arduino and RPi is established through a custom binary packet protocol, designed for efficiency, error detection, and extensibility. This protocol is implemented using structured TPacket data packets that encapsulate all necessary information for command execution, telemetry reporting, and error handling.

### 5.2.1. Packet Structure (TPacket)

1. **packetType (char):** Defines the type of packet being transmitted

| Packet Type | Description |
|---|---|
| PACKET_TYPE_COMMAND | Command issued by the Rpi to the Arduino |
| PACKET_TYPE_RESPONSE | Telemetry or acknowledgement from the Arduino |
| PACKET_TYPE_ERROR | Error Information from the Arduino |
| PACKET_TYPE_MESSAGE | General messages or debug information |
| PACKET_TYPE_HELLO | Initial handshake message |

2. **Command (char):** Enum representing actions

| Command | Description |
|---|---|
| COMMAND_FORWARD | Moves Alex forward. params[0] = distance(cm), params[1] = speed(0 to 100) |
| COMMAND_REVERSE | Moves Alex backwards. params[0] = distance(cm), params[1] = speed(0 to 100) |

| | |
|---|---|
| COMMAND_TURN_LEFT | Turns Alex left. params[0] = angle (degrees), params[1] = speed (0 to 100) |
| COMMAND_TURN_RIGHT | Turns Alex right. params[0] = angle (degrees), params[1] = speed (0 to 100) |
| COMMAND_STOP | Immediately stops Alex's movement. No additional parameters needed |
| COMMAND_GET_STATS | Requests telemetry (tick counts, distance travelled, turn info) from Arduino |
| COMMAND_CLEAR_STATS | Resets telemetry statistics on the Arduino |
| COMMAND_OPEN | Opens the front claw by setting left and right servo motor to open angle |
| COMMAND_CLOSE | Closes the front claw by setting the left and right servo motor to close angle |
| COMMAND_SCAN | Activates colour sensor to detect astronaut colour |
| COMMAND_DROP | Trigger the rear-mounted tail servo to turn 180 degrees and drop the medpak |

3. **Params[16] (uint32_t[16]):** Used to carry arguments like distance, speed, or raw RGB values
4. **Data[32] (char[32]):** String field for text-based messages or labels

### 5.2.2. Command Transmission and Response Workflow

The RPi initiates interactions by sending command packets over serial. Each command is accompanied by specific parameters that guide the Arduino's execution. This includes movement commands, actuation commands and sensor interaction. Once a command is processed, the Arduino sends a response packet back to the RPi containing either telemetry or confirmation data. The PACKET_TYPE_RESPONSE includes:

| Response Type | Description |
|---|---|
| RESP_OK | Confirmation of successful command execution |
| RESP_STATUS | Telemetry feedback (e.g. tick counts, distance, turn metrics) |
| RESP_COLOR | Colour detection result with raw RGB frequencies |
| RESP_BAD_PACKET | Error responses indicating malformed or unsupported requests |

The response ensures the RPi can verify correct execution, receive updated sensor data, or initiate fallback actions if necessary.

### 5.2.3. Communication Safeguards

To ensure reliable communication and accurate packet decoding, the following safeguards are implemented:

1. **Magic Number Verification**: Each packet contains a predefined magic number (0xFCFDFEFF) used to validate its origin and format.
2. **Checksum Validation**: A bitwise XOR checksum over the packet data ensures that transmission errors are detected before processing.

3. **Threaded Receiver on Pi**: A dedicated thread continuously listens for incoming packets without blocking the main control loop.
4. **Retry Logic**: Serial connection attempts are repeated multiple times at startup, with delays and diagnostics to ensure robust connection establishment.

### 5.3. Additional Noteworthy Firmware-Related Stuff

One of the key enhancements in the firmware design was the use of bare-metal Interrupt Service Routines to accurately track wheel encoder ticks during robot movement. This bare-metal approach enabled precise and real-time feedback on wheel rotation which is essential for distance calculation, turning accuracy and SLAM-based pose estimation.

| **Key Features of ISR Implementation** | **Description** |
|---|---|
| Direct Pin Interrupts | We configured hardware interrupts on digital pins connected to the encoder outputs |
| Efficient Tick Counting | Each tick increment is handled within an ISR with minimal overhead, ensuring ticks are not missed even at high speeds |
| Direction-Sensitive Ticking | The ISR accounts for movement direction, incrementing either forwards or reverse tick counter accordingly |
| Low Latency | Executing close to the hardware ensures minimal delay between signal changes and firmware response. |

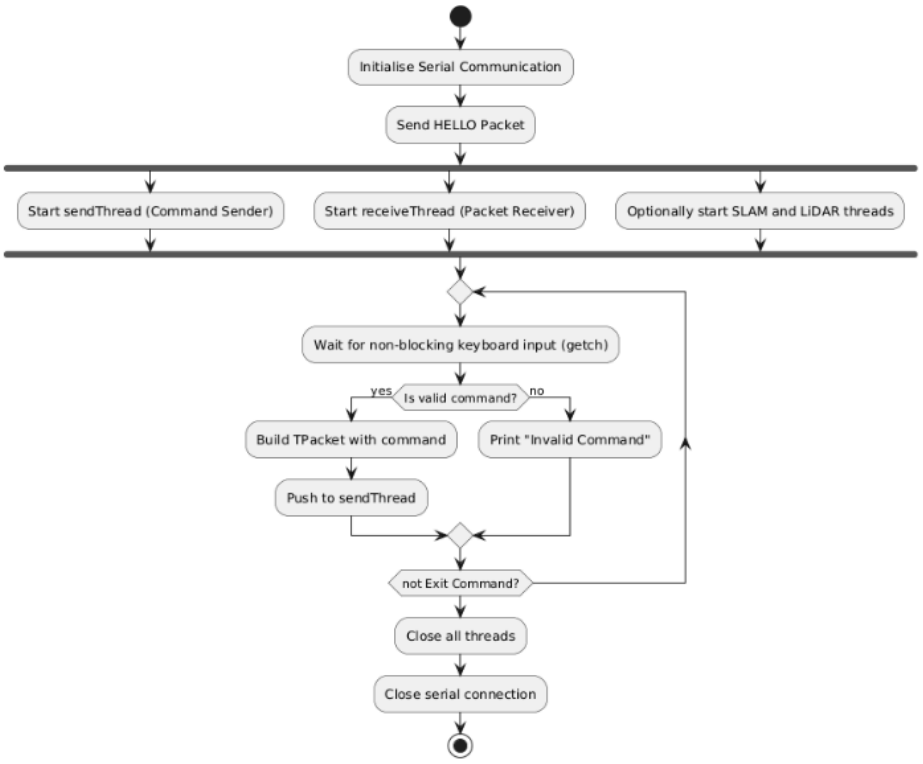
The advantages of bare-metal ISR over polling are key to achieving the fastest timing within our lab. With bare-metal ISR, we improved accuracy, guaranteeing no missed ticks, which would otherwise accumulate errors in movement estimation. Another advantage was increased responsiveness, where the immediate reaction to encoder changes allowed tight control in the dense maze and narrow paths. Moreover, CPU efficiency increases dramatically as ticks tracking from the main loop is offloaded, freeing the MCU to process commands or sensor data concurrently. This approach reflects an embedded systems best practice where performance, determinism, and low latency are prioritized in time-critical applications like mobile robotics.

## 6. Software Design

### 6.1. High-Level Algorithm on Raspberry Pi

The Raspberry Pi acts as the brain of the rescue robot, coordinating movement, sensing, and user interaction. It runs a multithreaded program that manages teleoperation, serial communication, colour detection, SLAM processing, and data visualization. All operator interaction is handled through a terminal interface, compatible with SSH and fully functional in headless setups.

The robot operates under a structured command system: movement, actuator control, and feedback queries. SLAM and LiDAR modules run continuously in the background, providing spatial awareness to the operator via a real-time web visualizer.




### 6.1.1. Teleoperation

The teleoperator controls the robot using three command groups: Movement Controls, Auxiliary Controls, and Query Controls. All commands are mapped to single characters and processed immediately using non-blocking input. Commands are encoded into structured packets, sent to the robot over serial, and executed in real-time.

**Movement Control**

Originally, the robot used manual input—users entered a command (e.g., f), and then provided a distance and speed. While precise, this method was inefficient for real-time control.

**Standard Movement Commands**

| Command | Description | Require Input |
|---|---|---|
| `f` | Move forward | Distance, Speed |
| `b` | Move backwards | Distance, Speed |
| `l` | Turn left | Angle, Speed |
| `r` | Turn right | Angle, Speed |

To streamline control, the system was updated to use **non-blocking, single-character input**. Pressing w, a, s, or d triggers a short nudge movement, while other keys initiate larger turns or translations. This method eliminates manual input and improves responsiveness. Details of the input method are discussed in Section 6.2.2.

**Improved Standard Movement Commands**

| Command | Description | Requires Input |
|---|---|---|
| **w** | Commands Alex to inch forward 3 cm | None |
| **a** | Commands Alex to turn anticlockwise 20 degrees | None |
| **s** | Commands Alex to inch backward 3 cm | None |
| **d** | Commands Alex to turn clockwise 20 degrees | None |
| **f** | Commands Alex to move forward 20 cm | None |
| **b** | Commands Alex to move backwards 20 cm | None |
| **q** | Commands Alex to turn anticlockwise 90 degrees | None |
| **e** | Commands Alex to turn clockwise 90 degrees | None |
| **1** | Commands Alex to turn anticlockwise 180 degrees | None |
| **2** | Commands Alex to turn clockwise 180 degrees | None |

**Auxiliary Controls**

The robot has two actuator systems: a claw and a medpak drop mechanism. These are controlled via instant single-key commands.

| Command | Description |
|---|---|
| o | Open the claw to release the astronaut |
| p | Close the claw to grab the astronaut |
| l | Tilt Medpak bed to drop Medpak |
| h | Stop all robot movement immediately |

**Query Commands**

Sensor and status queries are also triggered via single keys. These include colour scanning, retrieving movement stats and clearing stats. Results are parsed and printed in the terminal by the receiver thread.

| Command | Description |
|---|---|
| k | Sends a request to scan the current surface colour using the onboard RGB sensor. The Pi receives raw red, green, and blue frequency values and determines the closest matching colour. The result is printed alongside the RGB values in the terminal. |
| g | Requests movement statistics from the robot, including encoder tick counts for forward and reverse motion on each wheel, total distance travelled, and number of turns. |

| | |
|---|---|
| c | Clears the robot's internal movement statistics. |

### 6.1.2. Colour Detection

The robot detects surface colours using an onboard RGB sensor. When k is pressed, the Arduino reads red, green, and blue light intensities and sends them to the Pi. These values are shown in the terminal along with the colour label. The Pi uses a hardcoded reference method by calculating the Euclidean distance between the received RGB values and two predefined profiles. If the closest match is below a fixed threshold, it is accepted as the detected colour; otherwise, it reports "NO COLOR". This logic is mirrored on both Arduino and Pi for consistency. More on the method is explained in Section 6.2.3..

### 6.1.3. SLAM and LiDAR

SLAM and LiDAR modules run as separate Python threads or processes. The LiDAR continuously publishes range data, and the SLAM module uses it to update the robot's estimated position and generate a map. Both outputs are visualized using a Bokeh web server running on the Pi. Operators can access the live SLAM map through a browser on any device connected to the same local network. This setup provides real-time situational awareness without relying on a desktop interface or VNC. More details on the design choice and benefits of Bokeh are provided in Section 6.2.1.

## 6.2. Additional Noteworthy Software-Related Stuff

### 6.2.1. Bokeh

Bokeh is used as a lightweight, browser-accessible tool to display SLAM and LiDAR scan data in real-time. Running as a local server on the Raspberry Pi, it allows users to monitor the robot's location and environment from any device on the same network without needing a GUI or VNC. It streams data to the browser using WebSockets, providing smooth, non-blocking visual feedback. This separation of computation and rendering reduces CPU load on the Pi and helps maintain responsive control during operation.

### 6.2.2. Non-blocking, No-buffered Single Character Input

The improved teleoperation system uses a low-level `getch()` implementation to capture single-character inputs immediately. This function disables input buffering and echo, allowing the program to read a key as soon as it's pressed—without requiring the Enter key. This creates a smoother and more responsive control experience, especially for movement commands like `W`, `A`, `S`, and `D`, which mimic game-style navigation.

To avoid command flooding or buffer conflicts, a short delay (usleep(500000)) is added after each input. This balances responsiveness with stability, preventing packet overlap or serial congestion. The delay is tunable and provides a smooth, safe control experience.

### 6.2.3. Colour Classification Logic

Colour classification is done using a simple nearest-reference method. The Pi compares incoming RGB values to two fixed references using Euclidean distance. The closest match is selected if within a predefined threshold. This approach mimics a 1-nearest neighbour classifier but is lightweight and ideal for embedded systems. It requires no training, executes quickly, and fits the

mission's binary classification needs. Consistent logic is used on both Arduino and Pi, supporting accurate, explainable results in a resource-constrained environment.

## 7. Conclusion and Lessons Learnt

### 7.1 Mistakes Made

#### 7.1.1 Serial Packet Desynchronization (“Bad Magic Number”)

One recurring issue we faced was the occurrence of “bad magic number” errors in serial communication between the Rpi and Arduino. This typically occurred when input commands were sent in rapid succession, overwhelming the Arduino’s ability to process packets in time. These incomplete or malformed packets led to failed deserialization on the receiving end. To resolve this, we introduced a short delay usleep(500000) after each command to allow the Arduino sufficient time to process and respond, greatly reducing packet corruption and improving the system stability.

#### 7.1.2 Claw Design Limitations

Another mistake was constructing the robot’s claw opening to be too narrow, providing only a small margin for error when attempting to grip the astronaut. We also tested dragging the astronaut by simulating the astronauts with rolled-up paper or empty cans which failed to account for the actual mass and rigidity of the provided astronaut figures. Hence, when we eventually tested with a heavier prototype astronaut, the claw could not withstand the mechanical stress and ultimately broke due to insufficient structural support.

### 7.2 Lessons Learnt

#### 7.2.1 Importance of Repeated Testing

Through this project, we recognised the importance of repeated testing of our robot to familiarise ourselves with the controls of the robot for efficiency and quickly remedy any errors that we might face during the actual run. By repeatedly attempting the course with a different layout, we were able to judge based on the lidar scan map how close the object needed to be on screen to Alex for us to scan and/or grab the astronaut accurately. As a result, we were able to complete the maze in about 3 minutes and 30 seconds making us the fastest group in our lab. This gives us the insight that iterative testing is not only essential for debugging but also for developing familiarity with robot control and environmental dynamics.

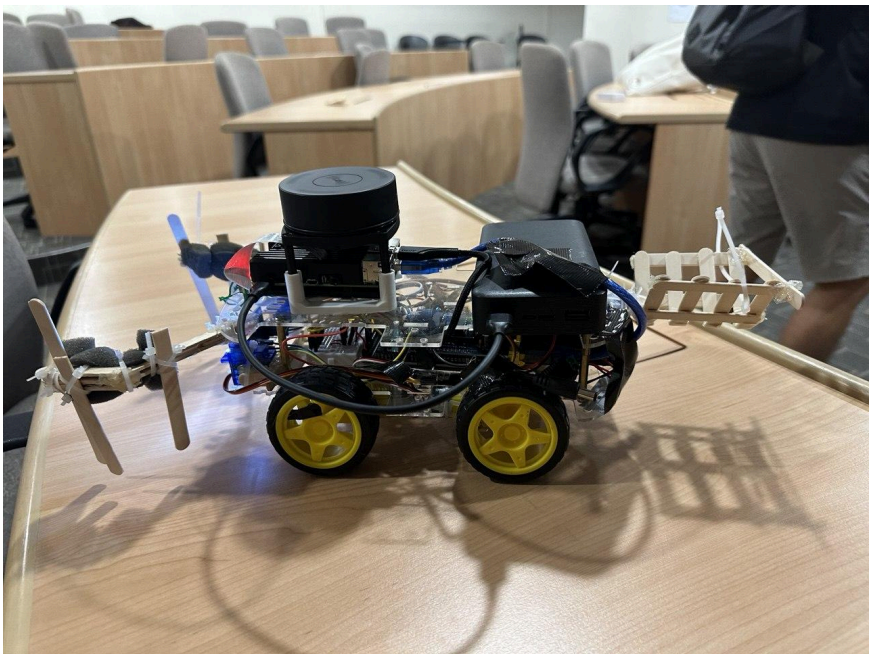
#### 7.2.2 Applying K-Nearest Neighbour for Reliable Colour Detection

Initially, our colour-sensing module produced incorrect readings nearly 40% of the time due to raw threshold-based classification. Upon learning and implementing the K-Nearest Neighbour (KNN) algorithm for RGB classification, we significantly improved detection accuracy. This method compared measured RGB values against reference profiles for red and green astronauts, resulting in near-perfect classification rates. Integrating KNN not only enhanced technical reliability but also boosted operator confidence, allowing us to make navigation and rescue decisions based purely on sensor feedback rather than relying too much on external camera confirmation.

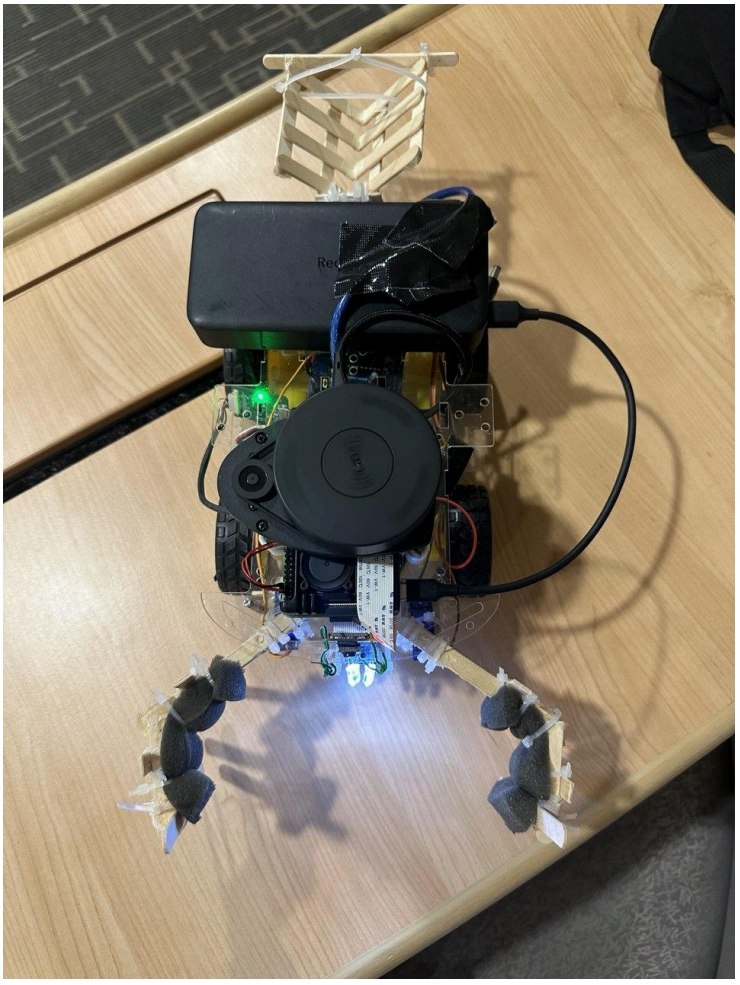
**Acknowledgements:** We would like to thank Prof. Colin, TAs and lab personnel for the guidance.

## 8. Appendix

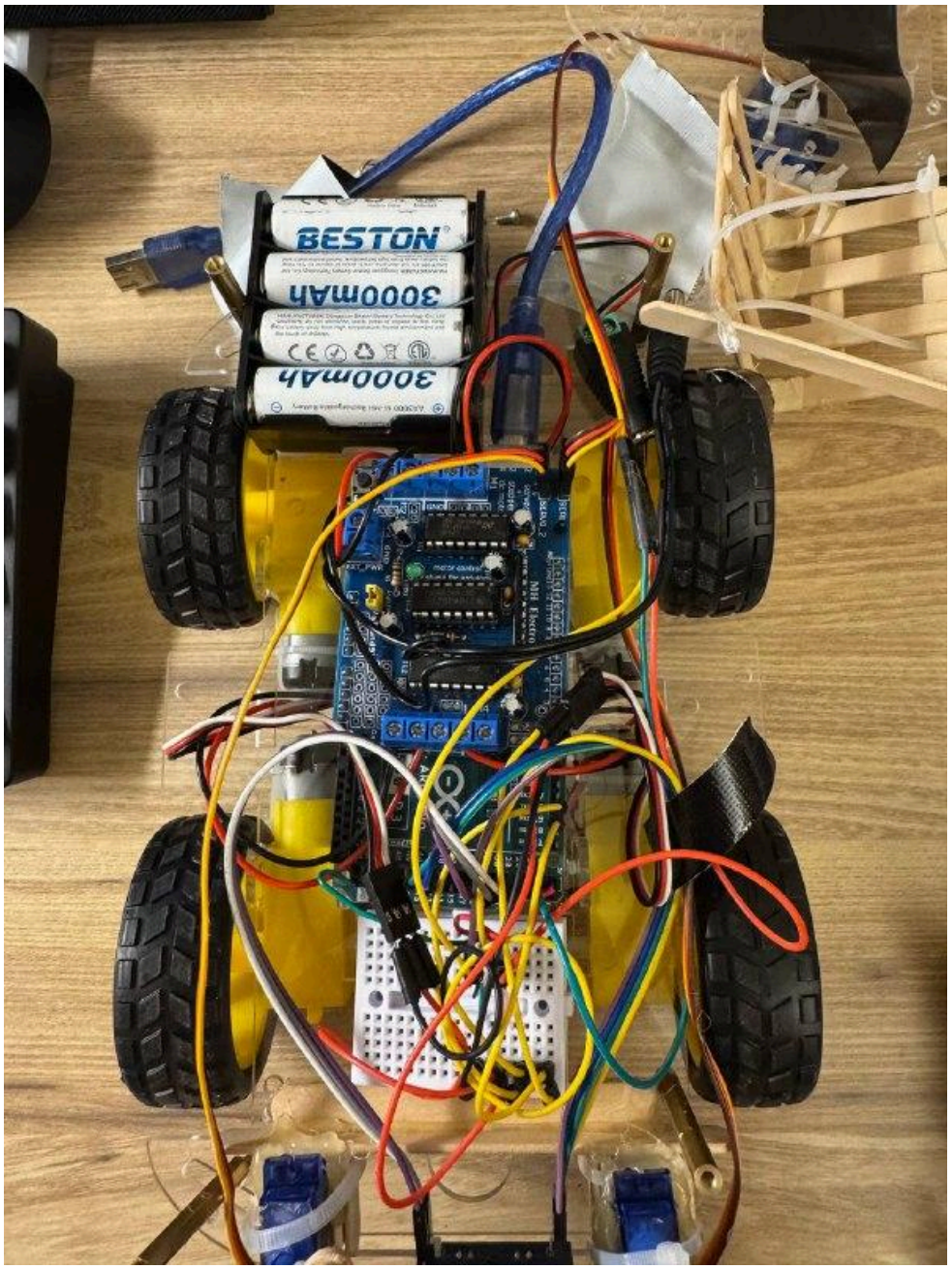
### 8.1. Side View of Alex



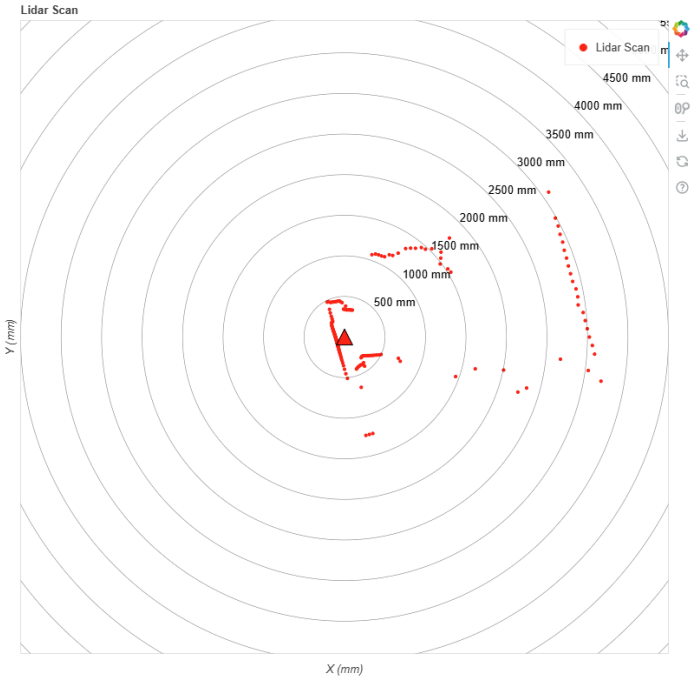
### 8.2. Top View of Alex

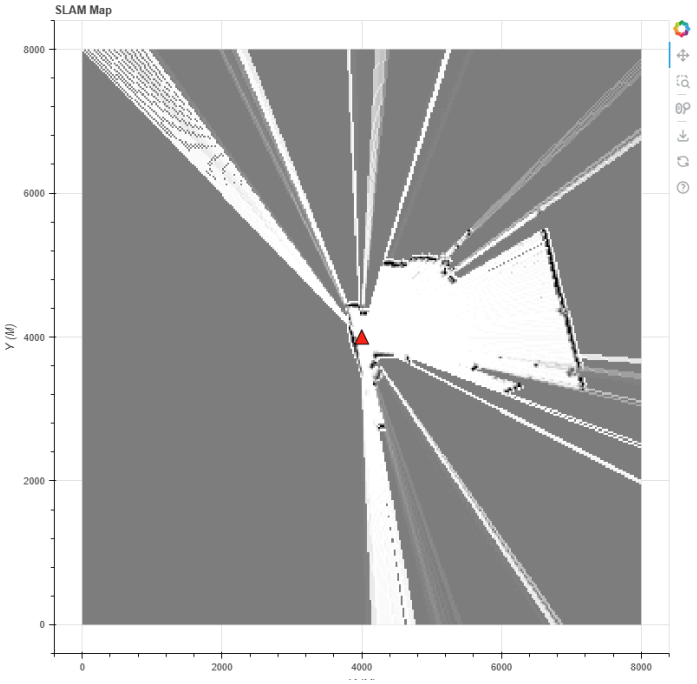


## 8.3. Interior View of Alex



## 8.4. Bokeh Interface



## 8.5. Bare-Metal Implementation

```
// Set up the external interrupt pins INT2 and INT3
// for falling edge triggered. Use bare-metal.
void setupEINT()
{
  // Bare-metal configuration of external interrupts

  // Configure INT2 and INT3 for falling edge detection
  // Bits 5:4 (ISC2[1:0]) for INT2: 10 = falling edge generates interrupt
  // Bits 7:6 (ISC3[1:0]) for INT3: 10 = falling edge generates interrupt
  EICRA = (1 << ISC31) | (0 << ISC30) | (1 << ISC21) | (0 << ISC20);
  // Alternative representation: EICRA = 0b10100000;

  // Enable INT2 and INT3 interrupts
  // Bit 2 (INT2): External Interrupt Request 2 Enable
  // Bit 3 (INT3): External Interrupt Request 3 Enable
  EIMSK = (1 << INT3) | (1 << INT2);
  // Alternative representation: EIMSK = 0b00001100;
}


// Define the actual interrupt vector handlers at the raw memory address level
void __vector_3(void) __attribute__((signal, used, externally_visible));
void __vector_4(void) __attribute__((signal, used, externally_visible));

// INT3 ISR calls leftISR while INT2 ISR
// calls rightISR.
void __vector_3(void) // INT2 vector - vector number 3
{
  rightISR(); // Call our handler function
}

void __vector_4(void) // INT3 vector - vector number 4
{
  leftISR(); // Call our handler function
}
```

## 8.6. KNN Algorithm Implementation

```
int getAvgFreq() {
  int reading;
  int total = 0;
  for (int i = 0; i < 5; i++) {
    reading = pulseIn(sensorOut, LOW);
    total += reading;
    delay(20);
  }
  return total / 5;
}

void sendColor() {
  TPacket colorPacket;
  colorPacket.packetType = PACKET_TYPE_RESPONSE;
  colorPacket.command = RESP_COLOR;

  colorPacket.params[0] = redFreq;
  colorPacket.params[1] = greenFreq;
  colorPacket.params[2] = blueFreq;

  sendResponse(&colorPacket);
}
```

```
struct ColorSample {
  const char* label;
  int r, g, b;
};

//Red and Green average from 20 readings
ColorSample colorDB[] = {
  {"RED", 71, 235, 179},
  {"GREEN", 59, 51, 54}
};
```

```
#define NUM_SAMPLES 2

const char* classifyColor(int r, int g, int b) {
  float minDist = 1e9;
  const char* nearest = "NO COLOR";

  for (int i = 0; i < NUM_SAMPLES; i++) {
    float dist = sqrt(pow(r - colorDB[i].r, 2) + pow(g - colorDB[i].g, 2) + pow
    (b - colorDB[i].b, 2));
    if (dist < minDist) {
      minDist = dist;
      nearest = colorDB[i].label;
    }
  }

  // Optional threshold to reduce false positives
  if (minDist > 100.0)
    return "NO COLOR";

  return nearest;
}
```

```
void scanColor()
{
  digitalWrite(S0, HIGH);
  digitalWrite(S1, LOW);

  digitalWrite(S2, LOW);
  digitalWrite(S3, LOW);
  delay(100);
  redFreq = getAvgFreq();
  delay(100);

  digitalWrite(S2, HIGH); digitalWrite(S3, HIGH);
  delay(100);
  greenFreq = getAvgFreq();
  delay(100);

  digitalWrite(S2, LOW); digitalWrite(S3, HIGH);
  delay(100);
  blueFreq = getAvgFreq();
  delay(100);

  const char* colorLabel = classifyColor(redFreq, greenFreq, blueFreq);
  dbprintf("Detected Color: %s", (char*) colorLabel);
}
```

```c
void handleColor(TPacket *packet) {
    uint32_t red = packet->params[0];
    uint32_t green = packet->params[1];
    uint32_t blue = packet->params[2];

    printf("\n --------- ALEX COLOR SENSOR --------- \n\n");
    printf("Red (R) frequency:\t%d\n", red);
    printf("Green (G) frequency:\t%d\n", green);
    printf("Blue (B) frequency:\t%d\n", blue);

    // Re-run the same k-NN logic here to mirror Arduino's result
  float distRed = sqrt((float)((red - 71) * (red - 71) +
                               (green - 235) * (green - 235) +
                               (blue - 179) * (blue - 179)));

  float distGreen = sqrt((float)((red - 59) * (red - 59) +
                                 (green - 51) * (green - 51) +
                                 (blue - 54) * (blue - 54)));


    const char* color = "NO COLOR";
    if (distRed < distGreen && distRed < 100)
        color = "RED";
    else if (distGreen < distRed && distGreen < 100)
        color = "GREEN";

    printf("Detected Color: %s\n", color);
}
```

```
--------- ALEX COLOR SENSOR ---------

Red (R) frequency:      72
Green (G) frequency:    184
Blue (B) frequency:     143
Detected Color: RED
```

```
--------- ALEX COLOR SENSOR ---------

Red (R) frequency:      52
Green (G) frequency:    46
Blue (B) frequency:     49
Detected Color: GREEN
```

```
Command (w=forward, s=reverse, a=turn left, d=turn right, h=stop, c=clear stats, o=open, p=close, l=med, k=scan, g=get s
tats q=exit)
Command OK
Command (w=forward, s=reverse, a=turn left, d=turn right, h=stop, c=clear stats, o=open, p=close, l=med, k=scan, g=get s
tats q=exit)
Message from Alex: Detected Color: NO COLOR

 --------- ALEX COLOR SENSOR ---------

Red (R) frequency:      287
Green (G) frequency:    320
Blue (B) frequency:     271
Detected Color: NO COLOR
```

## 8.7. Non-blocking, No-buffered Single Character Input

```
int main()
{
    // Connect to the Arduino
    startSerial(PORT_NAME, BAUD_RATE, 8, 'N', 1, 5);

    // Sleep for two seconds
    printf("WAITING TWO SECONDS FOR ARDUINO TO REBOOT\n");
    sleep(2);
    printf("DONE\n");

    // Spawn receiver thread
    pthread_t recv;

    pthread_create(&recv, NULL, receiveThread, NULL);

    // Send a hello packet
    TPacket helloPacket;

    helloPacket.packetType = PACKET_TYPE_HELLO;
    sendPacket(&helloPacket);

    while(!exitFlag)
    {
        char ch;
        printf("Command (w=forward, s=reverse, a=turn left, d=turn right, h=stop, c=clear stats, o=open, p=close, l=med, k=scan, g=get stats q=exit)\n");
        //scanf("%c", &ch);

        // Purge extraneous characters from input stream
        //flushInput();

        ch = getch();
        sendCommand(ch);
        usleep(500000); //delay 500ms
    }

    printf("Closing connection to Arduino.\n");
    endSerial();
}
```

## 9. References

**Teledyne FLIR.** (n.d.). *PackBot® 510*. Teledyne FLIR. Retrieved March 26, 2025, from https://www.flir.com/products/packbot/?vertical=ugs&segment=uis

**SuperDroid Robots.** (n.d.). *LT2 tracked all-terrain robotic platform*. SuperDroid Robots. Retrieved March 26, 2025, from https://www.superdroidrobots.com/store/robotic-kits-platforms/tracked-robots/product=1513