CG2111A Engineering Principle and Practice II

Semester 2 2023/2024

"Alex to the Rescue"

Design Report

Team: B04-5A

Name	Student #	Sub-Team	Role
Brandon Kang	A0272866N	Software	Software Lead
Roderick Kong Zhang	A0286550Y	Software	Software Engineer
Sean Ter Yong Jun	A0272977J	Hardware	Hardware Lead
Pavithra Srinivasan	A0282405J	Hardware	Hardware Engineer
Ser Jun Wei Kingsley	A0271886M	Hardware	Hardware Engineer

Table of Contents

Section 1 | System Functionalities

Alex is a robotic vehicle with search-and-rescue functionalities. Alex is teleoperated so that operators are not in contact with the disaster site. The Raspberry Pi installed on Alex allows for communication between the operator's laptop and the Arduino.

Section 1.1 | Mapping

Alex is able to map out its surroundings while simultaneously tracking its current location (Simultaneous Localisation and Mapping (SLAM)). The LiDAR sensor, in conjunction with Hector SLAM, allows Alex to detect obstacles and objects while identifying its position on the map.

Section 1.2 | Movement

Alex is able to traverse a rescue site by using its wheels to move forwards and backwards. It is also able to turn left and right, at a specified turning angle. Commands are sent to Alex to instruct it to travel for a specified distance at a certain speed, or turn at a certain angle.

Section 1.3 | Colour Sensing

Moreover, Alex is able to identify and differentiate victims, healthy or injured, from obstacles. This is achieved through the use of an ultrasonic sensor and a colour sensor.

The ultrasonic sensor detects the presence of an obstacle or victim by determining the distance between Alex and the corresponding object. The colour sensor is connected to an Arduino which computes and identifies the colour of the detected object. If the colour detected is red or green, an injured or healthy trapped victim has been identified respectively. If the detected colour is not red or green, Alex will identify the object as an obstacle instead of a victim.

Section 1.4 | Additional Functionalities

Additionally, Alex has obstacle-avoidance capabilities, allowing it to detect and dodge obstacles automatically, while the teleoperator controls it remotely to traverse the rescue site. One ultrasonic sensor is placed on each side of Alex (left and right) to detect and avoid nearby obstacles.

Alex also features an in-built buzzer, used to alert nearby pedestrians when a casualty has been found. The buzzer will only be triggered when a victim has been identified (it will not trigger when Alex comes across an obstacle).

Section 2 | Review of the State of the Art

Section 2.1 | Boston Dynamics – Atlas

The Atlas is a humanoid robot designed to assist emergency management teams in handling natural and man-made catastrophes. It can perform tasks such as flipping switches, shutting off valves, opening doors, and running power equipment. These functions are possible through the use of components such as lightweight hydraulics and 3D-printed appendages. It also uses LiDAR and stereo vision to effectively navigate through rough terrains.

Strengths:

- 3D printing is used to manufacture components to save weight and space, resulting in a compact robot with a high strength-to-weight ratio
- Advanced control algorithms enable the robot to plan and perform actions based on the environment it analyses

Weaknesses:

- Too large to fit in tight spaces
- Ineffective in other terrains, such as water and air
- Uses a lot of energy

Section 2.2 | Pilant Energy – Velox

The Velox is a robot equipped with fins that give it amphibious capabilities, allowing for rescue missions on both land and water. It is able to reverse and make quick turns instantaneously, allowing it to rapidly manoeuvre and look around and between objects.

Strengths:

- Able to traverse through a variety of terrains, including land and water
- Lower environmental impact compared to other robots with spinning propeller-thrusters
- Resistant to entanglement in plants and other aquatic debris

Weaknesses:

- Slow travelling speed
- Ineffective in overcoming large solid obstacles

Section 3 | System Architecture

Alex comprises multiple devices working and communicating together to complete its search-and-rescue mission. *Figure 1* illustrates Alex's system architecture, detailing how its various components are connected, and how they communicate with one another.

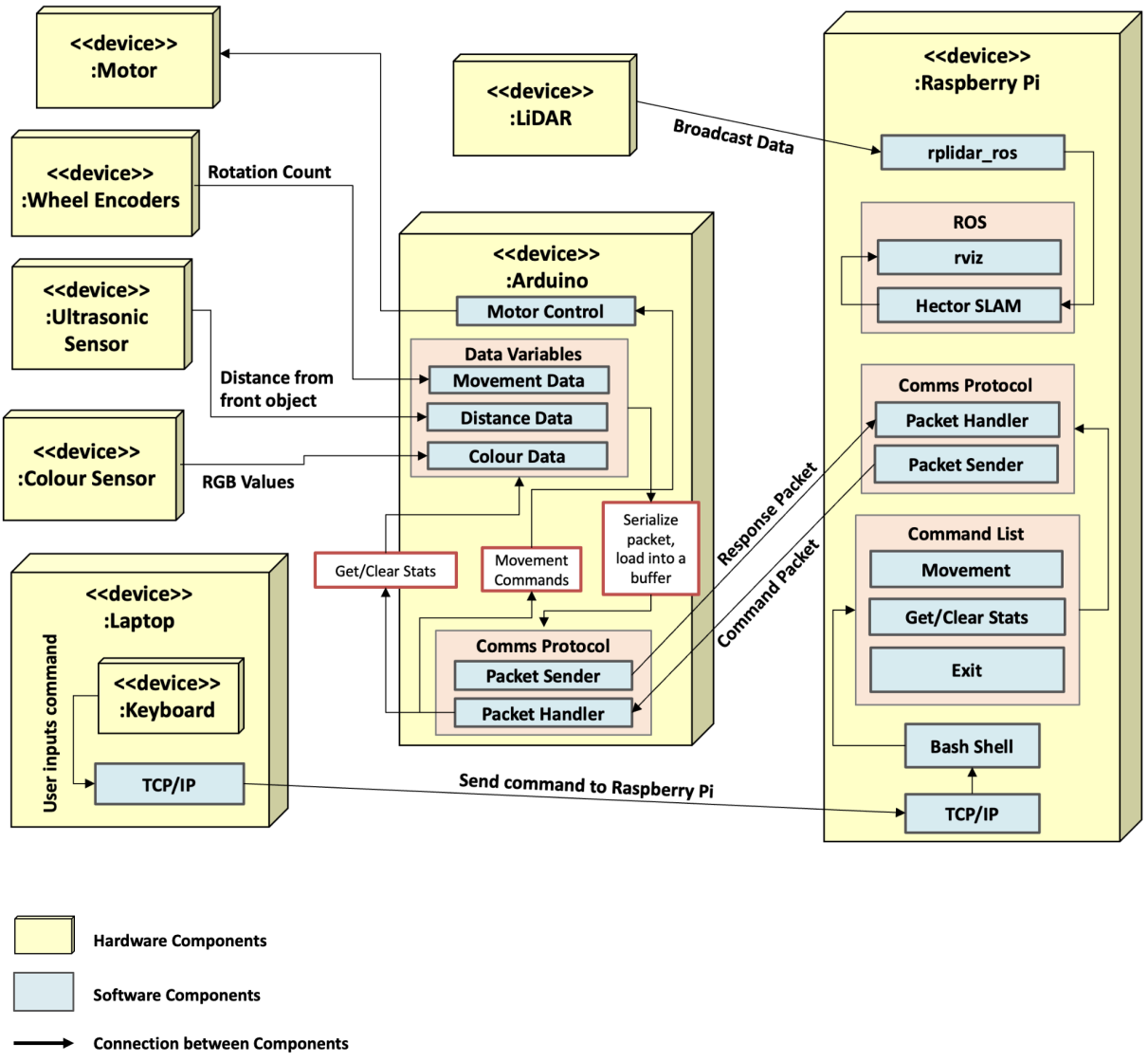
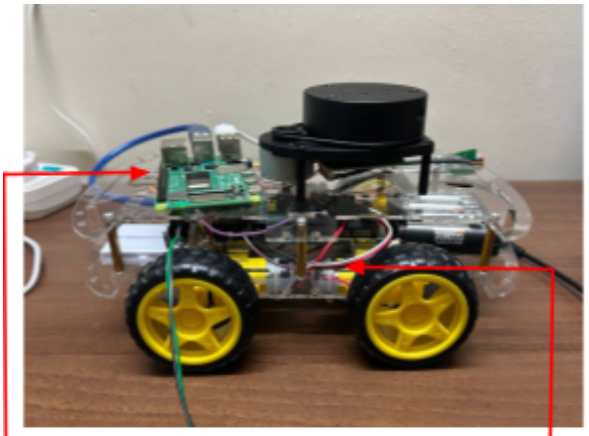


Figure 1. System Architecture Design

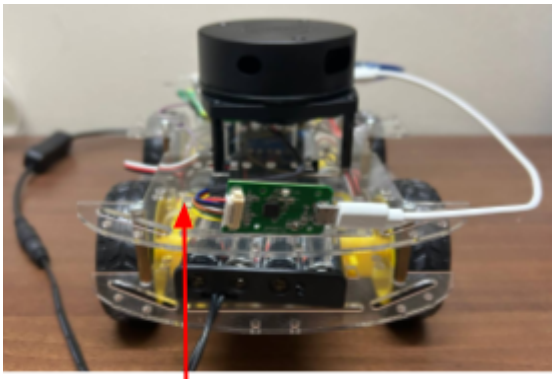
Section 4 | Component Design

Section 4.1 | Hardware Implementation

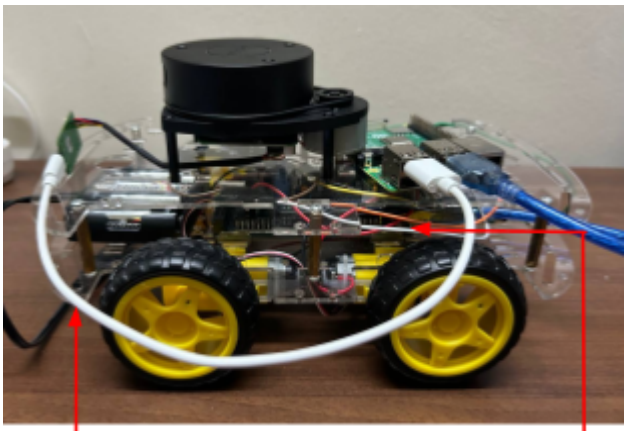


Raspberry Pi attached to top front layer

1 Ultrasonic sensor will be attached on this side between 2 layers

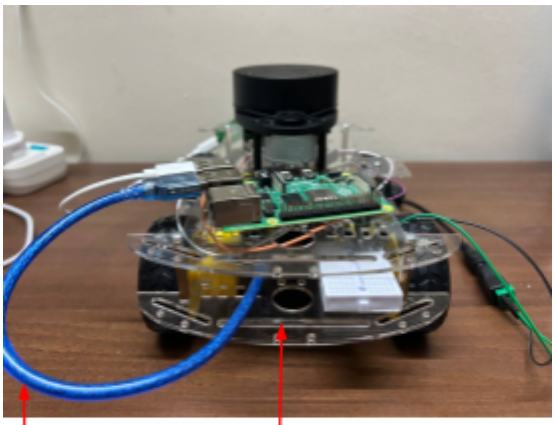


Powerbank will be placed behind the LiDAR



LiDAR connected to Rpi.

Wires will be bundled up to avoid collision with the motors. 1 Ultrasonic sensor will be attached on this side between 2 layers



Colour sensor and 1 Ultrasonic sensor will be attached next to each other here at the front between the 2 layers

Rpi connected to Arduino in the first layer. Buzzer will be connected to arduino and placed between 2 layers

Section 4.2 | Software Implementation

High Level Algorithm:

1. **Initialisation**
2. **Receive command**
3. **Execute command**
4. **Initialise LiDAR**
5. **Repeat Steps 2-3 until mission is over**
6. **Termination**

Step 1. Initialisation

Step 1.1. Establishing connection between Laptop and RPi

The communication between the laptop and RPi happens over the cloud when the *tls-alex-server* program is run on the RPi and the *tls-alex-client* program is run on the laptop. The client (Laptop) and server (RPi) communicate with each other securely through a TLS connection. To initialise the connection, a TLS handshake occurs where the client authenticates the server's SSL certificate with the certificate authority that issued it. After the handshake is complete, the laptop will be able to send commands to the RPi remotely and securely.

Step 1.2. Start communication between RPi and Arduino

The RPi initialises communication with the Arduino by creating a TPacket named helloPacket and setting the packetType to PACKET_TYPE_HELLO, which is then sent over to the Arduino.

```
typedef struct
{
	char packetType;
	char command;
	char dummy[2]; // Padding to make up 4 bytes
	char data[MAX_STR_LEN]; // String data
	uint32_t params[16];
} TPacket;
```

Figure 2. TPacket Struct

The RPi now waits for a command from the teleoperator.

Step 2. Receive command

Step 2.1. RPi serializes and sends command packet to Arduino

The teleoperator can now send commands to Alex to either move forwards, move backwards, turn left, turn right, get stats, clear stats, or quit. The RPi creates a commandPacket and sets the packetType to PACKET_TYPE_COMMAND. Depending on the input by the teleoperator, the RPi will store its respective command in the command field of the packet, before sending the packet over to the Arduino.

Step 2.2. Arduino receives and handles packet

The Arduino deserializes the data in the packet and checks the magic number and checksum for any errors that may have occurred during transmission.

Case 2.2.1. Magic number received is wrong

Arduino will create a badPacket with PACKET_TYPE_ERROR in the packetType field and RESP_BAD_PACKET in the command field. This packet is then sent back as a response to the RPi, signalling that an error with the magic number has occurred.

```
void sendBadPacket()
{
	// Tell the Pi that it sent us a packet with a bad
	// magic number.
	TPacket badPacket;
	badPacket.packetType = PACKET_TYPE_ERROR;
	badPacket.command = RESP_BAD_PACKET;
	sendResponse(&badPacket);
}
```

Figure 3. Send badPacket

Case 2.2.2. Checksum received is wrong

Arduino will create a badChecksum packet with PACKET_TYPE_ERROR in the packetType field and RESP_BAD_CHECKSUM in the command field. This packet is then sent back as a response to the RPi, signalling that an error with the checksum had occurred.

```
void sendBadChecksum()
{
	// Tell the Pi that it sent us a packet with a bad
	// checksum.
	TPacket badChecksum;
	badChecksum.packetType = PACKET_TYPE_ERROR;
	badChecksum.command = RESP_BAD_CHECKSUM;
	sendResponse(&badChecksum);
}
```

Figure 4. Send badChecksum

Case 2.2.3. Invalid command

Arduino will create a badCommand packet with PACKET_TYPE_ERROR in the packetType field and RESP_BAD_COMMAND in the command field. This packet is then sent back as a response to the RPi, signalling that an invalid command has been given.

```
void sendBadCommand()
{
	// Tell the Pi that we don't understand its
	// command sent to us.
	TPacket badCommand;
	badCommand.packetType=PACKET_TYPE_ERROR;
	badCommand.command=RESP_BAD_COMMAND;
	sendResponse(&badCommand);
}
```

Figure 5. Send badCommand

Case 2.2.4. No errors occurred

Arduino will create an okPacket to be sent to the RPi to acknowledge that the command is received before executing it. The okPacket will have PACKET_TYPE_RESPONSE in the packetType field and RESP_OK in the command field.

```
void sendOK()
{
	TPacket okPacket;
	okPacket.packetType = PACKET_TYPE_RESPONSE;
	okPacket.command = RESP_OK;
	sendResponse(&okPacket);
}
```

Figure 6. Send okPacket

Step 3. Execute command

The Arduino checks the command field of the commandPacket and executes the appropriate command accordingly.

Case 3.1. Movement Command

Arduino reads the direction in the command field of commandPacket, and gets the distance to move in cm, or angle to turn in degrees, as well as the speed to do so, in the params field of commandPacket. It then instructs the motors to move according to the commands specified. *Figure 7* shows an example for handling the command for moving forward, in the form of a switch case.

```
case COMMAND_FORWARD:
	sendOK();
	forward((double) command->params[0], (float) command->params[1]);
	break;
```

Figure 7. Handling command for moving forward

The forward() function stores the distance to be moved in deltaDist, and calculates the new total distance moved after the command, by adding deltaDist and forwardDist. forwardDist is the distance travelled by Alex and is calculated using:

```
forwardDist = (unsigned long) ((((float) leftForwardTicks) / (float)
COUNTS_PER_REV) * WHEEL_CIRC);
```

where leftForwardTicks is incremented when the ISR is triggered from the wheel encoders. The move() function will spin the wheels according to the speed and direction specified. The spinning of the wheels triggers the ISR for INT2_vect and INT3_vect; the respective variables to store the number of ticks will be incremented accordingly.

```
void forward(float dist, float speed)
{
	if (dist > 0) {
		deltaDist = dist;
	}
	else {
		deltaDist = 9999999;
	}
	newDist = forwardDist + deltaDist;
	dir = (TDirection) FORWARD;
	move(speed, FORWARD);
}
```

Figure 8. Function for moving forward

The motors will continue running until forwardDist is no longer smaller than newDist, meaning that Alex has completed travelling the intended distance. The stop() function is then called to stop Alex.

Case 3.2. Get Stats Command

Arduino creates a statusPacket with PACKET_TYPE_RESPONSE in the packetType field and RESP_STATUS in the command field. It enters all the necessary data from the respective variables into the params field, serializes it, and sends the packet to the RPi.

```
void sendStatus()
{
	TPacket statusPacket;
	statusPacket.packetType = PACKET_TYPE_RESPONSE;
	statusPacket.command = RESP_STATUS;
	statusPacket.params[0] = leftForwardTicks;
	statusPacket.params[1] = rightForwardTicks;
	statusPacket.params[2] = leftReverseTicks;
	statusPacket.params[3] = rightReverseTicks;
	statusPacket.params[4] = leftForwardTicksTurns;
	statusPacket.params[5] = rightForwardTicksTurns;
	statusPacket.params[6] = leftReverseTicksTurns;
	statusPacket.params[7] = rightReverseTicksTurns;
	statusPacket.params[8] = forwardDist;
	statusPacket.params[9] = reverseDist;
	sendResponse(&statusPacket);
}
```

Figure 9. Handling Get Stats Command

The RPi receives and deserializes the packet, then handles the packet, and prints out the data to be viewed by the teleoperator.

```
void handleStatus(TPacket *packet)
{
	printf("\n ------- ALEX STATUS REPORT ------- \n\n");
	printf("Left Forward Ticks:\t\t%d\n", packet->params[0]);
	printf("Right Forward Ticks:\t\t%d\n", packet->params[1]);
	printf("Left Reverse Ticks:\t\t%d\n", packet->params[2]);
	printf("Right Reverse Ticks:\t\t%d\n", packet->params[3]);
	printf("Left Forward Ticks Turns:\t%d\n", packet->params[4]);
	printf("Right Forward Ticks Turns:\t%d\n", packet->params[5]);
	printf("Left Reverse Ticks Turns:\t%d\n", packet->params[6]);
	printf("Right Reverse Ticks Turns:\t%d\n", packet->params[7]);
	printf("Forward Distance:\t\t%d\n", packet->params[8]);
	printf("Reverse Distance:\t\t%d\n", packet->params[9]);
	printf("\n---------------------------------------\n\n");
}
```

Figure 10. Handling response packet for getting stats

Alternatively, the Clear Stats command will set all the above-mentioned variables to 0.

The stats for obtaining the distance from the ultrasonic sensor, as well as the colour of the detected object, will be implemented in the future.

Case 3.3. Exit Command

If an exit command is issued by the RPi, proceed to **Step 6**.

Step 4. Initialise LiDAR

Firstly, an RPLidar Node is used to broadcast the LiDAR readings obtained. A SLAM node is then used to obtain the LiDAR readings for environment mapping. The visualisation node, *rviz*, is used to render the environment map. This map is used to aid the teleoperator for navigation in **Steps 2 and 3**.

Step 5. Repeat Steps 2-3 until mission is over

After executing the command by the teleoperator, the RPi now waits for the next command to be given, where **Step 2** and **Step 3** will be repeated again.

Step 6. Termination

The teleoperator enters the command to quit the program, and the RPi terminates communication with the Arduino.

Figure 11 shows a simplified flowchart of the High Level Algorithm from the Arduino's perspective.

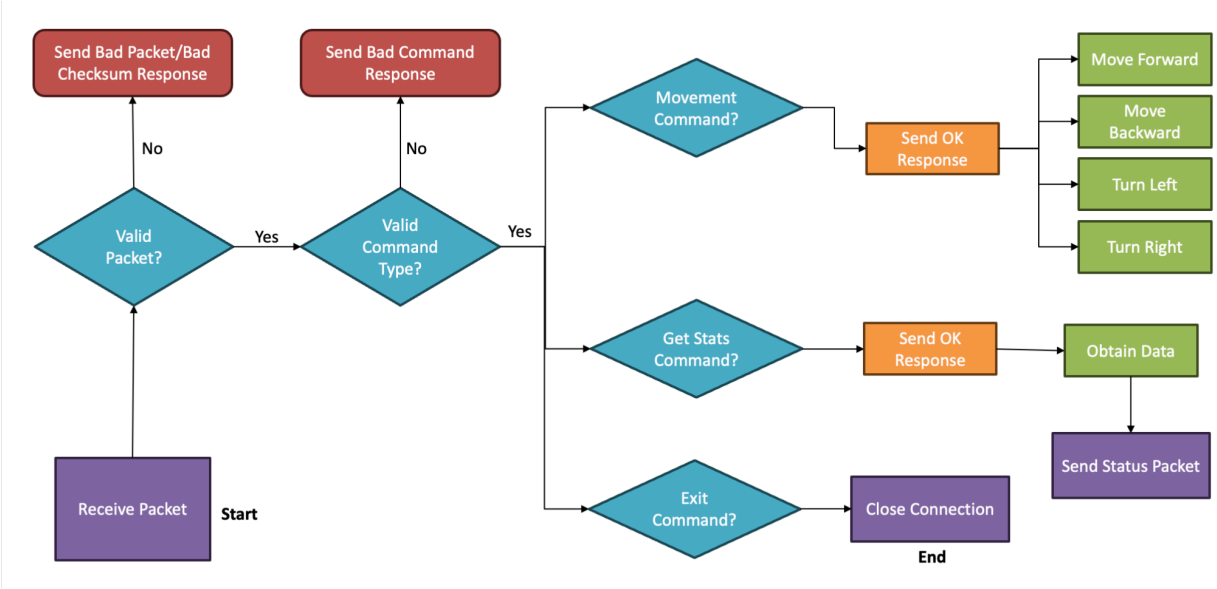


Figure 11. Flowchart of High Level Algorithm on the Arduino

Section 5 | Project Plan

Week 7	1. Learn and set up USART communication 2. Test LiDAR sensor 3. Visualise data from LiDAR sensor
Week 8	1. Assemble Alex 2. Test cross-platform communications (between Raspberry Pi and Arduino) 3. Set up and test wheel encoders 4. Set up Alex's motor control routines
Week 9	1. Implement tracking of current direction, forward and backward distances 2. Control Alex remotely via TCP/IP Protocol 3. Implement movement commands to: a. Turn Alex left or right at a specified angle b. Move Alex forward or backwards for a specified distance 4. Write TLS programs to create a server to control Alex over the cloud 5. Secure Alex components
Week 10	1. Debugging 2. Set up ROS and SLAM
Week 11	1. Implement colour sensor to detect colours of objects to search for victims 2. Implement ultrasonic sensor to measure the distance of an object from the Alex's front
Week 12	1. Calibrate motors for Alex to: a. Move in a straight line b. Turn accurately (as specified in the given command) 2. Calibrate colour sensor
Week 13	1. Trial Run 2. Final Demonstration

References

Boston Dynamics. (n.d.). Atlas. Retrieved from https://bostondynamics.com/atlas/

Pliant Energy. (n.d.). Robotics. Retrieved from https://www.pliantenergy.com/robotics

Wevolver. (n.d.). Atlas Robot. Retrieved from https://www.wevolver.com/specs/atlas.robot