CG1112 Engineering Principles and Practice

Semester 2 2020/2021

"Alex to the Rescue"
Design Report
Team: B02-6A

Name	Student ID	Sub-Team	Role
Zhuang Jianning	A0214561M	Firmware & Software	Algorithm Design & Implementation
Kwek Ming Shun	A0206038M	Hardware & Movement	Hardware Design & Movement Calibration
Alexander Tan Jun An	A0199267J	Wireless Operations	Wireless Communications
Flores Wraine Melveirich Rivadeneira	A0216014X	Environment Mapping	LiDAR Operations & SLAM

Table of Contents

Section 1 Introduction and System Functionalities

According to the Centre for Research on the Epidemiology of Disasters (CRED), natural disasters have claimed an average of 68,000 lives per year between 1994 and 2013 [1]. To reduce this number, faster and more efficient search-and-rescue (SAR) operations are necessary, with the first 72 hours being the "golden period" to locate and rescue survivors [2]. Fortunately, advancements in robotics have led to an increase in the usage of robots and drones in SAR operations [3]. Not only does this allow for operations in hostile environments, the information gathered by the robot can also be analysed quickly to organise effective rescue operations.

Alex is a SAR robot that can be remotely controlled. It will be able to aid in accelerating SAR operations through environment mapping and survivor location. The essential functionalities Alex would need to achieve its aim are as follows:

1. Remote communications:
   Alex can be commanded wirelessly by an operator to explore its environment. It can also relay information about the mapped environment and the colour of objects back to the operator.
2. Environment Mapping:
   Alex is able to analyse data received from the LiDAR to map out the environment around it. This map can also help the operator locate survivors.
3. Movement:
   Alex is able to move straight, turn left, turn right and reverse based on the commands given by the operator. The speed of the movement can also be controlled by the operator. With the mapped environment, the operator can identify obstacles and avoid them.

Section 2 Review of State of the Art

Alex is a remotely controlled SAR robot whose main objective is to map out a simulated post-disaster environment. In this section, two existing SAR robots, WALK-MAN and Colossus, will be evaluated in terms of their features and components. The strengths and weaknesses of these remotely controlled robots will also be discussed to determine what can be adapted for Alex to build a more effective SAR robot.

1. WALK-MAN [4]

WALK-MAN is a humanoid disaster robot invented in 2015. It has accelerometers, force and torque sensors to balance the body and allow for human-like movements. WALK-MAN's strengths include its ability to perform humanoid tasks in a danger prone zone. It is able to use fire extinguishers, walk through doors and help evaluate its surroundings. However, WALK-MAN's weight and dimensions result in bulky movement and a limited range of motion. It also reduces WALK-MAN's capacity to operate within tight and confined spaces.

2. Colossus [5]

Colossus by Shark Robotics is a tracked robot with a modular design. It has a variety of modular options such as water cannons, manipulation arms and smoke extraction fans. Colossus' strength is its versatility due to its modular functionalities, ranging from obstacle clearance to fire extinguishing, which allows Colossus to cater to specific needs in various situations. Colossus also has a high heat resistance of up to 900°C which makes it suitable for SAR operations in buildings engulfed in flames. However, Colossus has a heavy weight of 485 kg, which prevents it from being deployed to fragile or devastated structures in fear of building collapse.

3. Adaptations for Alex

Incorporating the modular design of Colossus, Alex will also be able to swap between various components such as a colour detector, ultrasound sensor and buzzer to cater to various needs and scenarios. Addressing the issue of weight and dimensions faced by both WALK-MAN and Colossus, Alex is designed with a small and light body so that it can manoeuvre through tight spaces. The compact body along with the ability to pivot on the spot also gives Alex a large range of motion which overcomes the limitations of WALK-MAN.

Section 3 System Architecture

This section will focus on the structural design of the system used to achieve the essential functionalities of Alex. Alex has three main control components: the Operator, Raspberry Pi and Arduino which can send commands and receive data from the peripheral devices and sensors. Figure 3 below illustrates the high-level components as well as the information that passes between them.

The green boxes represent hardware devices while the yellow boxes represent software modules. The connection between the components contains the information that passes between them.

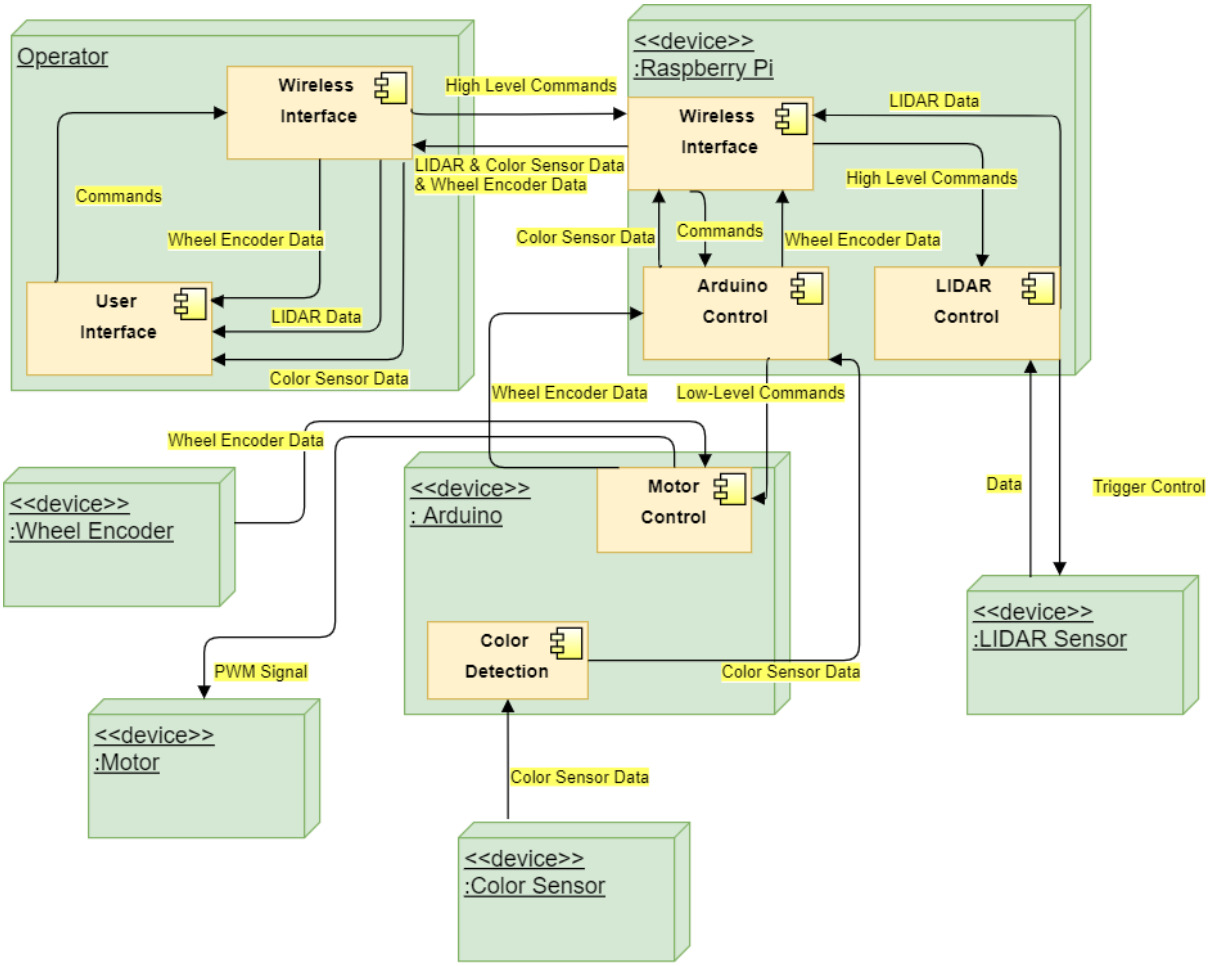

Figure 3: System Architecture of Alex

Operator

The operator component consists of two software modules, the user interface and the wireless interface. By utilising these modules, the operator can control Alex's movement and interpret the surrounding environment. The modules are described in further detail below.

1. **User Interface**
   The user interface allows the operator to issue commands to Alex and view data that is sent back. When a command is issued, it is sent to the Raspberry Pi through the wireless interface where it is converted to a high-level command. For example, a "move forward" command is converted to "rotate right wheel and left wheel forward simultaneously". When data is received, it is processed and displayed for the operator to view and make the next decision.
2. **Wireless Interface**
   The wireless interface sends the high-level commands, received at the user interface, over a wireless network to the Raspberry Pi mounted on Alex. Additionally, it receives data such as the LiDAR readings and Wheel Encoder data from Alex, which will be sent back to the user interface to map out the surrounding environment.

Raspberry Pi

The Raspberry Pi consists of a wireless interface and two software modules which control the Arduino and LiDAR.

1. **Wireless Interface**
   The wireless interface receives high-level commands from the Operator component and sends it to the appropriate software control module. It also receives LiDAR readings and Wheel Encoder data from the Arduino Control module and LiDAR control module and sends it back to the Operator component.
2. **Arduino Control Module**
   The Arduino Control module translates the high-level commands to low-level commands suitable for the Arduino to carry out. This controls the motor control software module on the Arduino. For example, "W" is converted to "Move forward 8 encoder ticks at 50% power". Additionally, it receives wheel encoder data from the Arduino and sends it back to the operator's device.
3. **LiDAR Control Module**
   The LiDAR Control module translates high-level commands to low-level commands that can be interpreted by the LiDAR device. It also receives the scanning data from the LiDAR device and sends it back to the operator through the wireless interface.

Arduino

The Arduino Uno consists of the Motor Control module and the Colour Detection module.

1. **Motor Control**

   The Motor Control module outputs a PWM signal at particular pins to rotate the motors according to the movement command received. Hall effect sensors are attached to the motors with a magnet to detect the rotation of the wheels. For each unit of rotation, pulses of current are sent to the Arduino Uno. This information is used to determine how far Alex has travelled and how much Alex turns when move commands are given. The data is also used to calibrate the movement commands and can be sent back to the operator to be displayed.

2. **Colour Detection**

   The Colour Detection module sends out white light from its 4 LED lights. Reflected light from the object is detected by the module. The reflected light is analysed by reading its respective R, G and B frequencies. Colours red or green are categorised by their range of values.

Section 4 Hardware Design

This section describes the thought process and reasoning for Alex's compact hardware design. It will be explored in three different parts. Firstly, the arrangement of different components in Alex. Secondly, the balanced weight of individual components when forming the robot. Lastly, the functionality of the final design with all components assembled.

Arrangement

The Lidar and Raspberry Pi are mounted on the top of the upper chassis. The powerbank, the Arduino and most of the wires for the circuitry are placed in between the upper and lower chassis. Finally, the batteries, motors, wheels and wheel encoders are placed below the lower chassis. The individual components are shown in the diagram of Figure 4.

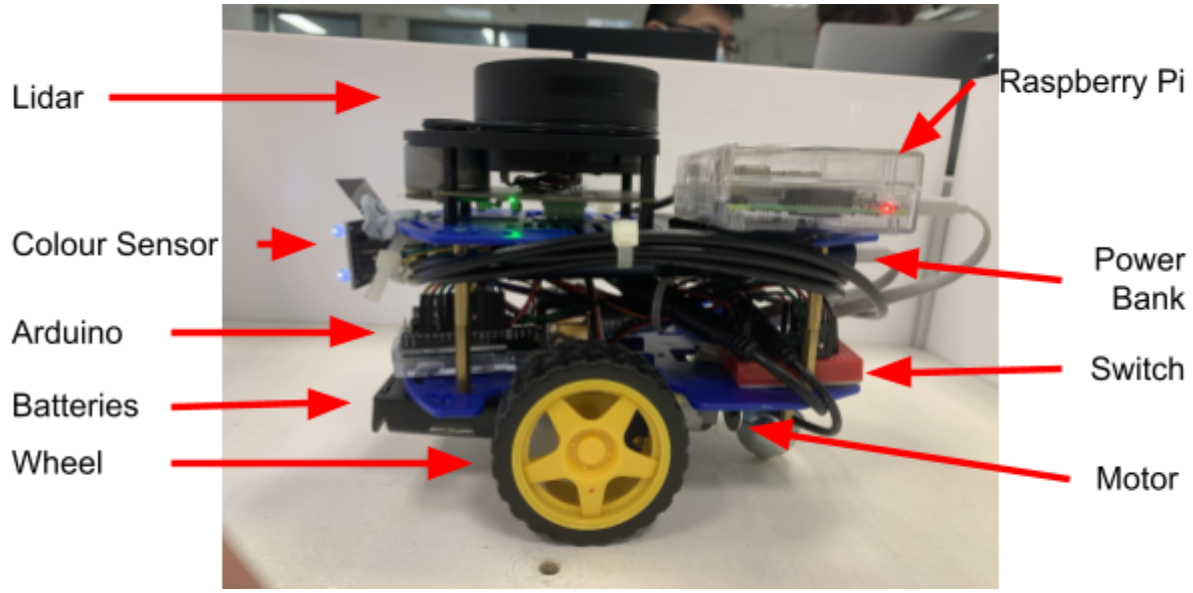

Figure 4: Alex Robot's left view

Balance

The Lidar and power bank are the two heaviest components of Alex. The Lidar is placed at the front to better detect and map the environment while the power bank is placed at the back to counterbalance the weight of the lidar. Alex would be stable due to the even weight distribution.

Functionality

The ports of the power bank and the Raspberry Pi are positioned to point outwards so that there is easy access. The Arduino and the Lidar are connected to the Raspberry Pi ports at the side. The wires are wrapped tightly around Alex to make it compact. The batteries and switch are attached to the lower chassis for the convenience of replacing the batteries or turning on/off the motors. Moreover, the overall design of Alex is designed to be compact so that it can easily move through the environment.

Section 5 Firmware Design

This section explains the firmware design of Alex, focusing on the high-level algorithm for the Arduino as well as the communication protocol between the Raspberry Pi and the Arduino.

High-level algorithm

The high-level algorithm for the Arduino includes initialisation, receiving commands and executing commands.

Initialisation

The initialisation phase is executed once when the source code for Arduino is uploaded. The various registers are configured to receive commands, execute commands and send back data.

Receiving Commands from Pi

Commands from the Raspberry Pi are sent over to the Arduino in a sequence of bits. The sequence of bits is deserialized and stored in a packet. The packet is read to identify the type of command sent.

Executing Commands

If the Arduino receives a movement command, it executes the command by writing appropriate values to the various Output Compare Registers. These registers determine the pulse width modulation (PWM) output to the pins which powers the motors. While executing the command, the Arduino also receives data from the wheel encoder to know much the wheels have turned. This data is then used to determine when the movement needs to be stopped.

If the Arduino receives a command to send data back to the Raspberry Pi, it creates a new TPacket of type response and populates it with the appropriate information. The packet is then serialised and sent over to the Raspberry Pi.

Communication Protocol

The communication protocol between the Arduino and Raspberry Pi uses a 9600 baud rate and the 8N1 serial configuration to serialise and deserialise data in the form of packets. The packetType and commandType variables store the respective packet and command types, dummy[2] is padding to ensure that the total number of bytes is a multiple of 4, data[32] sends debug messages back to the Raspberry Pi and params[16] contains 16 blocks of 4 bytes representing a certain parameter. Parameters include commands for movement of Alex or data from the wheel encoders. The packet structure is represented in table 1 below.

Table 1: Packet Structure of the Protocol

Variable	packetType	commandType	dummy[2]	data[32]	params[16]
Data Type	char	char	char	char	u32int_int
Byte Size	1	1	2	32	64

Section 6 Software Design

This section will explain the software design behind Alex, focusing on the high-level algorithm used by the Raspberry Pi to handle Teleoperation and Colour Detection. This section will also describe the operator interactions with Alex as well as details about Simultaneous Localisation and Mapping (SLAM).

High Level Algorithm on Pi

The high-level algorithm on the Pi to handle teleoperation and colour detection consists of initialisation, receiving commands from the operator and sending information back to the operator.

Initialisation

1. The operator connects to the Raspberry Pi from the laptop via Remote Desktop (e.g. VNC)
2. The operator compiles the Arduino source code and uploads it to the Arduino.
3. The operator opens a terminal and runs the **roslaunch.sh** bash script to start a RPLidar ROS node on the Pi. The RPLidar node is used to control the LiDAR and broadcast the LiDAR readings under the /scan topic.
4. The operator opens another terminal and runs the **nodeclient.sh** bash script which outputs the raw scan results on the console.
5. The operator opens one last terminal to run the server code compiled from tls-alex-server.cpp.

Receiving commands from the operator

1. Commands from the operator are transmitted over the network and read in by the worker via sslRead().
2. If data was written into the buffer, the handleNetworkData() function is invoked which checks if the packet is of type NET_COMMAND_PACKET.
3. The packet is broken down in handleCommand(), where the command type and any parameters the user passed in are read and saved in a TPacket to be sent to the Arduino.
4. Based on the command from the operator saved in the first index of the buffer, the corresponding command type for the TPacket will be set. For example, if the character 'W' is sent from the operator, the type of the TPacket will be set as COMMAND_FORWARD, which will instruct the Arduino to turn the motors forward.
5. The TPacket is then sent to the Arduino through uartSendPacket().

Sending information back to the operator

1. Information from the Arduino is received by uartReceiveThread() which deserialises the buffer into a packet to be handled.
2. If the packet is complete, handleUARTPacket() is invoked which determines the type of the packet and calls handleResponse(), handleErrorResponse(), or handleMessage() accordingly.

3. handleResponse() can return 3 types of responses
    a. Forwards the Arduino's message to the operator via handleMessage()
    b. Sends the status of the Wheel Encoder Information and Odometry via handleStatus()
    c. Sends the frequency of RGB and the colour identified via handleColour()

Operator Interactions

The high-level algorithm to view the generated map and interact with Alex is as follows:

Initialisation

1. The operator opens a terminal on the laptop and runs the roscore command to allow all ROS nodes to communicate.
2. The operator opens another terminal to run the alexlaunch.sh bash script to initialise the SLAM program to map the environment. The visualisation node (rviz) also starts to render the environment map.
3. The operator opens one last terminal to run the client code compiled from tls-alex-client.cpp. The operator can then enter command through this User Interface.

Sending commands to Raspberry Pi

1. Commands from the operator are read by writerThread() when the operator enters a character.
2. A buffer with type NET_COMMAND_PACKET is created and the parameters filled in based on the character input:
    a. 3 sets of movement input are accepted: "WASD" for fine control movements, "IJKL" for travelling longer distance and wider turning and 'TFGH' for movements with input parameters. Each set of movements input correspond to "<Forward><Turn Left><Reverse><Turn Right>"
    b. 'Z' tells the Raspberry Pi to stop all movement.
    c. 'X' tells the Raspberry Pi to send Odometry information while 'C' clears and resets this information.
    d. 'V' tells the Raspberry Pi to send back the RGB frequency of the object identified.
3. The buffer is sent to the Raspberry Pi by sendData() using sslWrite().

Receiving information from Raspberry Pi

1. Packets from the Raspberry Pi are read by readerThread() via sslRead().
2. The handleNetwork() function is invoked which determines the type of packet received and calls handleError(), handleStatus(), handleColour(), handleMessage() and handleCommand() accordingly
3. handleStatus() prints out the Odometry data for the operator in the User Interface.
4. handleColour() prints out the RGB frequency and the colour identified in the User Interface.

Section 7 Lessons Learnt - Conclusion

1. TCS3200 Color Sensor Range

Our team procured the TCS3200 colour sensor and soon realised that the hardware has its functional limitations. The sensory practical purposes as shown online does work to a certain extent as the range of colour detection varies from 5cm to 1cm. However, the colour sensor is better suited for average lighting in relatively dark places. What we failed to take note of is the lighting in the lab and how the intense light affected the readings taken by the colour sensor. As such, we adapted and attached a black coloured paper on top of the sensor to reduce the white light interfering with the readings. In addition, since the optimal distance for an accurate colour reading is between 1-5cm, we did not take into account how close the Alex robot had to be for an accurate reading. This caused a little uncertainty towards controlling the robot as we ran the risk of hitting the object. Though our group's approach of creating a set of fine control movements worked out in the end, we feel that a better alternative would have been to utilise an ultrasonic sensor that will stop the robot 3cm before the object to prevent a collision but still get an accurate reading.

2. Using Version Control

Throughout the project, there were several revisions done to the codebase, not just to Alex.ino but to the TLS Client and Server code as well. However, one great mistake was how we stored most of the codebase locally on the Arduino Uno. As the code was not stored on GitHub, our members were not able to access the codebase at home. Being able to consistently use version control tools like GitHub would have allowed us to checkout older revisions when something goes wrong as well as comparing newer versions of code to older versions, which would have made the process of debugging a lot easier. Hence, our team's productivity could have been higher if we consistently use Git and GitHub.

References

[1] "HUMAN COST OF NATURAL DISASTERS THE," *Centre for Research on Epidemiology of Disasters,* 2015. Accessed: Apr. 16, 2021. [Online]. Available: https://www.preventionweb.net/files/42895_cerdthehumancostofdisastersglobalpe.pdf .

[2] D. Vukovic, "Why the First 72 Hours After a Disaster Are Critical," 2020. https://www.primalsurvivor.net/why-the-first-72-hours-after-a-disaster-are-critical/ (accessed Apr. 16, 2021).

[3] "Search and Rescue Robots Market - Growth, Trends, COVID-19 Impact, and Forecasts (2021 - 2026)," *Mordor Intelligence*, 2021. https://www.mordorintelligence.com/industry-reports/search-and-rescue-robots-market (accessed Apr. 16, 2021).

[4] "The humanoid robot WALK-MAN for supporting emergency response teams | EurekAlert! Science News," *ISTITUTO ITALIANO DI TECNOLOGIA*, 2018. https://www.eurekalert.org/pub_releases/2018-02/iidt-hrw022118.php (accessed Apr. 16, 2021).

[5] "Colossus - ROBOTS: Your Guide to the World of Robotics," *Shark Robotics*, 2017. https://robots.ieee.org/robots/colossus/ (accessed Apr. 16, 2021).