**CG2111A Engineering Principle and Practice**
Semester 2 2021/2022

# "Alex to the Rescue"

# Final Report

# Team: B01-2A

| Name | Student # | Main Role |
|---|---|---|
| Tan Wei Li | A0216235M | Hardware |
| Bui Phuong Nam | A0244121X | Software |
| Anderson Leong Ke Sheng | A0223111E | Software |
| Chan Ee Hong | A0233898L | Sublead |
| Aung Phone Naing | A0234363J | Lead |

## Section 1 Introduction

A robotic vehicle, hereby referred to as Alex should be built for search and rescue operation purposes. The motors, embedded controllers, power bank, chassis, and Lidar provided should be used without any modifications.

The embedded controllers provided are a Raspberry Pi, connected to an Arduino Uno. Tele-operation is carried out by having the laptop communicate with the raspberry pi to command Alex.

Hence, Alex should be teleoperated from a distance away and map a maze about $3m^2$ in the area as it moves. It should also be able to detect people in the map that is produced by the Lidar mounting and be able to navigate the maze from a designated start point to a designated endpoint. The minimal requirement for Alex is that it should be able to go straight, or turn left/right. There may be unfriendly terrains such as a hump in the maze and Alex should be able to navigate over it and any forms of collision should be avoided.

## Section 2 Review of State of the Art

### Snakebot

Developed by Carnegie Mellon University, this modular snake-like robot can move on land and underwater to explore places that are difficult for humans to access (Aupperlee, 2021). It has a camera at the front of the robot that streams live video back to the teleoperator. It moves across land by twisting and turning different joints in its body, and can autonomously adjust its movement to climb vertical obstacles based on the pole's diameter. To move underwater, it has small propulsion fans on each of its modules, allowing precise orientation control in confined spaces. It is lightweight and can reach high speeds underwater. It is easy to learn how to operate the snakebot, as it has many LED indicator lights, a robust control box where live video is streamed back from the robot, and a lightweight compact tether (Gossett, 2019).

**Strengths** of snakebot include being able to fit in tight spaces (under debris, the hull of ships, nuclear reactors) to relay critical information which can save lives, time, and money, lightweight (easy to transport and deploy), and being able to traverse both on land and underwater.

**Weaknesses** include being unable to carry heavy payloads, unable to perform complex tasks (like lifting or moving objects), and being unable to move as fast as other robots over open terrain.

### Colossus

This fire-fighting robot by Shark Robotics helped the Paris Firefighter Brigade put out the fire at Notre Dame cathedral (Holley, 2019). It weighs about 500kg, and has treads as wheels to maneuver over uneven terrains such as stairs and debris. A water hose can be connected to the back of Colossus to allow it to spray more than 2000 liters of water every minute. It is also strong enough to transport heavy fire fighting equipment and carry victims out of dangerous areas. Colossus has a 360-degree high-definition thermal camera that allows teleoperators to have a good idea of what is happening (Gossett, 2019).

**Strengths** of Colossus include the ability to carry heavy payloads, the ability to withstand extreme heat, and the ability to navigate across urban obstacles like stairs and buildings.

**Weaknesses** include being hard to transport due to its heavyweight, and not being as agile and fast as smaller robots.

## Section 3 System Architecture

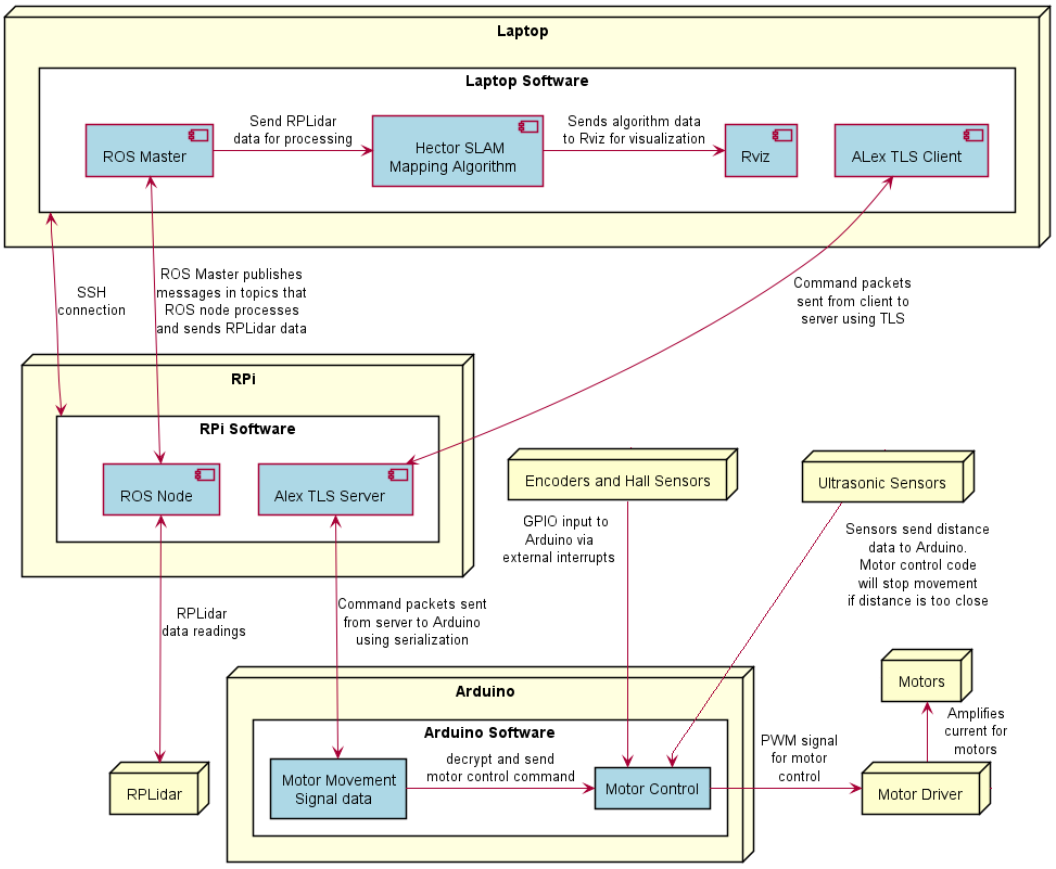
Laptop
Laptop Software
ROS Master
Send RPLidar data for processing
Hector SLAM Mapping Algorithm
Sends algorithm data to Rviz for visualization
Rviz
ALex TLS Client
SSH connection
ROS Master publishes messages in topics that ROS node processes and sends RPLidar data
Command packets sent from client to server using TLS
RPi
RPi Software
ROS Node
Alex TLS Server
Encoders and Hall Sensors
Ultrasonic Sensors
GPIO input to Arduino via external interrupts
Sensors send distance data to Arduino. Motor control code will stop movement if distance is too close
RPLidar data readings
Command packets sent from server to Arduino using serialization
Motors
Amplifies current for motors
Arduino
Arduino Software
decrypt and send motor control command
Motor Movement Signal data
Motor Control
PWM signal for motor control
Motor Driver
RPLidar

**Figure 1: System Architecture of Alex**

The above figure depicts the high level components of Alex. Yellow blocks represent hardware, in which there may be relevant software components to be emphasized which are indicated in blue rectangles.

## Section 4 Hardware Design

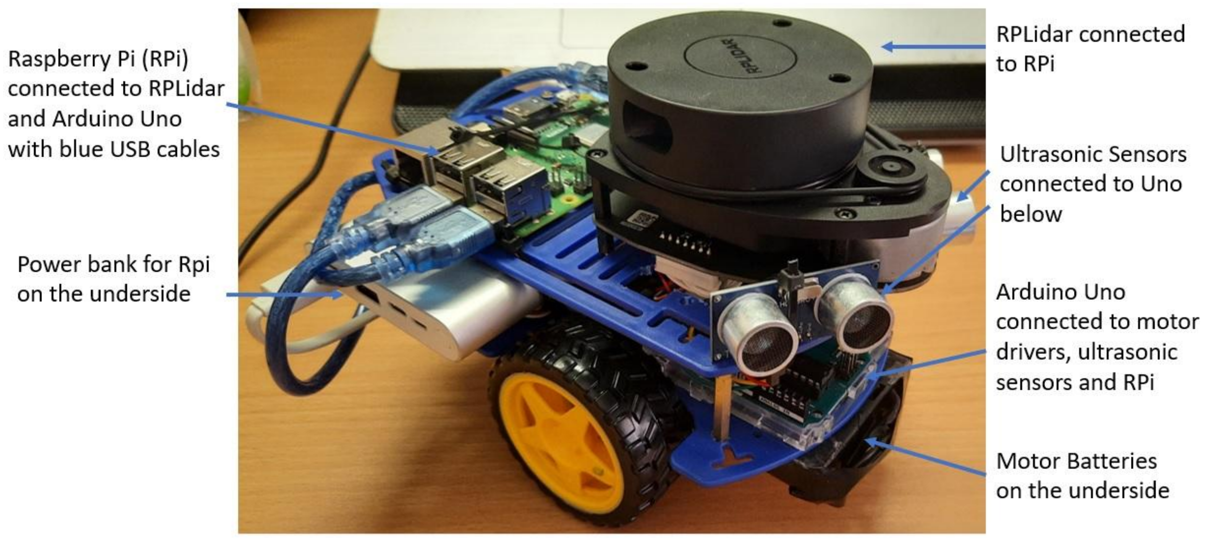
Raspberry Pi (RPi) connected to RPLidar and Arduino Uno with blue USB cables
Power bank for Rpi on the underside
RPLidar connected to RPi
Ultrasonic Sensors connected to Uno below
Arduino Uno connected to motor drivers, ultrasonic sensors and RPi
Motor Batteries on the underside

**Figure 2: Top View of Alex**

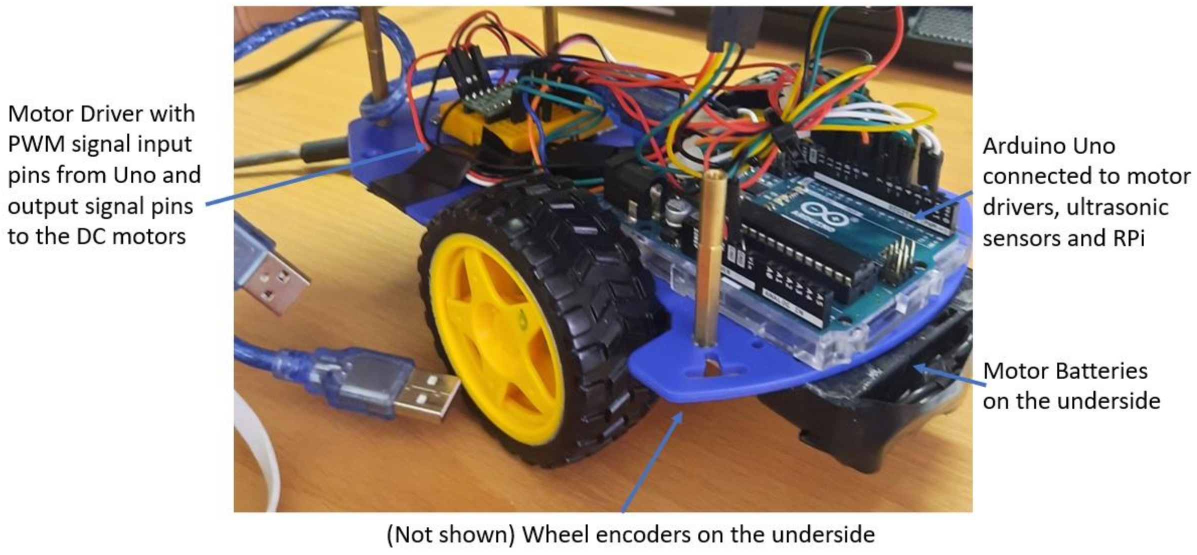
Motor Driver with PWM signal input pins from Uno and output signal pins to the DC motors
Arduino Uno connected to motor drivers, ultrasonic sensors and RPi
Motor Batteries on the underside
(Not shown) Wheel encoders on the underside

**Figure 3: Bottom Layer of Alex**

## Section 5 Firmware Design

<u>Overview</u>

Communication between the Pi and the Arduino always begins with the Pi sending a TPacket of which the Arduino will send back an acknowledgement in the form of an "ok" packet once received and read properly. In the Alex code, it was modified from the given sample through the implementation of an ultrasonic sensor code to prevent head-on collisions, conversion of all serial communications protocols and Pulse Wave Modulation (PWM) generation to bare metal as well.

<u>Communication Protocols</u>

- All communications are initiated by the Pi through the sending of a TPacket while the Arduino will constantly keep listening out for packets.
  - The TPacket type consist of the following data :
    - TPackettype (Command, Response, Error, Message and Hello)
    - Command (a char type determining the command for the Arduino to execute.
    - 2 char padding to make up to 4 bytes
    - An array of 32 bytes to be able to store messages
    - An array of 16 32-bit integers to store any parameters to be communicated

- Whenever a packet is received by the Arduino, the packet handler checks the type of packet sent.

- If the packet is an 'error' type, the Arduino will send back an 'error' packet specifying the type of error which are :
  - Bad Packet received
  - Bad Packet checksum
  - Bad Command issued
  - Bad Response

- And if the packet is of a ‘command’ type packet, it will be parsed into a command handler function where the Arduino :
    - Send an ‘ok’ packet to PI as an acknowledgement when the packet is properly received and read.
    - Calls relevant functions to execute the command parsed (COMMAND_FORWARD, COMMAND_REVERSE etc.)
    - Commands called will have params in the packet as inputs to activate the motors accordingly.

- After the command is parsed, deltaDist or deltaTicks will be greater than zero if the command type is related to moving (COMMAND_FORWARD, COMMAND_TURN_LEFT etc.).
    - A new target total distance traveled (or target ticks for turning commands) will be computed.
    - If the current distance traveled (or ticks) has not overcome the new targets, Arduino drives the motors as the command dictates.
    - When the current distance traveled (or ticks) has overcome the new targets, zero will be written into deltaTicks or deltaDist and targetTicks or newDist and stop() function is called to stop the motors.

## UART Circular Buffers & PWM Baremetal

In the Alex.ino code, serial communications and generation of PWM were baremetalized. To achieve maximum efficiency for serial communications, UART communications in an interrupt manner was also deployed with the help of the Circular Buffer library from AgileWare on the Arduino IDE, to ensure a more efficient form of data transmission. With circular buffers we are able to read from and write to the buffer at the same time as long as the buffer is not full to speed up the process.

As for the PWM aspect, a phase-correct-PWM mode was used to generate the required power for each motor. As for the timers, timer 1 and 0 were used with timer 1 set to a 8-bit phase-correct-PWM mode so as to ensure consistency between the motors. A calibration for the wheels in various movement types was also carried out, where the faster wheel will have a constant multiplied to its compare-match (OCR_A/OCR_B) values to slow it down so as to ensure both wheels have the same power delivered and move at the same speed.

## Ultrasonic Implementation

The ultrasonic sensors were deployed at a 30 degree angle from the front of Alex so as to prevent Alex from colliding into obstacles in front and to the sides. A 30 degree angle was chosen so as to ensure that there was sufficient clearance between Alex and the walls both in front and to the side to be able to execute a turn in order to avoid obstacles.

As for how it works, the ultrasonic sensors will continuously detect the proximity of Alex from the wall as Alex moves and when the distance from the wall reaches a critical value of 5.5cm, the arduino will trigger a stop command, fixing Alex in its tracks. This is followed by the prompting of the user to use a master command in order to move Alex again, without which Alex will no longer respond to any commands input by the user. This acts as a fail safe to prevent any possible collisions due to accidental commands after figuring that Alex is too close to the wall.

## Section 6 Software Design

High level steps

1. Initialization
    a. Start serial port with baud rate of 9600 on port 5000, with 8N1 serial protocol.
    b. Create a serial listener thread.
    c. Create a server that authenticates clients when they connect and send Alex's certificate.
    d. Send a 'hello' packet to the arduino and wait for a response packet from the arduino.
        i. If the Pi receives an 'ok' packet, it means a connection with Arduino has been established.
        ii. If the Pi receives any 'error' packet, it displays the type of error (e.g. bad checksum, bad magic number).
2. Receive user commands from client
    a. A worker function continuously reads from the SSL into a buffer as long as the network is active.
    b. When there is data (more than 0 bytes) in the buffer, it checks if the 0th byte of the buffer is NET_COMMAND_PACKET. If it is, the data is read from the buffer into a 'command' packet.
    c. The Pi copies over the parameters into this new 'command' packet
        i. Command (e.g. COMMAND_FORWARD, COMMAND_REVERSE)
        ii. Distance
        iii. Power
3. Send commands to Arduino
    a. This 'command' packet will be sent to the Arduino through UART.
    b. The packet is first serialized into a buffer before being sent.

Additional noteworthy software-related stuff

a. TLS
i. The operator sends movement and information retrieval instructions to RPi
ii. RPi sends back sensor data after executing movements or when the operator requests for sensor data

b. SSH from laptop to Pi
For remote control, we use Secure Socket Shell (SSH) to allows user to communicate with Alex via network by viewing Raspberry Pi mounted on Alex through a monitor. To do that, we enable SSH on Raspberry Pi, find the monitor IP and then SSH into it.

## Section 7 Lessons Learnt - Conclusion

**Lesson 1** - Automation of Alex's routines

One of our key takeaways from the project was to always try and automate long processes as much as we can when dealing with projects requiring multiple inputs for starting up. Our group had to navigate through 4 different terminals in order to boot up Alex and Hector slam, with 2 of the terminals being on the primary laptop to establish a TLS connection and the remaining 2 on the Virtual Machine installed to start up the ROS and RViz software for our mapping. This led to many possible sources of error as we were manually inputting specific and long commands on each terminal just so as to start up Alex before we begin mapping. Hence, with the use of scripts and aliases to automate various tasks on each terminal, things could have been expedited to let us have a hassle free time with starting up Alex's routines.

**Lesson 2** - To be less ambitious and work on things gradually

Another key takeaway was to set realistic goals for ourselves which we can achieve by doing things step-by-step. Starting off Alex, all of us were not too familiar with how we can modify Alex and yet we decided to set ambitious goals such as to implement bare metal code as well as ultrasonic sensors and rewiring Alex to achieve better cable management. We ended up neglecting the basic goals that would have helped in making a functioning Alex such as calibrating Alex properly or securing a platform to handle the mapping using the Lidar on Alex. This in turn led to us having to rush through the basic implementations of running Alex which showed in our failure to boot up Alex in our Demo run. Therefore, for future projects we will always settle the basic requirements and ensure its functionality before heading out to fulfill the additional achievements.

**Mistake 1** - Using phone hotspot for TLS connection at a distance

One mistake we realized after the project was that we did not take extra steps to secure the connection between our laptop and Alex's Raspberry Pi. Therefore, during the run we faced some difficulties in controlling Alex that could be due to the weak connection between the laptop and the Raspberry Pi. We had focused on implementing the core features required of Alex, such as its mapping capability, and did not realize that the network connection could be a potential point of failure early on. To rectify this issue, we should have prepared and used an internet router to help strengthen the network signal. The bigger lesson learnt here is to at times take a step back and review the project as a whole, so as to spot potential points of failure early on.

**Mistake 2** - Implementing the source code management late

Another mistake that we made was not managing the source files properly from the start. The files for the Arduino and the Raspberry Pi were haphazardly stored in the Raspberry Pi and our laptop, some files had the same names while others had multiple copies. We relied on VNC Viewer to transfer files to and from the Raspberry Pi, which took time and was a hassle. We only started using GitHub to manage the source files properly during the last week of the project, and realized that Git pulling from the main repository directly from the Raspberry Pi was much more efficient. This prevented accidentally transferring wrong files or running old versions of files. However, we only started doing this too late, and wasted a lot of time and effort on fixing issues and mistakes that could have been avoided if we managed the source files properly from the beginning.

## References

1. Aupperlee, A. (2021, April 13). CMU's Snakebot Goes for a Swim. *Carnegie Mellon University*. Retrieved April 17, 2022, from https://www.cmu.edu/news/stories/archives/2021/april/snake-robot.html

2. Gossett, S. (2019, June 25). 10 examples of Rescue Robots. *Built In*. Retrieved April 17, 2022, from https://builtin.com/robotics/rescue-robots

3. Holley, P. (2019, April 17). Firefighters had a secret weapon when Notre Dame caught fire: A robot named 'colossus'. *The Washington Post*. Retrieved April 22, 2022, from https://www.washingtonpost.com/technology/2019/04/17/firefighters-had-secret-weapon-when-notre-dame-caught-fire-robot-named-colossus/