CG2111A Engineering Principles and Practices

Semester 2 2021/2022

"Alex to the Rescue"

Design Report

Team: B01-3A

Name	Student ID	Sub-Team	Role
Benjamin Long Wei Ming	A0184978A	Software	Lead
Gan Zhen Yang	A0236526A	Software	Co-Lead
Nikhil Shashidhar	A0233992W	Hardware	Lead
Ajay Shanker	A0234806E	Hardware	Co-Lead
Bian Rui	A0239073A	Firmware	Lead

Section 1 Introduction

Alex is a tele-operated land rover designed to search and rescue survivors in the aftermath of a natural disaster. Basic functionalities were tested on a 216cm by 135cm grid littered with obstacles and walls, requiring Alex to maneuver the hazardous environment and map the surroundings on a laptop device using visualization tools.

Basic functionalities include the ability to:

1. Be controlled remotely over a wireless network
2. Scan its environment using LiDAR and relay that information to the operator to be mapped using Hector SLAM
3. Forward, Backwards, Left and Right directional control and variable power control options to overcome obstacles such as humps

The components used to achieve these functions are as follows:

1. Arduino UNO - Interrupts, PWM (Timers)
2. Ultrasonic Distance Sensors x2
3. Raspberry Pi Model 3 - WiFi, SSH
4. UART Communication Protocol between (1) and (3) on Pi through alex-pi.cpp
5. Personal Laptop (Master) - for SSH into (3) to control Alex remotely
6. RPLidar A1 - sends data to RPi
7. DRV8833 Dual H-Bridge Motor Driver - for dual motor control
8. ROS Noetic/Kinetic - Hector SLAM, joy_node, rviz
9. PlayStation 4 Controller

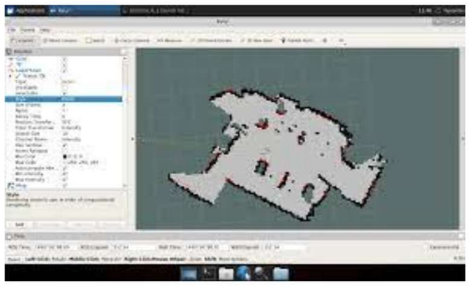

Fig 1. HECTOR SLAM with rviz

Fig 2. Playstation 4 controller being used to control Alex

Section 2 Review of State of the Art

1. UAV SAR Drone

This Unmanned Aerial Vehicle uses drone technology to locate and save lives during times of crises. It is designed to provide real time data and imaging in challenging conditions eliminating risk to personnel [1].

Fig 3. UAV

Aiding in the search for missing persons is the use of infrared (IR) thermal imaging cameras that can detect human body heat. This will significantly increase the probability of finding people or objects, regardless of day or night operation.

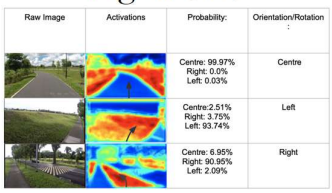

Fig 4. Model

One of the most challenging tasks is the need to travel to inaccessible terrains autonomously. The TrailNet deep neural network is used to train an AI model that allows the drone to navigate to the desired points by itself. However, running the model requires substantial processing power, and multiple cameras to provide a wide field of view to remove blind spots, which drains battery quickly and reduces flight range. It also cannot withstand extreme weather conditions.

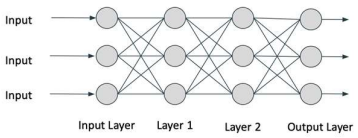

Fig 5. Diagram of a Deep Neural Network

2. Robot RAPOSA USAR

This Urban Search and Rescue Robot RAPOSA [2], is built to search for potential survivors of disasters (such as a building collapsing) in areas that are unsafe for humans to traverse. The main components of its mechanical design involve 2 modules (main and frontal body) and 2 side tracked wheels to provide locomotion for both modules. It is designed to exchange commands once the orientation of RAPOSA is flipped.

Fig 6. External view of RAPOSA

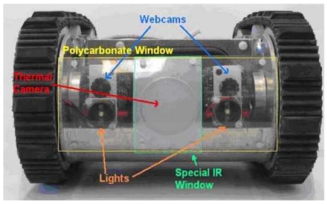

Fig 7. Front View showing various sensors

The thermal camera is sensitive to heat radiation, allowing the detection of heat sources. This is displayed in real-time on the GUI via tethered/wireless (tele)communication. The former provides better autonomy and ensured bandwidth, but the cables may get stuck mid-operation, limiting mobility. The wireless solution is ideal for indoor communication within 50m, but falls short in situations involving rubble and debris (obstacles), especially with EM noise (greatly affecting data transmission).

Fig 8. GUI seen by the operator

Section 3 System Architecture

Power

- The RPi is powered by a 20000mAh portable charger (and hence so is the RPLidar & Arduino)
- The motors are powered via the DRV8833 Dual H-Bridge Motor Driver by the 6V supply.

Communications & Networking

- The laptop connects to RPi through SSH over the same network (mobile hotspot/router)
- The RPi and Arduino communicate via USB (physical connection), 9600 baud rate 8N1 UART (communication protocol)
- The RPi polls the RPLidar for point cloud data by sending request packets (and displays the data on the laptop through rviz)

ROS

- The laptop runs ROS1 Noetic while the RPi runs ROS1 Kinetic
- The node '/hector_mapping' on the master (laptop) subscribes to the '/scan' topic published by the nodes '/rplidar'
- The node of RPi, '/pi', subscribes to the joystick topic '/joy'.
- rviz (ROS Visualization) then displays the processed point cloud data '/hector_mapping' visually, to map out the surroundings.

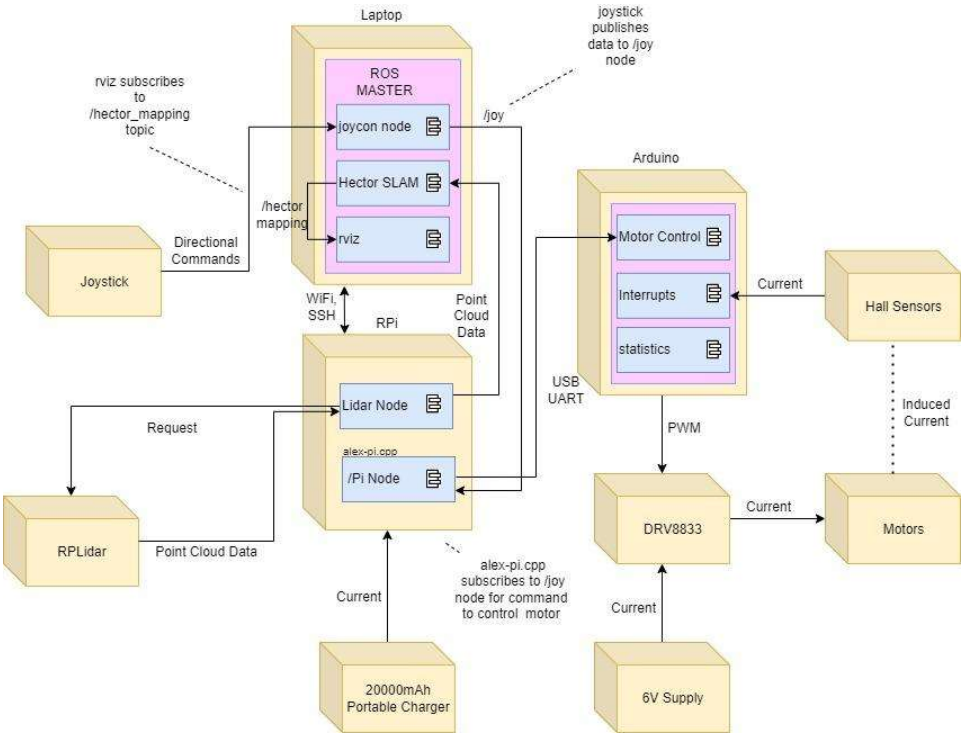

Fig 9. UML Diagram

Section 4 Hardware Design

Alex

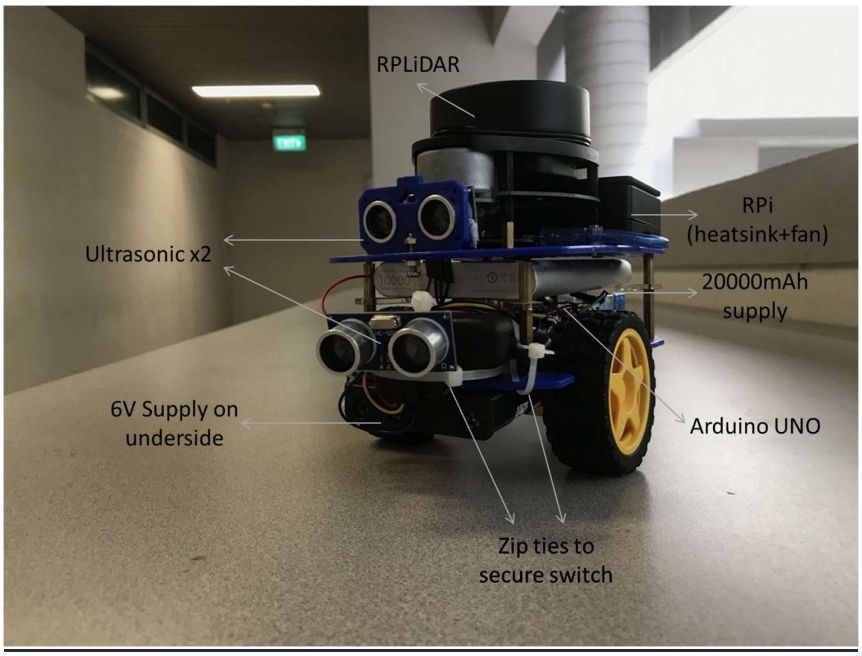

Fig 10. Photo of Alex

Pinout

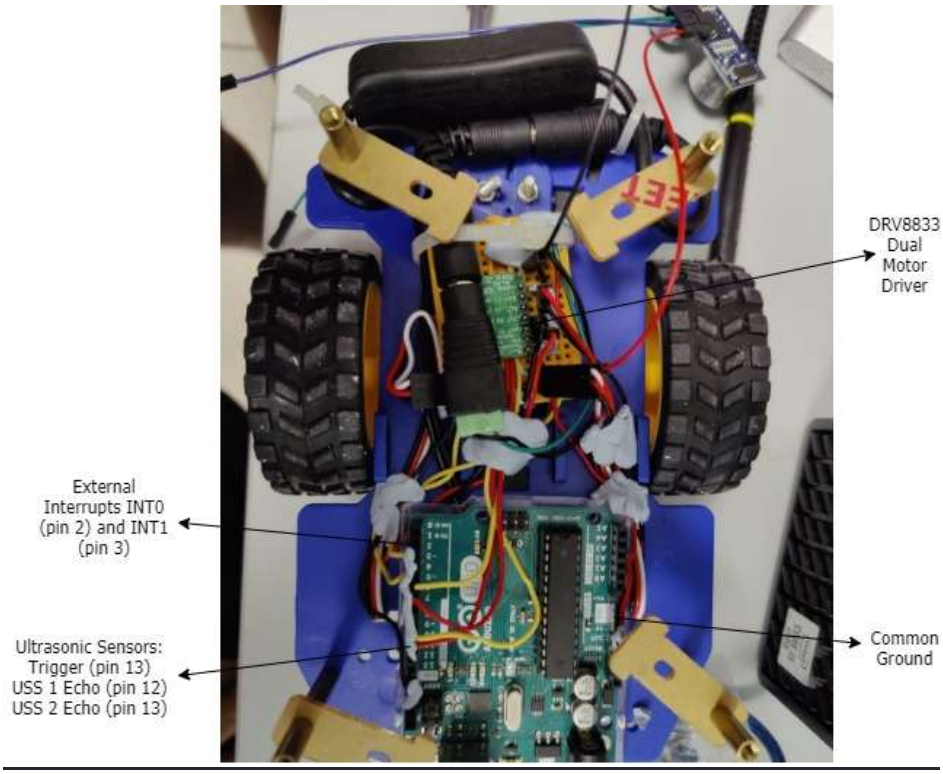

Fig 11. Wiring

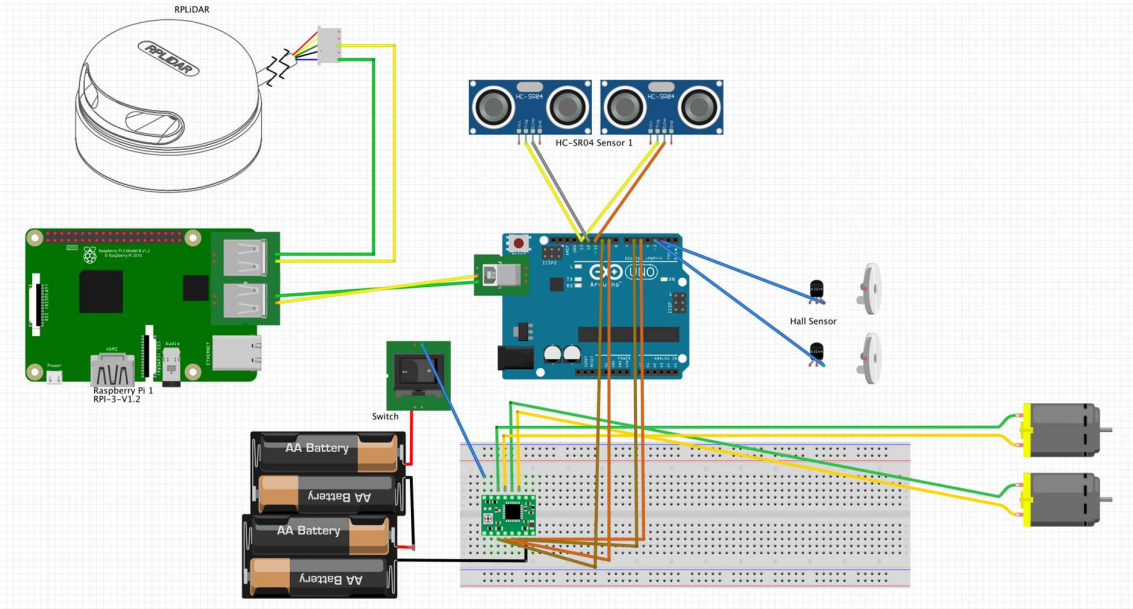

Fig 12. High-level (rough) circuit diagram with some connections omitted for clarity

PS4

The /joy node broadcasts the messages which are subscribed by the RPi node to send the commands to the Arduino for motor control.

Fig 13. PS4 Schematic

Section 5 Firmware Design

The Arduino controls the motors according to the command sent by the joystick and returns the movement data to the RPi. Additionally, 2 ultrasonic sensors are placed in front; one directly in front and one at an angle to detect the presence of elevation (ramp). These are set to poll the distance data in the main loop.

Arduino setup:

1. Enable external interrupts INT0 and INT1 (pins 2 & 3) to the 'FALLING' edge
2. Start serial communication at 9600 bps
3. Enable TIMER2 with output compare A match and overflow interrupt enabled in fast PWM mode with a prescaler of 32
4. Enable pull-up resistors
5. Clear counters for movement data.

After setup, the hall sensors will update the necessary movement ticks variable 8 times for every cycle of wheel movement through INT0 and INT1.

In the main loop, Arduino is polling for the incoming command through UART at a baud rate of 9600 bps and in 8N1 format. The command is received in a struct of TPacket consisting of packet type, command, string data and numerical data with appropriate paddings (dummy char array). If the packet is corrupted, the Arduino will send back an error packet of the respective error type. For example, the sendBadCommand() function tells the Pi that the Arduino does not understand the command sent. The Arduino then uses the sendResponse() function to serialize the packet (to buffer) and send it to the Pi. Else, the Arduino will respond with an 'ok packet', sendOk(), and handle the movement command. The UART was implemented using interrupts and circular buffers using the AgileWare CircularBuffer library.

Depending on the movement mode (Forward, Reverse, leftTurn, rightTurn), the respective pins controlling movement (leftForward, leftReverse, rightForward, rightReverse) will be set to their PWM values according to the speed in command packet. The PWM is controlled by setting the OCR2A value.

We control the movements using the buttons on the PS4 controller, with the Pi automatically sending "stop" when no buttons are pressed. For moving forward and reversing, we usually set the distance parameter to 0 to allow Alex to move forward indefinitely while the respective button was pressed. For the right and left turns, we used the revolution data from the encoders to limit the angle turned per button press based on the angle parameter received from the Pi. This was necessary to avoid turning too fast or for too large an angle, which would disrupt the Hector SLAM.

Section 6 Software Design

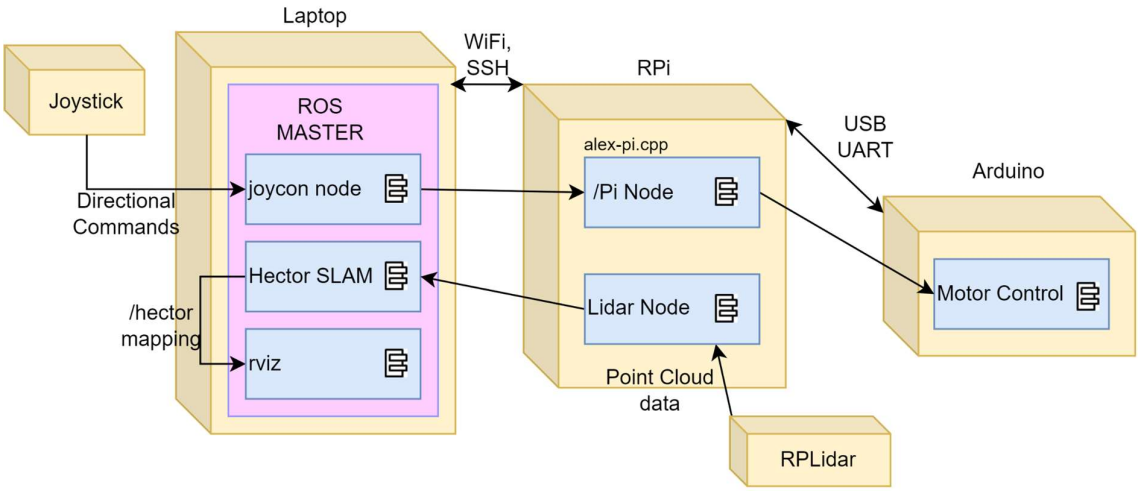

Fig 14. Overall Software Design

Our system involved three devices, the operator's laptop, RPi and Arduino board which is connected through Wifi(SSH) and USB(UART). **Teleoperation** and **slam mapping** is the main function of the system. For Teleoperation, the joystick command is published to RPi for translation into motor control command before sending it to the Arduino board on Alex. For slam mapping, lidar sends point cloud data to Hector SLAM node for processing before visualizing on rviz.

The high-level **Teleoperation** algorithm on the Pi is shown in Figure 14. The **port number** of the Arduino is entered by the user before starting the connection between Pi and Arduino. A **"Hello" packet** is sent from Pi to ensure the connection is set up successfully. By **initiating Pi** as a ROS node, Pi can now **subscribe to the joystick command** from the operator. **While the exitFlag is not triggered**, sub_callback function (Fig 15) is called to process the incoming joystick command.

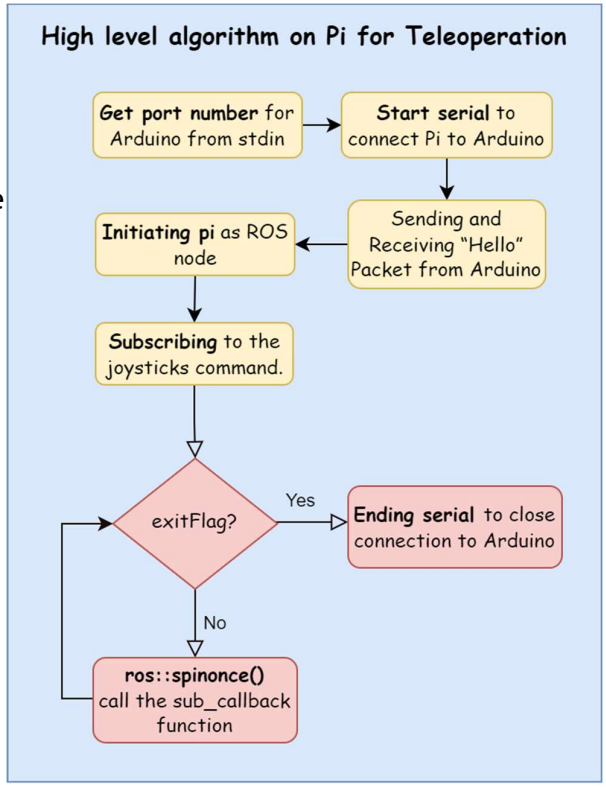

Fig 15. High level algorithm on Pi for Teleoperation

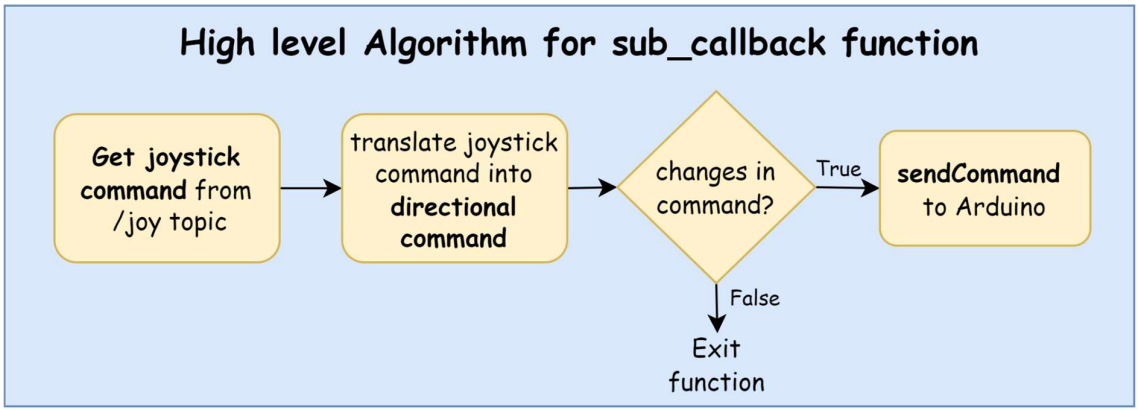

Fig 16. High level Algorithm for sub_callback function

sub_callback function is triggered constantly throughout the whole process. To handle this situation, the motor control command will **only** be sent when there are changes of command from joysticks. This is to prevent the joysticks command from jamming the communications channel with the Arduino.

Section 7 Lessons Learnt - Conclusion

1. Most important lessons:
   a. How to experiment to find the best settings

      The Hector SLAM rviz settings took a while to calibrate, and we learned that with patience can we achieve reliable settings that would have otherwise been a hit-or-miss. It took hours of modifying Alex's turn speed hand-in-hand with the rviz resolution settings to ensure the visualization didn't collapse while rendering. Spinning Alex too fast would result in the point cloud sample data being corrupted and result in a bad rviz render of the map. Spinning too slow would not allow us to complete the render in under 5 minutes.

      The point we are trying to make here is that there is tremendous patience required in trial-and-error. Only after hours were we able to achieve the optimal settings that enabled us to map Alex's surroundings without having to repeatedly restart rviz from a corrupted map.

   b. How to troubleshoot problems

      When facing issues with Alex, which essentially happened every lab session, we took the following approach (Oracle):
      1. Identifying the problem (and symptoms)
      2. Eliminate non-issues
      3. Narrow down to the cause
      4. Fix it

      This framework helped us combat every problem we faced. For example, from the very first lesson when assembling Alex, when following the instruction guide on the construction, we ended up with a very high center of gravity (problem) which we assumed would cause Alex to topple down the ramp. Running through the components of Alex (eliminate), we realized that the 20000mAh portable charger (narrow down) was placed too high up and needed to be wedged in between the chassis, which worked in lowering the COG, making Alex more stable (fix!).

2 Greatest Mistakes:

   a. Trying to go over the hump

      Controlling Alex's speed over the ramp was a difficult task because of the incline. This resulted in us using the 'boost' function on the PS4 controller, which was 90% PWM. It was not the *best* idea since there was much uncertainty as to how Alex would descend without having mapped the other side of the ramp. During our trial runs, Alex succeeded in overcoming the

ramp without facing any penalties. In our final graded run, much to our dismay, Alex crashed into a wall. We realized that in the intense time pressure, it didn't occur to us to map the area on the other side of the ramp. This was a mistake that cost us the marks from achieving a perfect score in the graded run.

b. Disregarding the importance of the motor power supply

   We did not pay attention to the 6V voltage supply until we observed the peculiar behavior of Alex when calibrating the PWM turn speed with time. The 6V supply drained fairly quickly and when we revisited the same PWM turn speed at a later time in the lab, there was an observable difference in performance - which resulted in confusion since there *surely* should not have been any discrepancy between the same set PWM values. It then occurred to us (after hours and hours of modulation and calibration) that the lower voltage of the batteries from the hours of use directly affected the turn speed.

References

1. A. Valsan, P. B., V. D. G. H., R. S. Unnikrishnan, P. K. Reddy and V. A., "Unmanned Aerial Vehicle for Search and Rescue Mission," 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), 2020, pp. 684-687, doi: 10.1109/ICOEI48184.2020.9143062.

2. Carlos Marques. "A search and rescue robot with Tele-Operated tether docking system." vol. 34, no. 4, 2007, p. 7.

3. Oracle. *General Steps to Troubleshoot an Issue*, https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/introclientissues002.html. Accessed 21 April 2022.