CG2111A Engineering Principle and Practice

Semester 2 2022/2023

"Alex to the Rescue"

Final Report

Team: B01-5B

Name	Student #	Main Role
Chooi Hao Wen	A0254467W	Report Editor and Software Debugger
Claribel Ho Jia Huan	A0257431H	Report Lead and Colour Sensor
Dylan Chia	A0253141U	Procurement and Hardware Lead
Tong Zheng Hong	A0251819Y	Software Lead

Table of Contents

Section 1 - Introduction

Natural disasters, such as earthquakes, hurricanes, and floods, are unpredictable and unavoidable occurrences that can cause immense damage and loss of life. Despite efforts to predict and prepare for such events, the scale and intensity of these disasters can overwhelm human response capabilities. As such, search and rescue efforts become a critical priority, and the use of robotic platforms can play a vital role in improving the speed and accuracy of search and rescues, as well as saving lives without risking human lives by replacing human rescuers with robotic ones. One such use was in the wake of the earthquake that struck Mexico City in 2017[1], search and rescue robots (e.g. snake robots) were used to locate and extract survivors from the rubble. These robots were able to navigate through the unstable debris and detect the presence of human life, which helped guide the rescue efforts and save lives.

Alex is set to achieve these functions. With a RPLidar sensor mounted on its top deck to map out its surroundings, the operator can remotely control Alex to navigate the disaster terrain with reference to the map relayed. Alex also has in-built voice communications via walkie-talkie to communicate with on-site emergency rescuers, further enhancing its search and rescue capabilities. In the simulation, Alex is tasked to navigate the maze within 8 minutes, to correctly identify 2 victims while minimizing collisions with the walls.

Section 2 - Review of State of the Art

We researched and summarized 2 tele-operating search and rescue robotic platforms below:

2.1 Boston Dynamics' Spot

Spot is a quadruped robot created by Boston Dynamics[2], which can remotely search and scan hazardous environments or disaster areas. He travels tough terrain with his flexible legs while collecting 2D and 3D information with on board-sensors to do inspection, surveying, and mapping. He is controlled from afar using a tablet application and built-in stereo cameras and has a 360° perception to map terrain and avoid obstacles as they appear.

Spot [3] has numerous hardware and software components (Fig. 1), including a sophisticated sensor system and a modular payload system to which additional sensors and tools may be attached. The software is configured to execute different tasks, from basic navigation to complicated item handling and decision-making. Spot API, a software component, allows applications to control Spot, read sensor information etc. It uses a client-server model, where communication with services running on Spot is over a network connection.

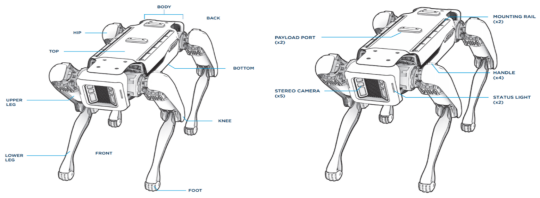


Figure 1: External Specifications of Spot

Strengths: Spot's quadruped design allows it to navigate through difficult terrain and perform a wide range of tasks. The robot is highly customizable, with a modular payload system that allows for easy attachment of various sensors and tools. Thus, it can adapt to a wide range of tasks and environments.

Weaknesses: Spot's relatively small size limits its payload capacity, which may limit its ability to perform certain tasks.

2.2 Snakebot

Snakebot [4], developed by Carnegie Mellon University, is a search and rescue robot that is designed to navigate through difficult environments to locate and rescue victims. It is made up of interconnected segments that can bend and twist in different directions, allowing it to move through tight spaces and over obstacles. It has sensors, such as cameras and microphones, to detect and locate victims, as well as environmental hazards that may impede the rescue operation. A human operator controls the Snakebot's movements and receives feedback from its sensors via teleoperation.

Strengths: The Snakebot is extremely flexible, able to bend and twist. This allows it to move through tight spaces (e.g. debris) to reach places that humans or other robots cannot reach. Its modular design also allows for easy repair and replacement of individual segments if they become damaged during the operation.

Weaknesses: It is unable to move quickly or cover large distances. It also requires a sufficient power source to function and a stable and reliable communication link with the controller, which are likely hard to maintain in disaster zones.

Section 3 - System Architecture

Alex involves 9 devices, namely the Arduino Uno, Raspberry Pi, LIDAR, Motors, Wheel Encoders, Keyboard, Color Sensor, Ultrasonic Sensor and the laptop of the teleoperator. Fig. 2 below shows the system architecture of Alex that shows all components of Alex and its connections. We also installed a Walkie Talkie externally that functions independently and is thus excluded from the diagram.

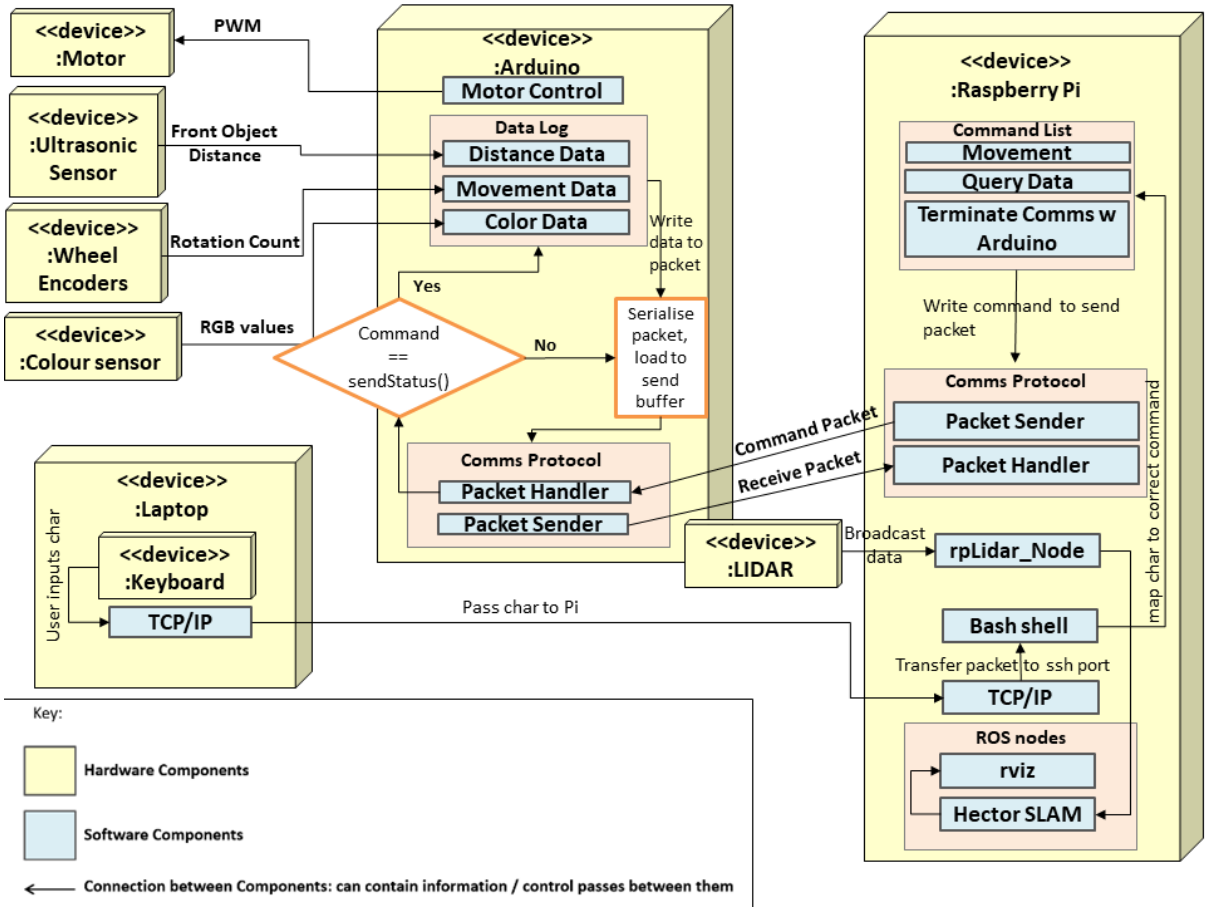


Figure 2: System Architecture Design

Section 4 - Hardware Design

4.1 Design Considerations for Alex

Fig 3 and 4 below shows the hardware components layout on Alex from its Top, Front, Side and Bottom views.

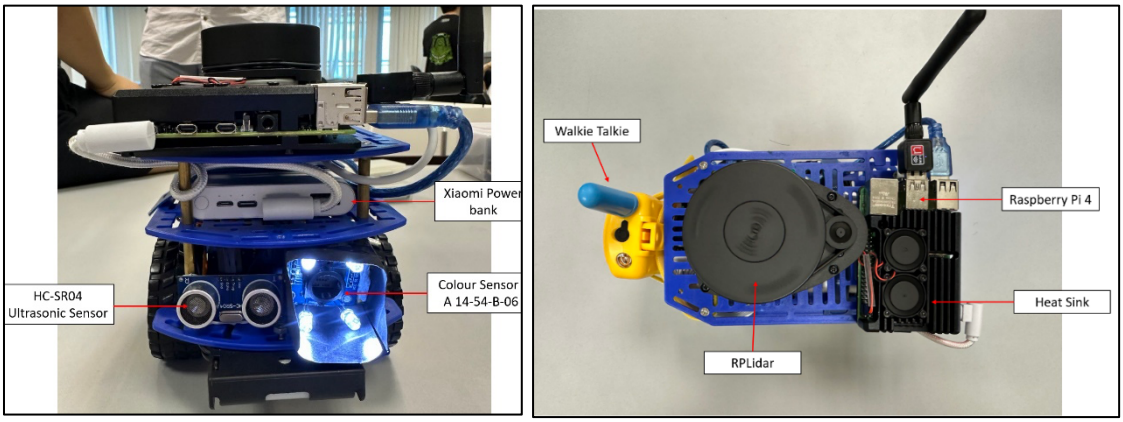


Figure 3: Front (left) and Top (right) View of Alex

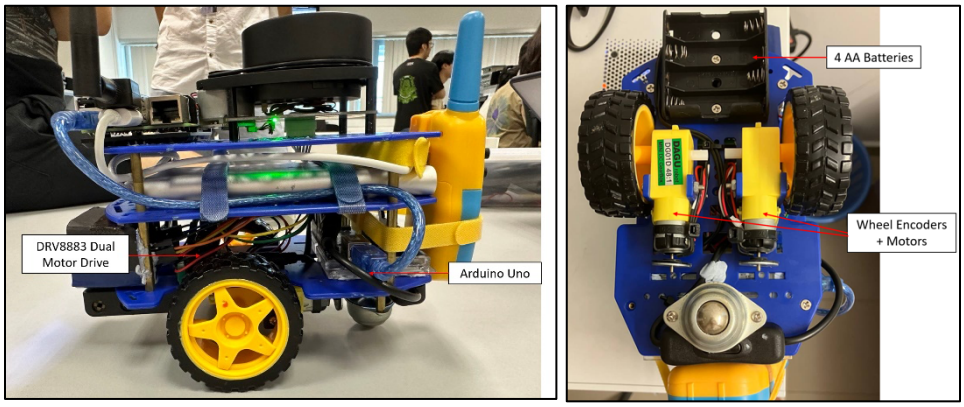


Figure 4: Side (left) and Bottom (right) View of Alex

We identified three main concerns when designing Alex, namely space management, centre of gravity and drifting.

Space Management

This includes the layout of the components and cable management. For example, the LIDAR has to be mounted at the top facing the front for it to scan the environment without obstruction, colour sensors and ultrasonic sensors are mounted at the bottom facing the front to scan the dummy. We ensured that we minimised the components protruding out from the Magician Chassis frame by using cable ties and blue-tack, to prevent any unnecessary collisions with the walls of the maze. Also, at the lowest compartment, the wires are given enough space to bend without them breaking internally.

Centre of Gravity (CG)

The heavier components are placed closer to the ground to lower the CG, which ensures that Alex does not topple over while navigating the hump. Laterally, the components are aligned to the center as much as possible so that the CG lies approximately in the middle.

Drifting

Our Alex still drifts to the right side when both motors are set to the same speed. This could be attributed to 2 reasons.

- Firstly, the centre of gravity could be offset from the centre, lying closer to the right which causes the robot drifting towards the right as there is a net rotational torque towards the right.
- Secondly, the right motor is actually slower than the left motor due to manufacturing inaccuracies.

We set both motors to 100% speed due to the weight of Alex which made movement difficult and less predictable at lower speeds. This weight issue will be discussed in later sections. To account for right drifting tendencies, we lowered the speed of the left motor slightly to around 98%. With this setup, Alex mostly moves straight but drifts occasionally likely due to uneven friction between the floor and each of the 3 wheels Alex has.

4.2 Hardware Functionalities

In this section, we will describe the functionalities of some of the hardware used in the project. These hardware components work together to provide the core functionality, which is to navigate Alex without collision.

4.2.1 HC-SR04 Ultrasonic Sensor

The sensor is mounted at the front of Alex to measure the distance between Alex and the obstacle ahead of it. This helps us to determine the working range of the color sensor as well as determine when Alex is too close to an obstacle, as a buffer distance must be kept between Alex and any surrounding obstacle to allow Alex to turn without colliding.

Additionally, in tight spaces such as the parking space, the LIDAR sensor becomes very inaccurate. Ultrasonic sensor becomes crucial when navigating in such spaces as it provides a secondary source of reliable information to obstacles in front which might not show up accurately on the LIDAR scan.

4.2.2 Eyein UPRO-CHD32-2C Walkie Talkie [5]

We chose our additional functionality to be a microphone and speaker system via the use of two Walkie Talkies, one attached to the Alex and one for the operator. In the real-world context, when Alex detects a victim, the survivor can use the on-built microphone to communicate with emergency personnel. This system aids in transmission of instructions or knowledge of environmental conditions, where both parties can hear and speak clearly to one another remotely using the Walkie Talkie. This thus assists on-site rescuers to locate the trapped victims using details provided by the victim, on top of the navigation map. The Walkie Talkie is mounted to the back of Alex using Velcro cable ties and operates using 4 integrated AAA batteries.

4.2.3 A 14-54-B-06 Color Sensor

The color sensor is mounted at the front of Alex to identify whether an object is a victim (Red/Green) or a dummy. We then identify the color of the target by measuring the RGB values of the reflected light. This is done when the user sends the command to identify color.

Since the reflected light readings are affected by ambient lighting, we shielded the 4 white LEDs and the sensor with black paper to block out ambient light from the sensor to improve its accuracy. During the testing phase, we calibrated the color values to ensure reliability of the readings to determine the working range of the color sensor so that we know the optimal distance between Alex and the dummy to take accurate color readings. The software implementation of the color sensor and the ultrasonic sensor with the Arduino will be discussed in Section 6.1.2.

4.3 Additional Hardware Improvements

4.3.1 Aluminum heatsink casing with dual fan

The idle temperature of the RPi hovers around 50°C. The temperature would easily reach 70~80°C when we were running RViz, Hector SLAM and the Master Control Program simultaneously to operate Alex. At that temperature range, the RPi became less responsive, and it made it difficult to send commands to Alex and navigate it through the room.

To address this issue, we mounted a dedicated aluminum heat sink casing on the RPi to physically reduce its core temperature. This simple and effective approach was able to dissipate heat over a larger surface area. The additional fans that came with the casing further aided in heat dissipation by directing cooler air into the CPU core.

With the casing mounted, the maximum temperature of the RPi is 45°C when running all the necessary programs for operation, which is much lower than the original idle temperature without the casing. By maintaining a significantly lower operating temperature, we were able to drastically improve its stability and performance, allowing for easier and more agile control of Alex. However, one drawback of this approach was the additional weight of the aluminum casing, which we will discuss later in the mistakes made.

4.3.2 Soldered compact PCB Breadboard

One issue our team ran into was cable management, especially when it came to the cables being routed through the center of Alex. Not only were the connections to the original solderless breadboard confusing and messy, but they were often not secure and could become dislodged and loose when the robot navigates rough terrains such as the hump in the room.

To reinforce for this weakness, we implemented a customized soldered PCB Breadboard with a very compact circuit design as well as female and male header pins to interface all connections that the original solderless breadboard makes. This allows for a much cleaner and more organized cable solution that was more secure and consistent. Refer to Appendix 9.5 for a before and after comparison of the wiring of components on Alex.

Section 5 - Firmware Design

5.1 High Level Algorithm on Arduino Uno

While the serial communication between Arduino and RPi is active, the Arduino repeatedly polls for incoming data packet and performs the checks as follows:

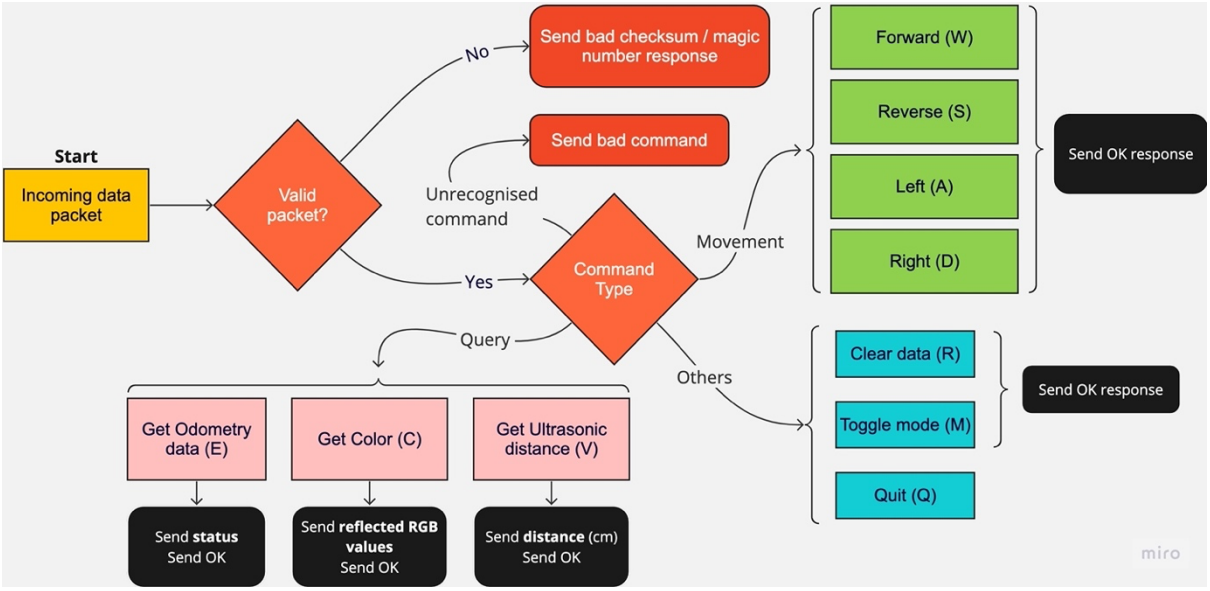


Figure 5: High Level Algorithm on Arduino Uno

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. After accounting for extra bytes used for error checking like appending magic number, data size and checksum plus the padding used to preempt the Pi's compiler to the data packet when serializing, each packet is 140 bytes long. This means that the time taken for RPi to send a command to Arduino and vice versa is (140*10) / 9600 = 0.15s or 150ms.

This places a limitation on the rate at which the operator can send commands to the Arduino to operate Alex. Since it takes at least 150ms for transmission of one packet between the Arduino and RPi, if the operator sends a packet before the previous packet has been completely received by the Arduino, it could alter the sequence the frames are received in or result in dropped frames. This limitation will be further discussed in the conclusion.

Another consideration is that the data packet does not arrive at the receiver simultaneously as one complete packet. Instead, the data arrives in fragments of bytes. Consequently, there is a need to wait for all 140 bytes to be received before the received information can be deserialized into a valid packet. Therefore, 150ms is the minimum time taken to send a complete packet, since the calculation excludes the time taken for all packet fragments to arrive and get assembled.

When the Arduino completes executing a command, it sends a response packet to the Pi. Refer to Appendix 9.1, which describes the different possible packets sent to Pi.

5.3 Bare Metal Programming for Motors

With reference to the code in Appendix 9.3, the wheel encoders enable pull up resistors on pins 2 (connected to motor of left wheel) and 3 (connected to motor of right wheel), as well as set bits 2 and 3 in DDRD to 0 to make them inputs.

With reference to the code in Appendix 9.3, to control the motors for the wheels, we used a pre-scaler value of 64 to generate a 490Hz PWM frequency, given that the frequency of the Arduino Uno is 16MHz. Subsequently, the movement of Alex will be controlled by enabling and/or disabling the signal on OC0A, OC0B, OC1B and OC2A pins.

Section 6 - Software Design

6.1 High Level Algorithm on Pi

1. Initialize communication between Pi, Arduino and PC.
2. Generating Map using Hector SLAM
3. Teleoperator issues command based on data
4. Arduino processes command
5. Repeat steps 2-4 until Alex reaches the parking spot

6.1.1 Teleoperation

The overarching teleoperation design objective is for the controls to be simple and intuitive because the given template code uses controls: f (forward), b (reverse), l (left) and r (right) which are very unintuitive to operate Alex. We changed the mapping of the movement controls to the standard WASD keys, which is a popular key binding for video games. The added intuitive nature reduces the chance of keying the wrong command. This, together with the features mentioned in Section 6.2, makes it more user-friendly for the teleoperator.

During execution, the terminal waits for the specific characters from keyboard input which are bound to different commands. There are a total of 13 commands (font color in **dark red**), and these commands are subdivided into the following groups based on their functionalities.

Movement commands:

Commands (WASD)			
Forward	Reverse	Left	Right
W	**A**	**S**	**D**

Auto mode				Manual mode
Mode	Park	Normal	Hump	- Requires direction, distance or turn duration as input - Allow operator to select specific distance or turn duration for navigation - Used for back up mode if the preset values in auto mode are insufficient for Alex to navigate the room
Command	**1**	**2**	**3**	
Distance / Turn duration presets	~2 cm / 0.05s	~5 cm / 0.1s	~25 cm / 0.25s	
Description	- For fine controls especially for parking in 1x1 tile. - Used to inch closer to target without hitting them inadvertently	- Used for normal operations to explore room at a sufficiently slow pace for LIDAR to scan surroundings without distortions in SLAM map image	- Used to traverse hump with longer travel distance. - Likely to distort map image which would require a restart of RViz	

As shown above, there are 2 main modes, auto and manual mode. By default, upon starting the master control program, Alex is configured to be in auto mode. In auto mode, the distance for forward and reverse, and angle for left and right turns are preset. This pre-determined value can be varied by changing between 3 modes (bound to 1,2,3 keys): Park, Normal and Hump modes, each with increasing values of distance and turn durations (as shown in the table above).

We changed the turn command to be based on turn duration instead of angle. If the motor stalls, the wheel will not be turning and thus it is impossible to reach the desired turn angle,

causing the program to freeze. As such, we changed the turn to be based on turn duration so that Alex will always stop turning even if its motor stalls. This also simplifies the software implementation as it eliminates the need to calculate the number of encoder turns based on Alex's dimensions.

In manual mode, the operator can achieve finer movement control by selecting the exact distance or turn duration value to be applied in the selected direction. However, this mode requires the operator to hit enter for each command which makes the controls cumbersome and unintuitive. Therefore, this mode is designed for backup purposes when the preset values are insufficient to navigate the room. Enter 'm' as command to toggle between the 2 modes.

Query commands:

Command	Key	Description
Get Color Data	**C**	- Returns information from the color sensor to RPi via a response packet, prints the information on the RPi terminal. - This information includes the RGB values, as well as the ultrasonic distance and it is stored in the Arduino using global variables. The ultrasonic distance is sent together with the RGB values to allow the teleoperator to determine the optimal working range of the color sensor during the testing phase. - The RPi prints the received RGB values and uses them to compute the color of the dummy according to section 6.1.2, together with the ultrasonic distance.
Get Ultrasonic Data	**V**	- Returns information from the ultrasonic sensor to RPi via a response packet, prints the information on the RPi terminal. - This information includes the distance between Alex and the object in front of it. The object can be the wall or the dummy. - During operation, ultrasonic distance will be sent to the RPi after each movement command to notify the operator of any obstacles in front of Alex. The operator can use this command for further verification.
Get Odometry Data	**E**	- Returns information from the wheel encoder to RPi via a response packet, prints the information on the RPi terminal. - This information includes the number of turns made by each wheel, as well as the distance travelled by each wheel, and it is stored in the Arduino using global variables.

Other commands:

Command	Key	Description
Data Clearing	**R**	All global variables storing odometry data will be reset to 0.
End serial comms	**Q**	RPi will stop reading inputs from the keyboard and end serial communications with Arduino
Toggle mode	**M**	Toggle between auto and manual modes

6.1.2 Color Detection and Calibration

While configuring the color sensor, we noted a difficulty in differentiating between red and orange colors, and between green and blue. Thus, we pre-determined three values, RED_THRESHOLD = 1100 (absolute color value), GREEN_THRESHOLD = 25 (in percent) and COLOR_THRESHOLD = 10 (in percent). These values were determined at the beginning by comparing differences in R, G & B reflected light values of each colored surface placed at a fixed distance away from Alex.

The primary distinction whether a target is **red** or **green** is by comparing the reflected R and G values from the color sensor. The figure below shows the flowchart for determining the color of the target. Refer to Appendix 9.2 for the code implementation of the algorithm.

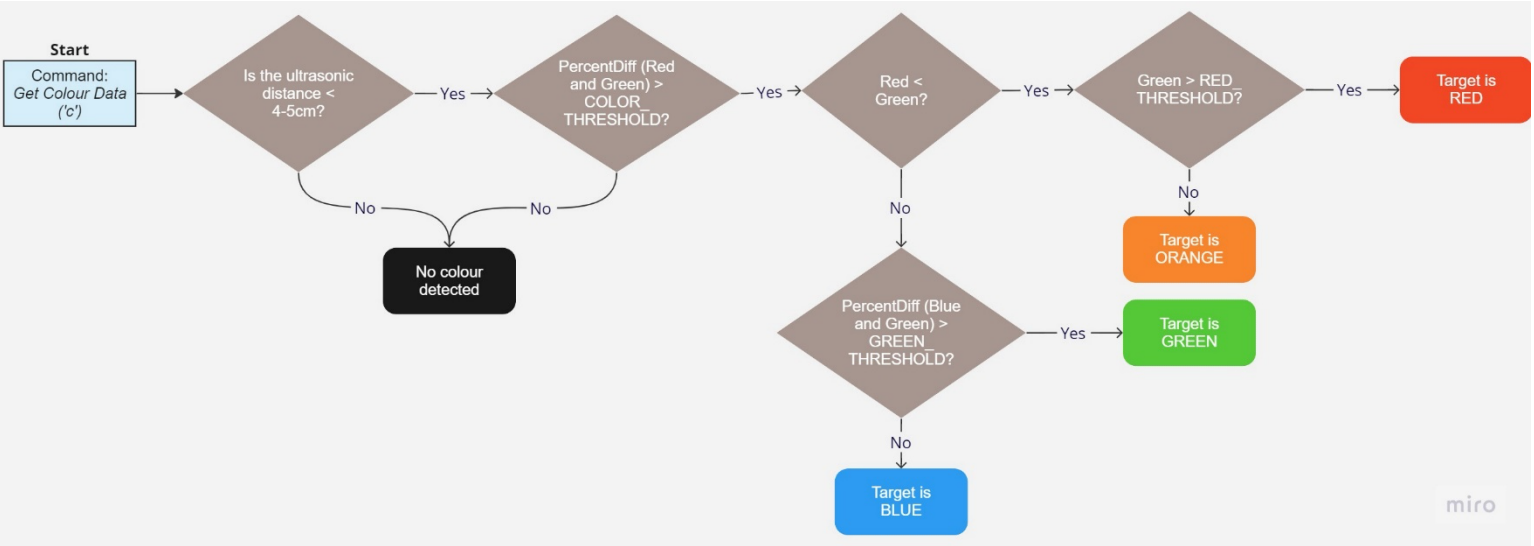


Figure 6: Color Detection Algorithm

One observation when tuning the color sensor was that the reflected light values for each surface color varies based on the distance the sensor was away from the surface. Hence, color calibration and identification were **always** done when Alex was at a predetermined distance of around 4-5 cm away from a surface with the help of the ultrasonic sensor for the best consistency and accuracy.

6.2 Additional Software Features

Auto mode using getch()

Auto mode is implemented using getch() instead of scanf() since getch() reads directly from standard input without waiting for the enter key whereas scanf() requires the enter key to execute. Also, getch() does not write to stdin buffer, whereas scanf() does without flushing it after executing the command.

This means that the teleoperator only needs to press a single key for a single command. This makes it more user-friendly and intuitive for the user to press WASD to move without pressing enter after each move. Also, the programmer does not have to worry about flushing inputs when implementing move commands in AUTO mode, while having a slight improvement in runtime. (since getch() does not wait for the newline character, it performs one less comparison than say getchar(), although, the time delay from this is overshadowed by the time delay from the user forgetting to press enter after each command)

Additionally, since getch() is called from the conio.h header file, which is not supported by macOS, getch() was implemented by our software lead using termios.h. This makes the function cross platform and so our program can compile on all operating systems as opposed to importing getch() from the conio.h which is only supported by MS-DOS compilers. Refer to Appendix 9.4 for the code snippet.

Using Transform Frame (TF) to model Alex in RViz

The studio on lidar programming only showed us how to display the pose of the lidar as an arrow on the map in RViz. However, this arrow is quite inaccurate at displaying the position of Alex on the generated map. Hence, we implemented additional markers that are positioned offset relative to the lidar's position using TF. Each marker by default is modeled with 3 axis which can be rotated to form the corners of Alex. As such, 4 markers were used to outline the boundaries of Alex on the RViz map.

In order to achieve this, we had to add the code the following code snippet into the view_slam.launch file and add the TF element within RViz. Next, we also needed to display the correct markers within the TF settings to achieve the follow effect:

```
<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="0.2 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="0.2 0 0 1.57 0 0 bottomLeft close" />
```

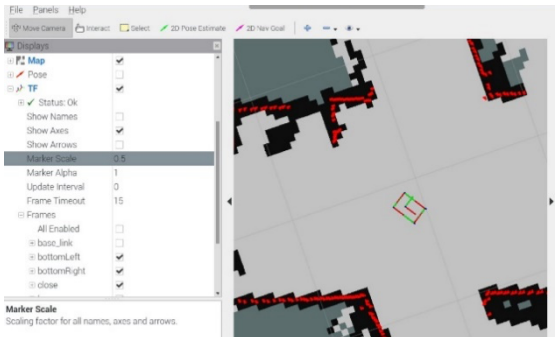


Figure 7: Transform Frame markers settings

Section 7 - Conclusion

7.1 Mistakes Made

7.1.1 Bad Magic Number

During the trial run, we experienced Bad Magic Number error. As mentioned in Section 5.2, the theoretical transmission time for each packet between the Arduino and RPi is at least 150ms. If a packet fragment arrives at the receive buffer when it is full, the incoming packet fragment is dropped silently at the time of buffer overflow. The next time the buffer is ready to receive another packet fragment, it will be very likely be different from the dropped fragment. This means that when the fragments are assembled, the received packet will become corrupted, meaning that there will be a bad checksum error returned when attempting to deserialize the bit stream. Subsequent packets following the previous packet will return the Bad Magic Number error because of the dropped packet fragment.

The above scenario can happen when the teleoperator presses the keys in a manner such that the time between each key press is less than the transmission time. For the trial run, we tapped the keys about 4 times a second, which triggered the Bad Magic Number, which makes sense given the 150ms transmission time is close to the 250ms tapping frequency of the teleoperator.

To address the issue, we need to ensure that a command has been received by the Arduino before the next command can be sent. This can be achieved by maintaining a 'sending' flag that will be set to true when a command is sent. It will only be set to false when the RPi receives an OK packet, indicating that the Arduino has received the command successfully. While the flag is true, the RPi ignores any incoming commands from the operator as the previous command is still being sent to the Arduino. With this check in place, we can continuously hold any key without bad magic number error as the program helps ensure that each command is sent successfully before soliciting and sending the next command.

7.1.2 Failure to plan the design of the robot before assembling

Secondly, we were inattentive to the initial assembly of the hardware components, leading to unnecessary hassle subsequently (refer to section 4.3.2). Towards the end of the project, the wires were disorganized and jutted out unnecessarily. As a result, we had to disassemble the entire robot to reconfigure the wiring between the motors, color sensor, ultrasonic sensor, and Arduino Uno, all located at the lowest deck of Alex. This could have been avoided should we have been more cautious from the start, by soldering connections and using header pins for the repeated 5V and GND port connections to reduce use of wires.

7.2 Lessons Learnt

7.2.1 Weight consideration

The most important lesson we learnt was the significance of Alex's weight on its performance. The addition of the aluminum RPi casing and Walkie Talkie resulted in a considerable increase in the weight of Alex. As a result, our motors had to operate at near 100% speed for optimal performance. During the runs, we experienced limited mobility towards the end of 8 minutes as the batteries were limited on charge capacity to support the added weight of our Alex.

In the future we will solve this by using 1.5V Alkaline batteries that have higher voltage compared to the 1.33V NiMH Eneloop batteries. This is so that more power can be supplied to the motors to operate our Alex which was heavier than normal over a longer period. This also highlighted the need to test each change comprehensively as we would not have known this issue without attempting the run with the additional weight.

7.2.2 Effective team collaboration using software

We also learnt how to use project management software to coordinate and distribute work effectively as a team. For example, developing code can be done as a team using version control systems like Git which allows the entire team to keep up to date on the latest changes made. We also learnt how to review code written by someone else, vice versa, as well as debugging them. We also learnt how to use Microsoft Teams to work together on group reports, as well as using plugins like Zotero for easy citations, as well as installing code formatters to generate the coding font, which helps to improve productivity as well as report quality.

Section 8 - References

[1] ‘After Disaster Strikes, a Robot Might Save Your Life’, *Discover Magazine*. https://www.discovermagazine.com/technology/after-disaster-strikes-a-robot-might-save-your-life (accessed Apr. 19, 2023).

[2] ‘Spot® - The Agile Mobile Robot’, *Boston Dynamics*. https://www.bostondynamics.com/products/spot (accessed Mar. 15, 2023).

[3] ‘Spot Specifications’. https://support.bostondynamics.com/s/article/Robot-specifications (accessed Mar. 15, 2023).

[4] ‘Modular Snake Robots - CMU Biorobotics’. http://biorobotics.ri.cmu.edu/projects/modsnake/ (accessed Mar. 28, 2023).

[5] ‘Eyein Walkie Talkies 2 PCS For Children, Rechargeable Walkie Talkies 22 Channels 2 Ways Radio with LCD Screent Receiving Signal Within 3 Km, Children’s Birthday Gift For Travel, Camping, and Hiking : Amazon.sg: Toys’. https://www.amazon.sg/Eyein-Children-Rechargeable-Receiving-Childrens/dp/B0BVMGSB4B/ref=sr_1_5?adgrpid=142212404075&hvadid=622542035176&hvdev=c&hvlocphy=9062530&hvnetw=g&hvqmt=e&hvrand=11623035342776628681&hvtargid=kwd-43891236673&hydadcr=4880_429751&keywords=amazon%2Bwalkie%2Btalkie&qid=1681797006&refinements=p_n_prime_domestic%3A7993547051&rnid=7993546051&sr=8-5&th=1 (accessed Apr. 18, 2023).

Section 9 - Appendix

9.1 Response Packets

Response	Details	Packet Check Status
RESP_OK	When Arduino receives either a move or query command successfully, return an ok packet. For move commands, on top of sending the ok packet, also send the distance packet.	Success
RESP_STATUS	When Arduino receives a get 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 with the wrong checksum, return this response 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_COLOR	When Arduino receives a get color query command successfully, returns the color packet, the distance packet and the ok packet	Success
RESP_DIST	Same as RESP_OK	Success

9.2 Color detection algorithm

```
if (redGreenDiff >= COLOR_THRESHOLD and distance <= DIST_THRESHOLD) {
        if (red < green) {
                if (green > RED_THRESHOLD) printf("\nRED!\n");
                else printf("\nORANGE!\n");
        } else {
                if (blueGreenDiff < GREEN_THRESHOLD) printf("\nGREEN!\n");
                else printf("\nBLUE!\n");
        }
} else printf("\nNo color detected!\n");
```

9.3 Bare-metal Programming Code Examples

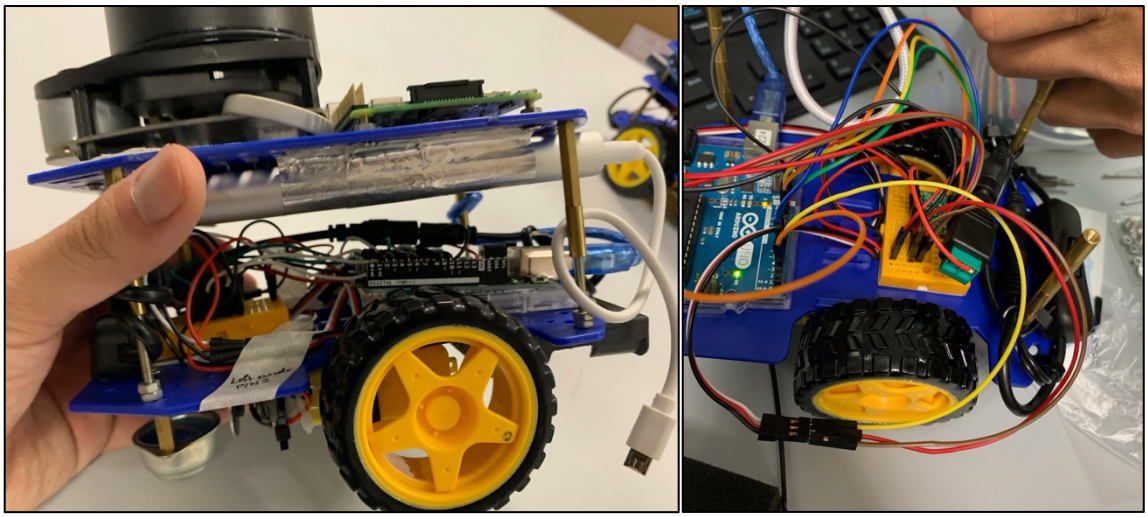
```
void enablePullups() {
  DDRD &= B11110011;
  PORTD |= B00001100;
}
```

```
void startMotors() { // Start PWM for motors
  TCCR0B = B00000011;
  TCCR1B = B00000011;
  TCCR2B = B00000011;
}
void rightMotorReverse() {
  TCCR1A = B00100001; // Enable waveform on OC1B pin
  TCCR2A = B00000001; // Disable waveform on OC2A pin
}
```

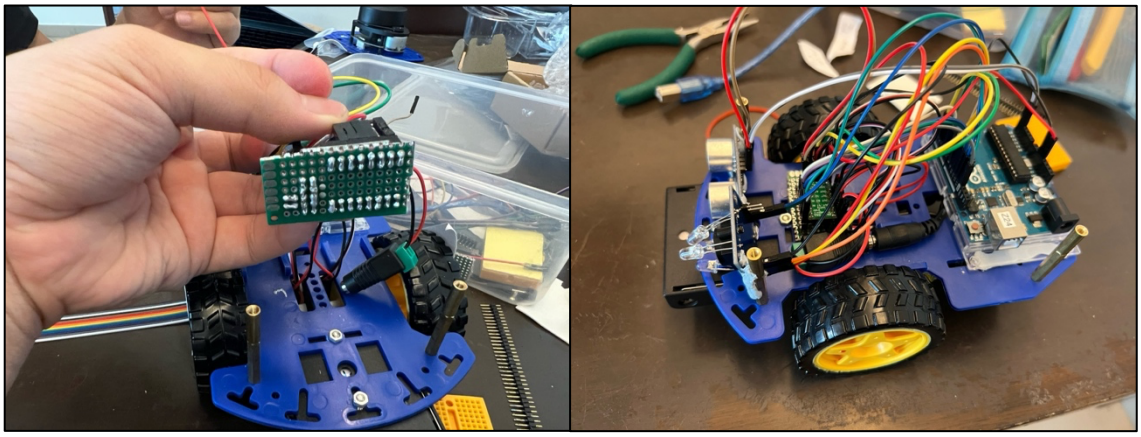
9.4 getch() Code

```
#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);
}
```

9.5 PCB Breadboard



Before - Messy and insecure cable management using solderless breadboard



After - Tidy and secure cable connections using custom soldered PCB