

**CG1112 Engineering Principle and Practice**

Semester 2 2018/2019

**“Alex to the Rescue”**

**Final Report**

**Team: 02-01-01**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Student # | Sub-Team | Main Role |
| HONG WEI YANG | A0182619X | Software | Lidar Implementation |
| JIANG HONG QUAN | A0190306M | Hardware | Circuit design |
| KHAIRUL IMAN NASRI B AZHARIE | A0182359W | Software | Robot Movement |
| LEOW YONG KANG ANDREW | A0188376L | Hardware | Robot Calibration |
| NG WEE KIAT | A0182496R | Software | Sensors Implementation |

**Section 1: Introduction**

Every year, natural disasters kill around 90 000 people and affect close to 160 million people worldwide [1]. In natural disasters, the goal of a search and rescue operation is to rescue the largest number of people in the shortest time, while minimizing risk to the rescuers [2]. Therefore, information regarding the site of the disaster and the location of the casualties are extremely vital to the rescue operation. With this information, the rescuers would be able to make informed decisions that would greatly reduce the time required for search and rescue.

Our group has developed a robotic vehicle with search and locate functions, named Alex. Alex has been tested under a simulated environment for fast deployment, navigation and object identification capabilities. Primarily, the operator controls Alex’s movements remotely via a laptop while Alex constantly scans the environment. Information collected by Alex is then relayed back to the laptop, where it is processed into a map. Alex is equipped with colour recognition technology, which could be utilised by the user to identify casualties.

Alex provides a fast, safe and efficient way to gather real-time information of the disaster site and identify casualties. Therefore, it will greatly improve the efficiency of a search and rescue operation.

**Section 2: Review of State of the Art**

Over the years, robotic advancements have opened new possibilities in the field of search and rescue operations. In this section, we discuss the development and examples of robots in this field. Ultimately, we evaluate the strengths and weaknesses of each example which serve as learning points to be incorporated into our project, Alex.

**ROBOCUE (Rescue Robot)**

ROBOCUE is a search-and-rescue robot used by the Tokyo Fire Department since 2009 [3]. It is used for searching and rescuing victims in environments too hazardous for rescuers to enter.

Dimensions: 1.90m x 1.20m x 1.60m

Weight: 1500kg

Wireless control distance: 100 m

ROBOCUE is able to pull a casualty inside its body via a conveyor belt. After which it will transport the casualty to a safe location [4].

**Strengths**:

* Active scope camera could slither through crevices to locate casualties (Fig. 1)
* Able to provide immediate assistance to casualty upon discovery
* Reduces manpower needed for casualty extraction

**Weaknesses**:

* Unable to reach casualties trapped behind obstacles due to large size
* A complicated system which is challenging to operate

*Fig. 1 Active scope camera Fig. 2 ROBOCUE extracting a casualty [6]*

*searching for casualties [5]*

**Robotic Snakes**

Robotic snakes developed by the Carnegie Mellon University were used for search and locate purposes during the 2017 Mexican earthquake [7].



*Fig. 3: Robotic Snake [8]*

The robotic snake utilises a series of movement patterns called gaits for its mobility. When combined, these gaits enable the robotic snake to complete complex movements such as climbing, rolling, or even swimming. This makes the robotic snake especially useful for accessing tight and irregularly shaped places created by debris, where it then captures information using its camera located on its head [9].

**Strengths:**

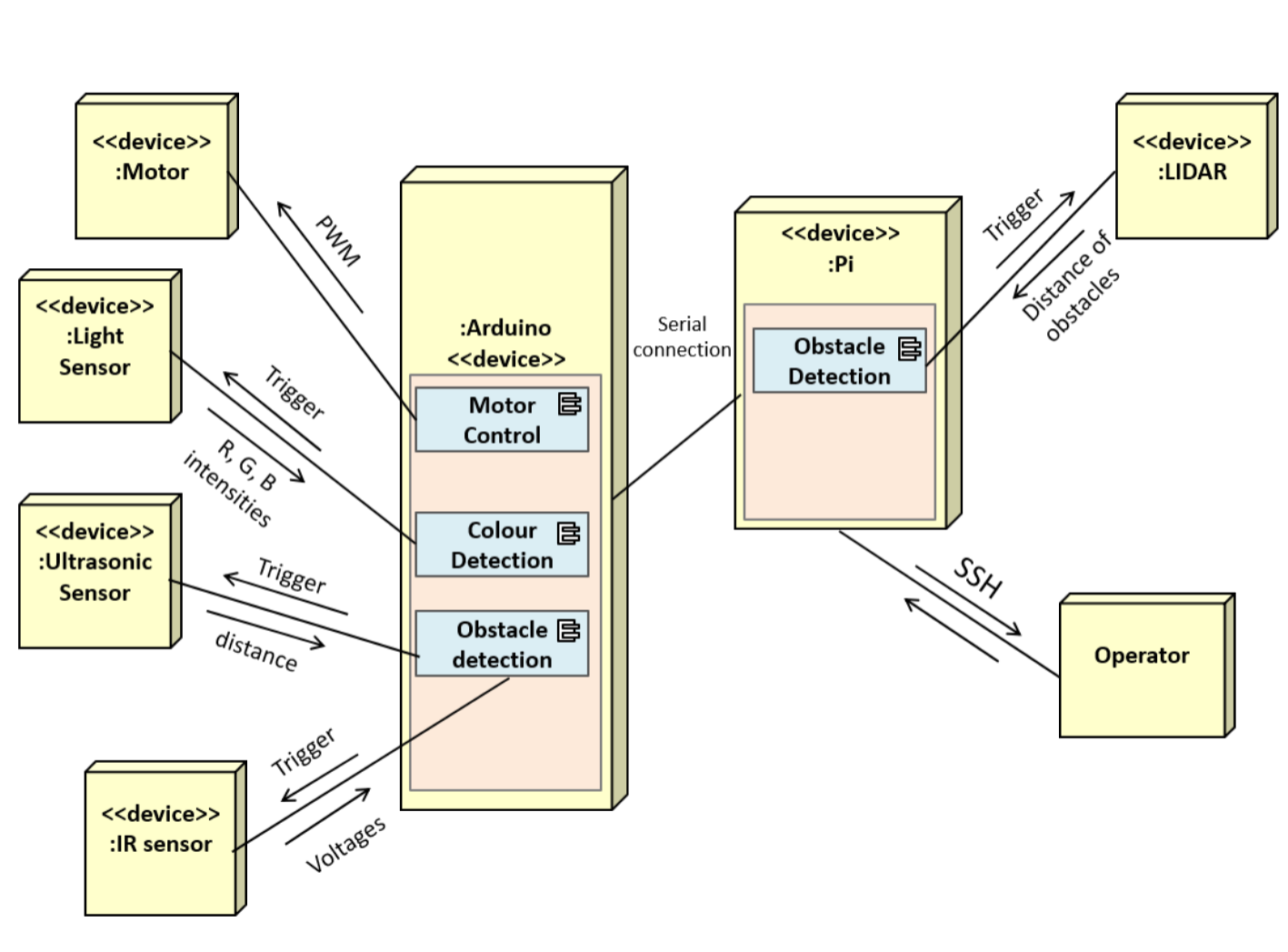
* Able to fit into tight spaces
* Able to cross uneven terrain, and even climb poles
* Can be controlled wirelessly

**Weaknesses:**

* Hard to operate, requires 4 operators for optimal performance
* Information gathering is inefficient since only one camera is installed at the front
* Slow movement

**Section 3: System Architecture**

This section aims to explain the general architecture of Alex. The main components of Alex consist of the various sensors, the 2 Microcontroller Units (MCUs) and motors. We will explore the relationship between these components (Fig. 4) and the way they cooperate with one another to accomplish the mission objectives.

****

*Fig. 4: Alex’s system architecture*

The 2 MCUs, namely Arduino Uno and Raspberry Pi (commonly referred to as Pi), are connected serially. The Pi is analogous to the human brain which oversees the entire operations in Alex whereas the Arduino Uno is analogous to the human spine which coordinates other functions in Alex. Both MCUs are engaged in constant serial communication, working together to carry out Alex’s search and locate functions.

**Raspberry Pi**

The Pi is accessed remotely by the operator’s laptop using Secure Shell (SSH) Protocol. This function enables the operator to remotely send command packets to Pi, thus supporting teleoperation.

The Pi is connected directly to the Light Detection and Ranging (LIDAR) unit which scans and generates a live data plot for Pi to process using the Simultaneous Localization and Mapping (SLAM) algorithm. The sensor data would be published to a Robot Operating System (ROS) node, an executable file, which treats the master as the Pi. The data would then be processed to generate a 2D map. Further details regarding the ROS network setup is further elaborated in Section 6 regarding Software Design.

**Arduino**

The Arduino is connected to the motors, the colour detection sensor, the ultrasonic sensor, and the infrared sensors. Both the movement and the colour detection commands are integrated with the Master Control Program (MCP), the program which controls Alex. Commands from the operator are transmitted in the form of packets between the Pi and the Arduino.

When the operator issues a movement command from the Pi, a command packet would be sent via serial communication to the Arduino which controls the motors. Once the Arduino finishes executing the command packets sent by Pi, it would send an acknowledgement packet to signal that it is ready for the next command.

Colour detection is implemented using a Light Dependent Resistor (LDR) sensor connected to Arduino. Upon command, the LDR sensors perform colour detection of the obstacles in proximity before returning a message to Pi through Arduino, indicating the colour detected.

Ultrasonic sensor and Infrared (IR) sensors are connected to Arduino to aid in obstacle detection. Both sensors measure the distance between surrounding obstacles and Alex. They are pre-programmed within Arduino to constantly detect obstacles that fall within a threshold distance. Once obstacles are deemed to be too close, Arduino autonomously stops Alex to avoid collision and simultaneously sends an alert message across the serial connection to Pi to warn the operator.

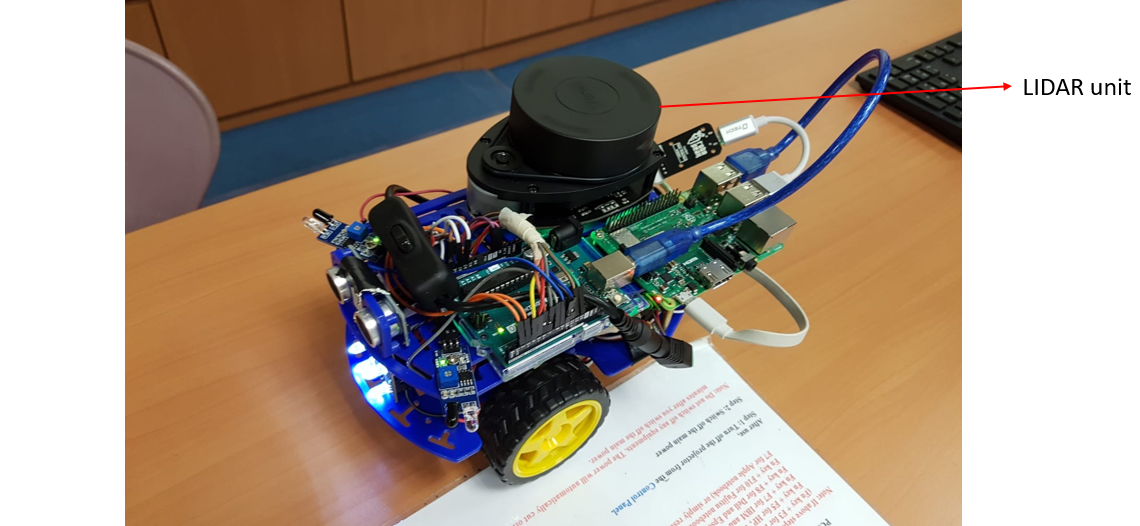
The power consumption of both Pi and Arduino can be reduced by switching off the components which are not in use. As such, the ethernet and the HDMI port on the Pi as well as the ADC module and the external Watchdog timer on the Arduino have been switched off. For obstacle detection, interrupts are utilized instead of polling to reduce the amount of constant activity on the CPU and save power.

**Section 4: Hardware Design**

This section describes the hardware design for Alex. The workings behind the respective hardware components and the reasons for incorporating them into Alex would be explained. Furthermore, the specific placements of the hardware components would be depicted along with explanations on the rationale behind such arrangements.

**Sensors:**

**LIDAR**



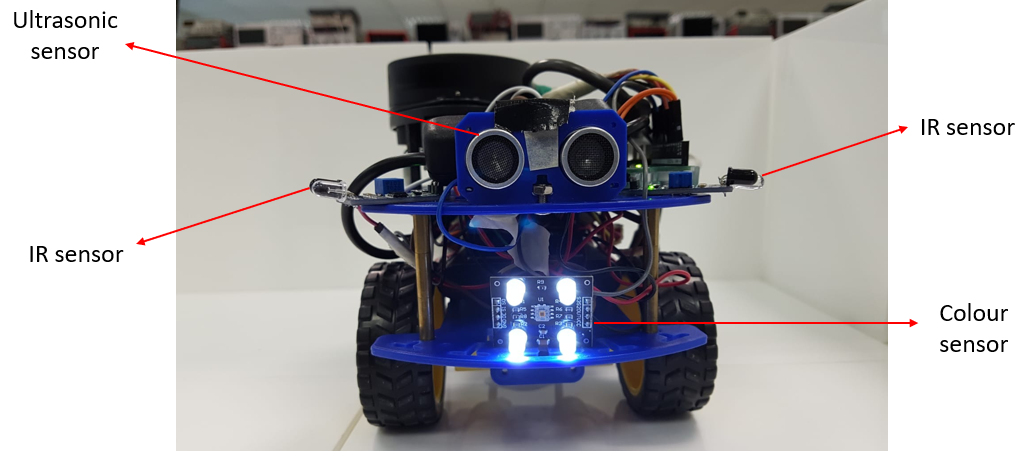
*Fig. 5: LIDAR unit mounted on top of Alex*

The LIDAR unit (Fig. 5) emits light pulses which are reflected off surrounding obstacles and back into its detection unit. With a constantly rotating mount, the LIDAR is able to scan the surroundings and collect raw data of the environment. The collected readings would be transmitted to Raspberry Pi and used in the generation of a 2D map.

The LIDAR is instrumental in mapping out the environment. It is mounted at the top of Alex to ensure that the LIDAR has a clear space to perform its scanning without obstruction from other components.

However, the LIDAR is unable to accurately detect obstacles within 15cm of its radius. Thus, an ultrasonic sensor and 2 IR sensors are used to complement the LIDAR.

**Ultrasonic and Infrared(IR) Sensors**



*Fig. 6: Position of the ultrasonic, colour, and IR sensors*

The ultrasonic sensor (Fig. 6) emits ultrasound which is reflected off obstacles and back into its receiver. The time elapsed between the transmission and reception is used to compute the distance of obstacles ahead by the Arduino Uno. If at any time obstacles ahead of it are too close, it will send a message to the teleoperator that the front is too close to something. This sensor complements the LIDAR by detecting incoming obstacles in front of Alex that are too close but not detected by the LIDAR.

IR sensors (Fig. 6) are placed at the sides of Alex. They emit infrared light which is reflected off the obstacles, and back into its receiver. The output is generated in the form of voltages which corresponds to distance when interpreted with Arduino Uno. Like the ultrasonic sensor, the 2 IR sensors complement the LIDAR by detecting obstacles on the sides of Alex that are too near but not detected by the LIDAR.

Additionally, since the sides are more vulnerable to hitting the walls during motion due to protruding wires, the IR sensors work on a pin change interrupt, which triggers faster and more efficiently than the polling of the ultrasonic sensor. When the interrupt is called, Alex will stop, and a message will be sent to the teleoperator that the sides are too close.

Once Alex has moved through an environment, the areas covered would have been considered cleared of obstacles. Since Alex’s main movements are moving forward and turning on the spot, there is no need for obstacles sensor to be placed in the rear.

**Colour sensor**

The model used is TCS230 (Fig. 6). The sensor emits light from its 4 LEDs which is reflected off the obstacles and received by the central photodiode. Readings from the photodiodes are converted into a square wave signal with frequency directly proportional to the light intensity. The Arduino interprets the output square wave and identifies the colours accordingly. The colour sensor is placed in front of Alex, so the operator could easily aim it at the object.

**Circuitry:**

**MCUs and LIDAR**

The 2 MCUs namely Arduino Uno and Raspberry Pi are placed beside each other on the top layer alongside LIDAR. Since both MCUs need to be connected serially by a USB cable and the LIDAR is connected to Pi, they are placed on the top layer.

**Others**

The remaining circuit consists of the batteries, power bank and motor driver chip with its wire connections (Fig. 7). They are positioned within the middle layer of Alex. This arrangement allows most of the wired connections to be cable-managed effectively and stored in the middle layer. There would also be less bulk of wires on the top layer that could potentially obstruct the LIDAR scanning process.



*Fig. 7: The circuitry at the back of Alex*

**Section 5: Firmware Design**

As mentioned above, the Arduino is analogous to the spine of Alex, coordinating its various functions and the Pi is analogous to the brain which manages the entire Alex. Therefore, algorithm on the Arduino and communication protocol between both MCUs must be thoughtfully implemented. This section explains the thought process and describes the firmware design, comprising mainly of the high-level algorithm on the Arduino Uno and the communication protocol.

**Algorithm of Arduino**

**Receival of Command Packets**

The Pi sends commands over to Arduino in the form of packets via serial communication. Upon receiving the packet, the validity of the packet received would be checked with the use of checksum and magic number. Their workings would be explained in the communication protocol below.

If the packet received is valid, the command packet’s parameters would be read by the “handleCommand()” function, allowing Arduino to control the motor to move in the direction and speed as prescribed by the received packet. The directions to move are implemented within the functions namely, FORWARD (*parameters*), REVERSE (*parameters*), LEFT (*parameters*) and RIGHT (*parameters*). The parameters taken in by these functions are the distance to move or the angle to turn coupled with the speed of movement.

Once the command is executed to completion, a “commandOK” packet will be sent back to the Pi, informing the operator to input the following command.

**Calibration of Motors**

Within the FORWARD (*parameters*) and REVERSE (*parameters*) functions, PWM values are written to both motors to enable Alex to move in their respective directions. As no two motors are the same, calibration is required for Alex to move as straight as possible. Hence, a trial and error experimentation are performed to find the best pair of analogue PWM values to write to each motor for the respective directions.

**Mechanism behind Movement Functions**

The wheel encoder sensors are used to determine the distance that Alex is supposed to travel. The Arduino has a variable which keeps count of the number of ticks during the turning of the motors. Whenever the sensors move one-quarter of a revolution, the interrupt for the left or right wheel is triggered accordingly, increasing the ticks counts according to the direction of travel. These ticks help to calculate the actual distance Alex travels through an equation involving the wheel circumference and the earlier distance parameter taken in. Upon finding the actual distance required (deltaDist), the distance will then be measured using the ticks of the wheel encoder as the wheels start to move. In the void loop () function, when the ticks of the wheel encoder exceed the required distance to move, the stop () function is called, where the motors will have the analogue value 0 written to them, stopping any movement.

The turning function also takes in two parameters input by the operator, the angle to turn and the power supplied to the motors. The ticks are calculated based on the turn radius of Alex using the function computeDeltaTicks(). The calculations are based on the Alex’s circumference, the wheel circumference and number of encoder count per wheel revolution. After the required ticks per angle of movement are found, the same logic is applied in the Arduino void loop function, where any movement of the motors increases the respective ticks. Once the ticks exceeded the total ticks required for the movement, the stop () function is called to stop the motors. By calibrating the computeDeltaTicks function, as well as adjusting the PWM values of the speed written to the motors, we can get the turns as accurate to the degree the teleoperator specifies in the command.

With these functions, the teleoperator could navigate Alex according to the mapping generated by Pi using the LIDAR. The use of ultrasonic and IR sensors also complements the navigation process.

**Communication Protocol**

The Arduino and Pi are linked via serial communication. Luckily as both the Arduino and the Pi work in the same endianness, there is no need to rearrange the bits during serial communication. Command packets are sent from Pi to Arduino and status response packets back from Arduino to the Pi, if requested. As packets might get corrupted during transfer, there is a need to ensure the correctness of the packet.

For transferring command packets, Pi sends a serialized packet of its corresponding packet type and command to Arduino. The Arduino reassembles the packet data to determine if the packet received is complete. If the packet is incomplete, there would be errors in checksum and magic number. Arduino would thus send a status packet to Pi informing the operator that the packet is corrupted and wait for the next action.

If the packet is complete, the packet would be deserialized into the data structure in Arduino Hence, the information regarding movements for Arduino to execute - distance to move, angle to turn, direction, as well as speed, could be obtained and subsequently executed by the Arduino movement functions.

**Checksums and Magic Number**

Checksums and magic numbers are instrumental in checking the correctness of a packet. They are included within the packet sent out and deserialized on the other end. Checksums are computed on both the Arduino and Pi to ensure the data received is uncorrupted. The magic number is a constant hexadecimal used to determine if the packet received is complete.

Therefore, checksums and magic numbers work in tandem to deduce the validity of packet. Depending on the status of the packet received, different response packet functions are called. These functions to be sent from Arduino to Pi are as followed.

|  |  |
| --- | --- |
| Function | Explanation |
| badChecksum() | Called when the computed checksum is different from the sent checksum. |
| badPacket() | Called when the packet type to be received is not a command packet, or when the magic number differs from that of the Pi. |
| badCommand() | Called when the user sent an undefined movement command. |
| badResponse() | Called when the initial handshaking packet sent from Pi to Arduino was unsuccessful. |
| sendOK() | Called if the packet has passed all the checks.  Informs the operator on Pi that the command has been received and would be executed as prescribed by the operator. |

*Fig. 8: A summary of the communication protocol functions on the Arduino*

**Fragmentation**

Fragmentation in packets is a common occurrence when the packet is sent in fragments at different timings. Hence, the receiving party would receive only a portion of the expected packet. This could cause errors or represent non-meaningful information.

The approach to address fragmentation requires the fragments to be accumulated to the full packet size before being deserialized into the data structure. A “leftover buffer” algorithm is written within the serialize file which tracks if the accumulated data received is equal to the expected packet size. If packet size mismatches, the next fragment received would be concatenated with the previous packet till the size matches. Any additional data exceeding the size of the expected packet size would be put into the leftover buffer to be concatenated to the next packet fragment.

**Section 6: Software Design**

The following section explains the principles behind the location map generation, Alex’s wireless communication with the user, and the colour identification.

On the Pi, a modified MCP program running via SSH from the remote laptop will give user the ability to send movement commands remotely. Through another SSH terminal, the user would also have to run the RPLIDAR.launch on the Pi to initialize the RPLIDAR node on the ROS network between the Pi and the remote laptop.

This is the essential component to offloading the SLAM algorithm as it publishes the scan data on the network, allowing the configuration file view\_slam.launch to read the data seen by the LIDAR, and to be transmitted to the user to map the foreign environment by running Hector SLAM node on the remote laptop. The Rviz module would then display the data calculated by Hector SLAM from the LIDAR readings to give a live mapping of the foreign environment.

The changes we made to establish the ROS nodes on the Pi and the remote laptop is explained in the table below:

|  |  |  |
| --- | --- | --- |
| Steps | Pi | Remote Laptop |
| Initialize Pi as ROS master | In the “.bashrc” file for the Pi, we exported the Pi’s Internet Protocol (IP) to be the ROS\_MASTER\_URI, and the ROS\_IP to the IP of the Pi to enable the nodes to recognize each unique IP address on the ROS network. | Similarly, the remote laptop would also have the ROS\_MASTER\_URI to be set as the Pi’s IP.  Also, set the ROS\_IP to the IP of the remote laptop to enable the nodes to recognize each unique IP address on the ROS network. |
| Running rplidar node on the Pi | This is the key node which publishes the scan data on the ROS network. The user does this via SSH connection on the Pi. | Check that the remote laptop is able to see the Pi as the master node, and that the scan topic is correctly publishing data by running Rostopic echo /scan |
| Running Hector SLAM and Rviz on remote laptop | Ensure RPLIDAR node is running and sending scan data while the hector slam node is active. | The hector slam would be displayed on a GUI called Rviz, together with the live laser scan data. This is what we use to map our foreign environment remotely. |

*Fig. 9: Details regarding ROS node setup between Pi and Remote Laptop [10]*

For teleoperation, small modifications were made to the given MCP code from the lab. One of Alex’s objectives is being able to determine the colour of an object when prompted, we needed to create the function that prompts Alex to run the colour sensing Arduino code. To accomplish this, we created another case for the command and handleCommand function, making edits to the constants.h file to add the additional case in.

The table below serves to outline the process when a colour test command is sent from the Pi to the Arduino on the MCP.

|  |  |  |
| --- | --- | --- |
| Command Process | Pi | Arduino |
| User sends Colour test command from MCP on Pi to Arduino | By pressing the letter ‘t’ or ‘T’ on the MCP, a colour command packet would be sent from the Pi to Arduino | Arduino unpacks the command packet sent and checks if its magic number is valid, followed by its checksum.  If it obtains a bad magic number, it will send a Bad Magic Number Packet back to the Pi.  If the checksum of the packet received does not match the checksum of the packet when it was sent, Arduino will send a Bad Checksum error packet back to the Pi.  When the Arduino correctly receives a proper colour command, it will run the colour sensing function. |
| Arduino runs the colour sensing function upon receiving valid command packet. | Pi waits for returning packet from Arduino. | The Arduino runs the colour sensing code, which samples a red and green frequency 50 times in succession to find the average frequency of the red and green colour of the object in front of Alex.  By testing with known red and green coloured paper, we can determine the approximate frequency ranges for the paper/object.  After we determine what colour the object is through the above tests, we created a red and green character arrays based on colour the object. |
| Arduino sends back the determined colour array and SendOK packet back to the Pi | Pi receives the colour of the object as a message together with the sendOK packet, informing the user that either the colour of the object or to try the colour test again. | The Arduino would send these back to the RPi in the sendMessage function. In this way, we were able to prompt Alex to determine the colour of the object, and in return receive a message telling the teleoperator what colour the object is.  If the colour does not match the determined ranges of the red and green coloured paper, a message is sent back to the RPi to tell the teleoperator to “Try again”, either requiring further movement adjustment to place the sensor closer to the object, or just another run of the colour sensing function. |

*Fig. 10: Details regarding colour detection command*

Another modification was to hard-code the desired distance/turning angle and power of the movement commands in the MCP. This gives the user a quicker way to control the robot, as just by inputting the direction of travel, Alex will move a fixed distance in that direction along with a fixed power value. This allows for faster movement, although at the loss of some precision. To combat this loss of precision, we deliberately set the movement thresholds of each command to be much more minute, such that the operator can more accurately move to the required direction through their command line inputs.

**Section 7: Lessons Learnt and Conclusion**

This section highlights our mistakes and lessons learnt throughout the execution of the project. It also discusses our learning points and solutions that were implemented during the project and those that could have been implemented in retrospect.

**Mistake 1: Using Windows subsystem for Linux instead of remote Ubuntu OS**

Since the Windows Ubuntu Subsystem for Linux (WSL) is not a full kernel of Ubuntu, it led to a lot of problems when creating the ROS nodes needed to offload slam to the laptop, and it is partly due to unsupported network communication between the WSL system and its corresponding nodes. This resulted in early testing of LIDAR mapping to be extremely inconsistent.

**Solution: Switching to a full Ubuntu OS running on a virtual machine**

After the switch was made, most of the issues we faced regarding ROS node connections were resolved, and we were able to successfully offload SLAM to our laptops through minimal changes in the launch files. Having gained experience from our unsuccessful attempts, we were able to switch to running a virtual machine from WSL with relative ease.

**Mistake 2: Attempted to mount LIDAR on Alex without planning out its position first**

We did not consider the LIDAR’s mounting hardware in relation to the magic chassis. This give us difficulties in maintaining the motor circuit require to power Alex.

**Solution: Repositioning all the components mounted on Alex’s back**

Once the LIDAR was added, there was insufficient space and not all the components were able to be fitted onto the magic chassis. After much deliberation, we had to shift our Raspberry Pi to a position where it cannot be secured by screws. Eventually, we managed to overcome this by using wires to secure our Pi to the chassis.

**Lesson 1:**

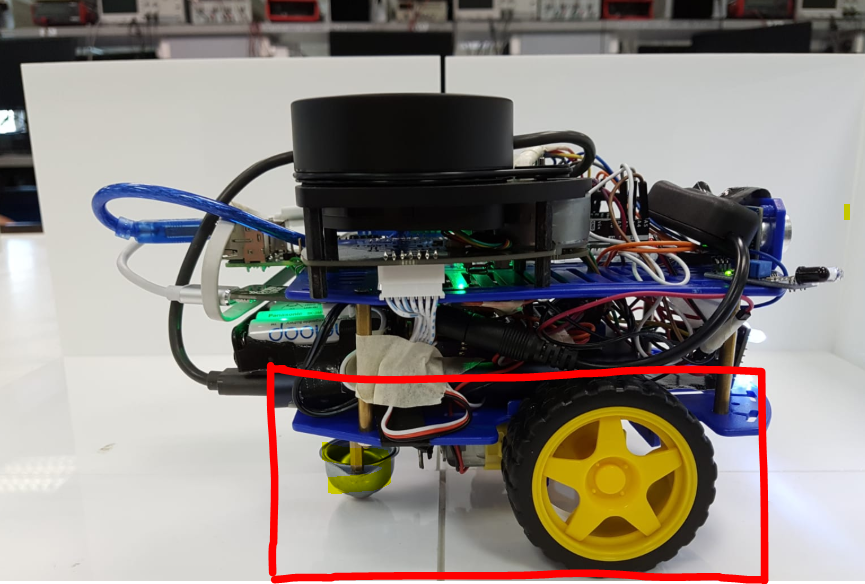
One of the biggest lessons we learnt was that we must be prepared to make on the spot changes to our project when we encounter difficulties. In that sense, we had to have an open mind for problem solving, when a component breaks down, or seemed to be unreliable, we must make a swift decision to either attempt to repair the issue or completely redo the component from scratch.

An example of this was when our robot started to smoke due to exposed contacts between the wires on the breadboard, resulting in a shorted circuit. Our decision was to immediately take the robot apart and redesign the circuit to ensure that there will be no such exposed contacts, and to reduce the bulk of wires that was contributing to the incident.

We were able to accomplish the task within 2 hours and was only possible when we had 2 sub-teams delegated to one main task each. One to focus on rebuilding our robot, and another to complete the presentation materials.

**Lesson 2:**

The other major lesson was that we did not considered the weight distribution of our robot and designed the robot in the wrong direction as seen from the image below.



*Fig. 11 Weight distribution problem*

This is crucial since the magic chassis was built in such a way that most of the weight distribution should be placed above the wheels instead of the free-moving ball bearing. As a result, our robot’s forward movement was weaker compared to the reversing motion which was too strong in power.

This issue caused our initial attempt during the test to be unsuccessful. Given a second attempt, we had to quickly change the MCP code to include a larger power threshold to power the forward motion.

This corresponds well with our initial lesson learnt, where we must think of a solution to a malfunctioning component on the spot and under pressure. After the change, we were able to complete the demo task within the stipulated time limit.

What we learnt from this was that we should plan our hardware designs beforehand and not rush into implementation to avoid such issues during the later stages of development, where it would be extremely tedious to redesign from scratch caused by poor initial design. By having an open mind, we were able to solve the forward movement and complete the search and rescue task within 7 minutes.

To conclude, our team had learnt a lot from this challenge. From understanding the reasoning behind why the source code are written the way it is, debugging the various colour and mapping challenge requirements, the pros and cons of each design implementation, and finding suitable solutions on the spot while under pressure. These are all valuable lessons that would be useful in our journey to become a better engineer.

**References**

[1] World Health Organization, “Natural events”. [Online].

Available: [ttps://www.who.int/environmental\_health\_emergencies/natural\_events/en/](https://www.who.int/environmental_health_emergencies/natural_events/en/).

[Accessed Apr. 15, 2019]

[2] UNHCR, “Search and rescue response and coordination (natural disasters),” *The UN*

*Refugee Agency Emergency Handbook*. [Online].

Available:

[https://emergency.unhcr.org/entry/258567/search-and-rescue-response-and-coordinati](https://emergency.unhcr.org/entry/258567/search-and-rescue-response-and-coordination-natural-disasters)

[on-natural-disasters](https://emergency.unhcr.org/entry/258567/search-and-rescue-response-and-coordination-natural-disasters)

[Accessed Apr. 15, 2019]

[3] Trends in Japan, “Rescue Robots,” September, 2010. [Online].

Available: <https://web-japan.org/trends/09_sci-tech/sci100909.html>

[Accessed Apr. 5, 2019]

[4] Tokyo Fire Department, “救出ロボット(愛称　ロボキュー),” [Online]

Available: <http://www.tfd.metro.tokyo.jp/ts/soubi/robo/05.htm>

[Accessed Apr. 17, 2019]

[5] Trends in Japan, “Rescue Robots,” September, 2010. [Online].

Available: <https://web-japan.org/trends/09_sci-tech/images/l_scia100909.jpg>

[Accessed Apr. 5, 2019]

[6] Muza-chan, “Robocue, the Firemen's Robot,” February, 2011. [Online].

Available:

<https://muza-chan.net/aj/poze-weblog2/robocue-tokyo-fire-department-robot.jpg>

[Accessed Apr. 5, 2019]

[7] D. Starkly, “Researchers used robotic snakes to search for earthquake survivors,” June

10, 2017. [Online].

Available: <https://www.geek.com/tech/researchers-used-robotic-snakes-to-search-for-earthquake-survivors-1718587/>

[Accessed Mar. 24, 2019]

[8] Biorobotics Lab Carnegie Mellon University, “Modular Snake Robots,” 2010.

[Online].

Available: <http://biorobotics.ri.cmu.edu/projects/modsnake/images/5.JPG>

[Accessed Apr. 17, 2019]

[9] Biorobotics Lab Carnegie Mellon University, “Snake Gaits”, 2010. [Online].

Available: <http://biorobotics.ri.cmu.edu/projects/modsnake/gaits.html>

[Accessed Apr. 17, 2019]

[10] Rahul Bhadani, “ROS/ NetworkSetup,” August 08, 2018. [Online].

Available: <http://wiki.ros.org/ROS/NetworkSetup>

[Accessed Apr. 02, 2019]