CG1112 Engineering Principle and Practice

Semester 2 2017/2018

"Vincent to the Rescue"

Final Report

Team: 02-02-01

Name	Student #	Sub-Team	Role
James Arista Yaputra		Hardware	Leader - Hardware
Lee Kok Teng		Firmware	Leader - Firmware
Ler Wei Sheng		Software	Nominal Team Leader
Matthew Alexander Jacobus		Hardware	Leader - Hardware
Srivastava Aaryam		Software	Leader - Software

Section 1- Introduction

1.1 Mapping the environment

Vincent, is a search and rescue robot that is meant to be operated both manually and autonomously. The robot is intended to survey its surroundings with the help of Light Detection and Ranging (LiDAR), Inertial Measurement Unit (IMU) and the wheel encoders. Vincent's mapping can be broadcasted via WiFi onto a Robot Operating System (ROS) server in the form of visual data to aid in manual controls as the operator will not be able to see the bot's surroundings very clearly during operation. The robot will also be able to make use of the map data to find its way back to the operator upon losing connection or upon request.

1.2 Place markers

In addition, during the teleoperated phase of the assessment, Vincent will be required to mark three locations along its traversal path. This will be achieved by storing the waypoint locations in a stack, to indicate that the positions have indeed been marked.

1.3 Backtracking to the initial location

Finally, Vincent has to tackle the problem of backtracking to its initial location once it has reached the final location. To solve this, Vincent will store the set of commands issued to it in a memory buffer on the RPi. Once Vincent reaches the final position, each command in the memory stack will be inverted in such a way that the robot reaches the initial position, while making sure it passes through each of the markers set during the teleoperated phase. Additionally, Vincent is able to signal a visual cue using a Light Emitting Diode upon every visitation of each marker.

Section 2- Review of State of the Art

2.1 Quince, Search and Rescue robot - Teleoperated

Quince is a versatile search and rescue robot jointly developed by Japan's International Rescue System Institute, Tohoku University and the Chiba Institute of Technology to work as a search and rescue robot in CBRNE (Chemical, Biological, Radiological, Nuclear or Explosive hazards) type emergencies.

2.1.1 Architecture of Quince

Quince has a modular architecture in both hardware and software components which allows flexibility in attaching/swapping additional sensors depending on the mission criteria. The robot's traversal mechanism is designed to overcome uneven terrain such as debris and stair steps [2]. As of yet, Quince has been deployed to survey the damages of the Fukushima Daiichi nuclear plant caused by the East Japan Earthquake on March 11, 2011[1].

2.1.2 Uses of Quince

The robot proved useful when radiation levels were alarmingly high, to an extent that is too unsafe for workers to work in[6]. Due to its modular base design, modifications were made to better suit the needs of the survey mission in the nuclear plant. Some of these include

additional sensors and tools such as dosimeters, a water level gauge and a mechanism for handling a water-sampling cup[7].

Strengths:

1. Powerful motors and well-designed crawlers allow traversal over rough and uneven terrain, as seen in Figure 1. Some of these features include:
    a. Debris filled terrain
    b. Staircases
2. Dustproof/Waterproof that allows easy clean up after inspecting hazardous substances
3. Various environmental sensors available for attachment
4. Modular design

Weaknesses:

1. Communication cable rewinding device on Quince was not designed to rewind long cables and thus failed during the Fukushima Daiichi mission[7]

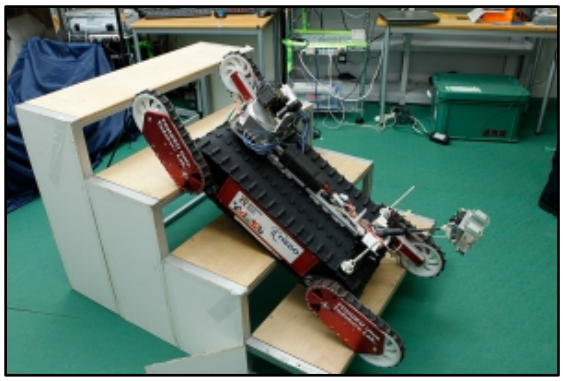

Figure 1. Disaster Response robot Quince [6]

2.2 Quince, Search and Rescue robot - Autonomous algorithm module

2.2.1 Movement of Quince

The movement mechanics in Quince is enabled by its low centre of gravity, main body crawler and four independent sub-crawlers [4]. However, since the movement of every component is individually teleoperated, there is a possibility that the operator will face difficulties during stressful missions. This shortcoming was documented during the Fukushima Daiichi nuclear plant disaster [7].

2.2.2 Future of autonomous version of Quince

A research project started by University of Campinas in collaboration with various universities, institutes and companies lead to the creation of an autonomous algorithm module that aims to address this issue [3]. This module adopts different control strategy for the flippers depending on movement direction [4].

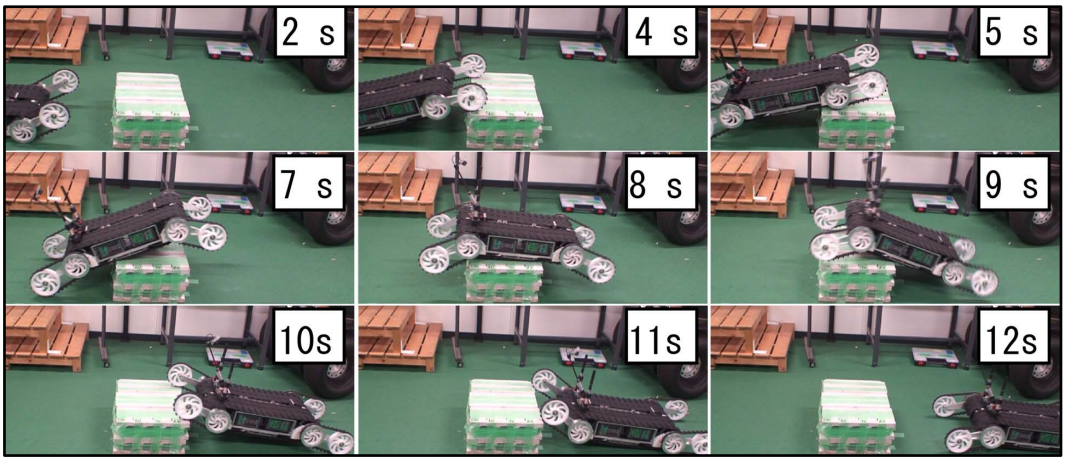

Figure 2. Quince overcoming obstacles with algorithm module

Strengths:

1. Substantially aid the operator in moving the robot, as seen in Figure 2.
2. Reduce operation stress on the operator
3. Provide improved navigation which reduces the chance of accidents

Weaknesses:

1. The algorithm may not be robust enough to handle extreme terrain conditions
2. The robot is still prone to human errors

Section 3- System Architecture

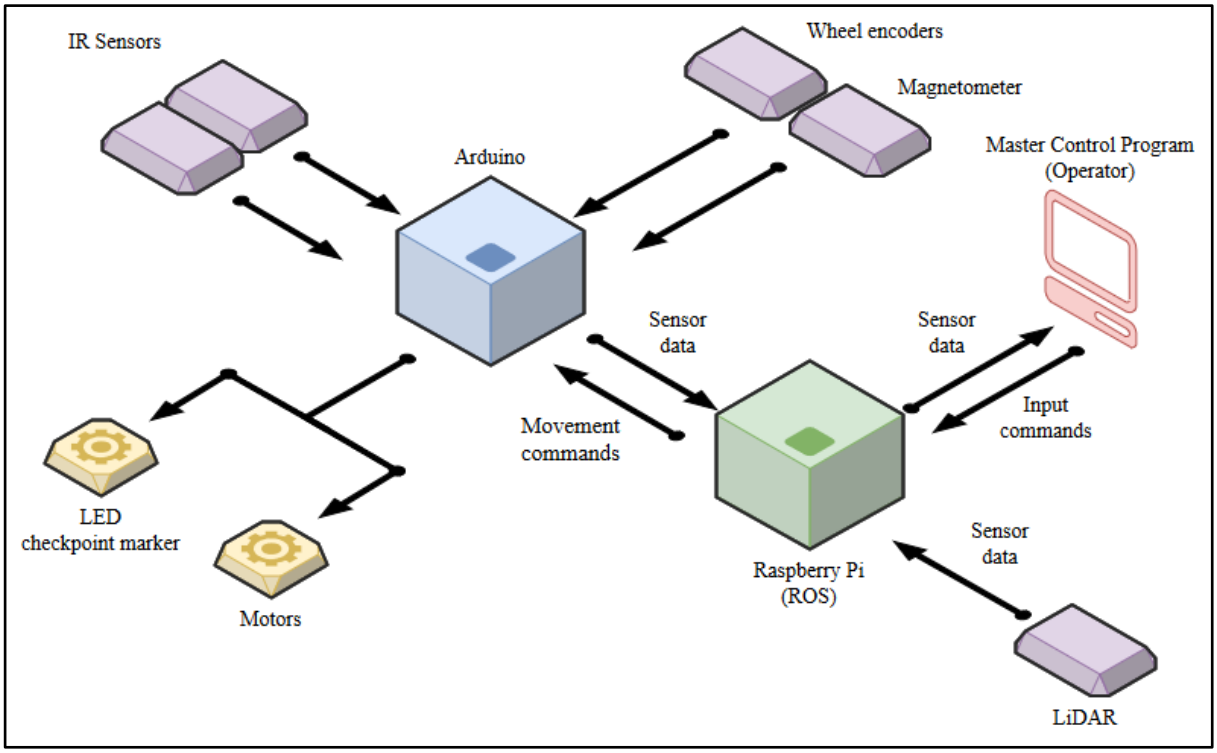

Figure 3. Overall system architecture for Vincent

3.1 Main architecture of Vincent

The Vincent Search and Rescue robot will have an Arduino Uno microprocessor to interface with various sensors. The sensors that will be included within the system are as follows:

1. Magnetometer
2. Wheel encoders
3. Infrared (IR) sensors
4. Light Detection and Ranging (LiDAR) sensor

These sensors will be interfaced to the Arduino using I2C (Inter-IC) or digital input with the exception of LiDAR, which will be interfaced with the Raspberry Pi 3 (RPi) directly. To elaborate further, the magnetometer will be used alongside the wheel encoders to approximate Vincent's location with respect to its surroundings. The IR sensors will be used for path correction.

3.2 Interfacing between Arduino and RPi

The Arduino will be interfaced with the RPi, which will be connected to the master control program (MCP) via a SSH secure channel. The underlying software that will be implemented for the LIDAR is the gmapping[6], which is a package in the Robot Operating System (ROS).

Section 4- Hardware Design

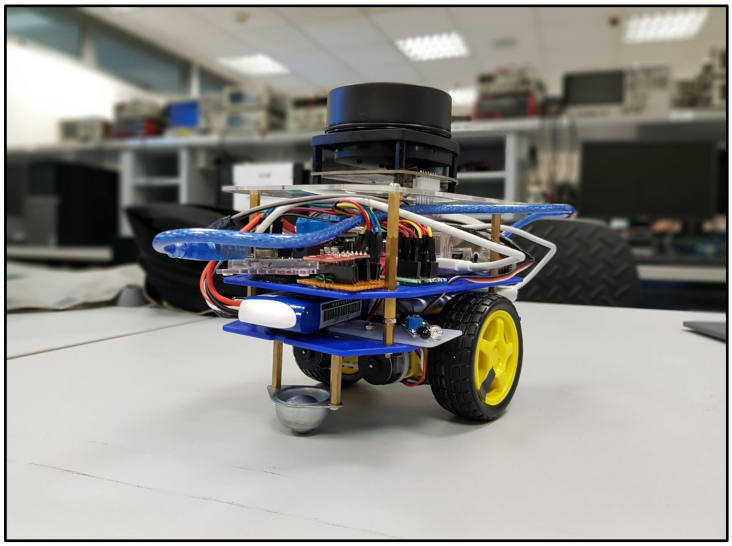

Figure 4. Final Design of Vincent

4.1 Overall Vincent Design

As seen in Figure 4 above, the team's design of the Vincent Search and Rescue robot consists of various subsystems and additional components. These systems will be explained and illustrated in the subsequent sections.

4.2 Magnetometer for accurate headings

To provide Vincent with accurate new heading readings while turning, the team decided to incorporate a magnetometer (shown in Figure 5) to allow Vincent to determine the direction it is facing. The magnetometer used (MAG3110) is powered with a 3.3V voltage source, so in order for the Arduino Uno to read proper data from the magnetometer, the team decided to purchase a logic level converter to convert the 3.3V analog data provided by the magnetometer to a 5V analog data stream that can be properly read by the Arduino Uno. Furthermore, to fine-tune the magnetometer readings, the team has also included an offset calculation routine within the software.

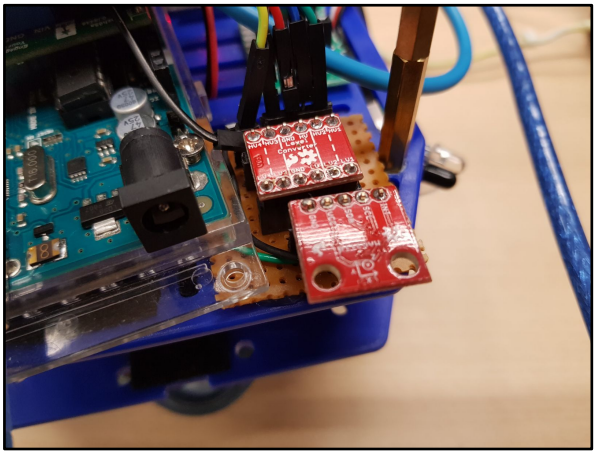

Figure 5. Magnetometer for Vincent

4.3 IR Sensors for path straightening

To account for the narrow passageways Vincent will need to traverse through, and the unsynchronized nature of the motors provided, the team opted to use a pair of infrared sensors (as depicted in Figure 6) to determine the distance of the walls surrounding Vincent's sides to allow Vincent to move straight without colliding with the walls.

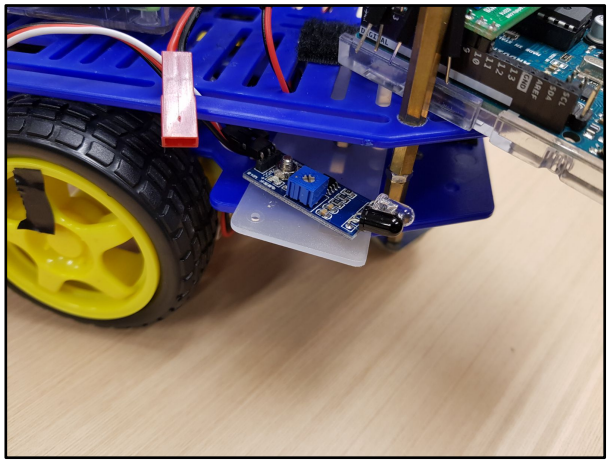

Figure 6. Vincent IR Sensors

4.4 Use of a new motor driver

The team also decided to use a commercially available motor driver (shown in Figure 7) since the driver has integrated logic gates that simplifies the control interface of the A4990 by reducing the number of PWM signals required. Each channel has a speed control input, MxPWM, and a direction control input, MxDIR. Arduino pins 9 and 7 are used to control the speed and direction, respectively, of motor 1, and pins 10 and 8 control the speed and direction of the right motor.

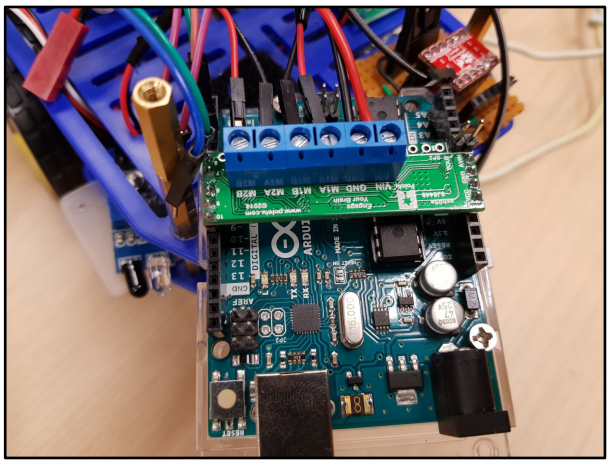

Figure 7. Vincent Motor Driver Shield

Moreover, the motor driver has a built in internal logic circuit, the truth table of which is illustrated below in Table 1.

TABLE I. TRUTH TABLE FOR THE OPERATIONS OF THE SHIELD

MxDIR	**MxPWM**	**MxA**	**MxB**	**operating mode**
0	PWM	PWM	L	forward/brake at speed *PWM %*
1	PWM	L	PWM	reverse/brake at speed *PWM %*
X	0	L	L	brake low (outputs shorted to ground)

Section 5- Firmware Design

5.1 Motor control design

Initialization

- ➢ The Robot Operating System (ROS) that is hosted by the RPi will launch as a system service.
- ➢ ROS will establish a connection with the Arduino mounted on Vincent, the Arduino would then control the motors and sensors.
- ➢ The LiDAR will then activate and begin sending data to the RPi, which will then be used to map Vincent's surroundings.

Sending user command

- ➢ The operator will connect remotely to the RPi via a SSH channel.
- ➢ The operator sends a command with three different parameters namely, direction, distance or angle for Vincent to traverse, and the power allocated to the motors.
- ➢ The command will be sent to the RPi through a local area network connection.

Processing user command

- ➢ A subroutine takes in the three parameters as input and signals the A4990 motor driver channels through the respective Arduino pins. The motor driver then relays the appropriate signals to each motor.
- ➢ Another subroutine computes the target travelled distance using the input traverse distance and current travelled distance measured with wheel encoders.
- ➢ The motor spins and Vincent starts to move.
- ➢ During movement, the program continuously poll for the current travelled distance.
- ➢ In addition, the IR sensors detect close proximity with walls on each side and adjusts each motor accordingly to avoid collision.

Termination

- ➢ When the current travelled distance matches the target travelled distance, the motors will stop and movement is completed.

5.2 Magnetometer-based turning

Initialization

- ➢ The magnetometer is initialized using the Wire library by specifying to the Arduino the I2C address of the magnetometer, and which register will be used to store readings.

Sending user command

- ➢ The operator will send a command specifying a turn (left or right).
- ➢ The angle parameter specified by the operator will then be passed to the heading processing subroutine.

Processing user command

- ➢ After the angle parameter has been received, the subroutine computes the destination heading required using the angle parameter.
- ➢ Vincent starts turning.
- ➢ During turning, the magnetometer continuously poll for the current heading.

Termination

- ➢ When the current heading matches the destination heading, the motors will stop and the turning is completed.

5.3 Communication Protocol

Initialization

- ➢ A dedicated router is used to provide a wireless connection between the RPi and the master control program.
- ➢ During initialization, the RPi will begin sending requests to the Arduino via a USB cable.
- ➢ The Arduino will then begin receiving data from the sensors via the I2C interface or analog pins and begin processing it.
- ➢ The Arduino will initialize and remain idle until a user input is passed by the RPi.

Sending user command

- ➢ When a command is entered, it will be sent to the RPi, this packet of data would be sent wirelessly to the RPi.
- ➢ The RPi will send the user command to the Arduino.
- ➢ An USART RX Interrupt Service Routine (ISR) will be triggered once the Arduino has finished receiving the operator input.
- ➢ In this ISR, user data received from pin UDR0 will be read. Then, the interrupt flag for the received data will be reset.
- ➢ Once Arduino has successfully processed the command, the MCP is notified and waits for next input.

Processing command

- ➢ There will be several predefined user inputs, each unique input will be mapped to set the speed and direction of the motors.
- ➢ Then, the required PWM control pins will be set for each specific motor function such as *rightMAG()*, *leftMAG()*, etc. The motors will then be activated.
- ➢ An integer will then be passed to the Pololu motor driver shield to be converted into a valid PWM value to be used to power the motors.
- ➢ The PWM cycle that is generated will remain unchanged until the user inputs a new command.

Section 6- Software Design

6.1 General movement - Teleoperated and Autonomous

Vincent's movement mechanics can be classified into two main conditions, namely autonomous and non-autonomous (teleoperation) mode. By default, Vincent is booted up in teleoperated mode. In teleoperation mode, Vincent requests for user input and process that command and parameters subsequently. Once Vincent is switched to autonomous mode, it attempts to process all the commands in the backtrack stack, starting from the top until the stack is empty. These processes are represented in Figure 8 overleaf.

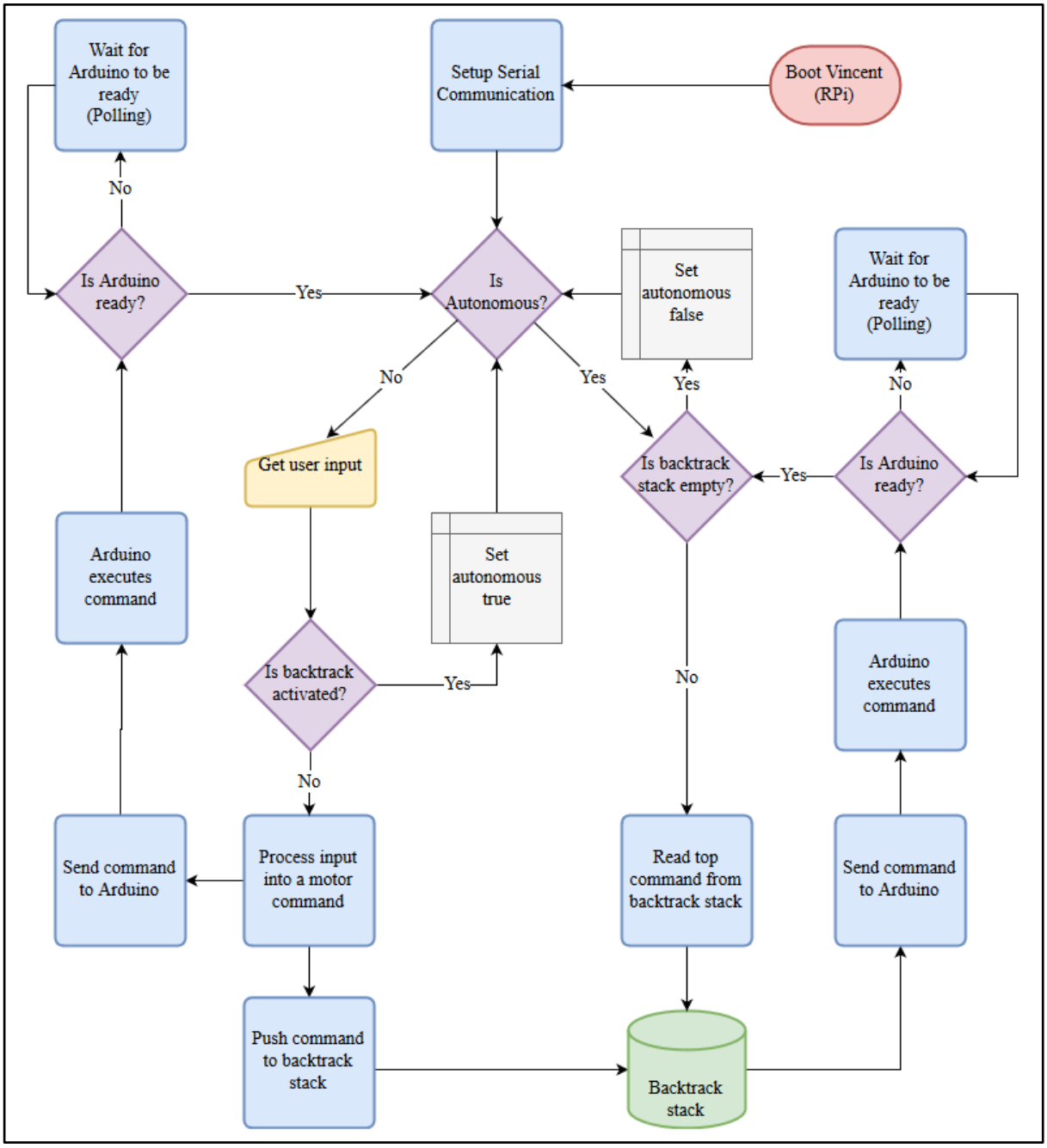

Figure 8. Algorithmic Flowchart

6.2 Marking checkpoints of Vincent

The Vincent robot would need to mark checkpoints as it passes through the required locations. These markers are marked based on floating point coordinates on the occupancy grid. This allows Vincent to mark locations with a 0.05m precision based on the team's LiDAR settings.

Once it has passed through all the locations, it will then backtrack using the saved location markers. This will allow Vincent to navigate autonomously by referring to the location markers and backtrack through the shortest path that will be determined using a pathfinder algorithm.

`

6.3 Backtracking of Vincent - Command Stack

Vincent would store teleoperated commands issued to it as it moves. These commands would be stored in a stack which behaves in a FILO (First in Last out) manner allowing it to stack up commands until the most recent command. At the end of its journey, just one last command has to be issued to tell it to perform autonomous backtracking. Vincent will then pop the instructions off the stack one by one and perform the exact opposite movement as it traverses back to its origin.

6.4 Backtracking of Vincent - Marker Stack and Quaternion Angles

On top of the command stack used for backtracking, the team decided to implement another method of backtracking, where instead of storing commands in the navigation stack waypoints are instead stored inside the navigation stack. This method would remove any inconsistencies with incorrect movements during the backtracking phase, as now Vincent will associate an euler angle and direction to move from its current position to the stored waypoints. Moreover, all of the waypoints are placed in the stack such that each waypoint is either completely horizontal to each other or vertical to each other (i.e. markers are placed each time Vincent moves to another junction).

In order to associate each waypoint with an euler angle, Vincent needs to listen to a ROS topic called "Slam Outpose", and thereby receive a quaternion angle based on the current heading of the Lidar. A quaternion angle is essentially a linear combination of four variables, which will be represented as q_0, q_1, q_2, q_3 for this report. Using the following equation, we transform the given quaternion angle into a Euler angle which is the standard 0-360-degree scale:

$$Angle = atan\left(\frac{2(q_0 q_3 + q_1 q_2)}{1 - 2(q_2^2 + q_3^2)}\right)$$

6.5 A* Algorithm

As an additional feature, the robot can also mark virtual checkpoints into a stack in the form of coordinates using ROS markers and then find the shortest path to these markers sequentially. Shortest path will be determined by the A* algorithm [5], $O(V^2)$ time complexity. A* is a variation of Dijkstra's shortest path algorithm that incorporates heuristics as a method to determine the shortest path in a graph optimally. In this case, the team has modeled the SLAM map output as a graph where every grid cell is a node and every node that is next to each other and not an obstacle is a connected edge. A* is then used to find the shortest path from Vincent's current position to the destination marker.

Unlike traditional methods of constantly generating A* star on the map to update and autocorrect the robot's path, the team has decided to only generate A* after each junction so as to save up on computational resources. This will be called by the master control program in a ROS topic asynchronously and will receive the heading and distance to next junction as a response.

A brief definition of a junction in an A* path is illustrated in Figure 9.

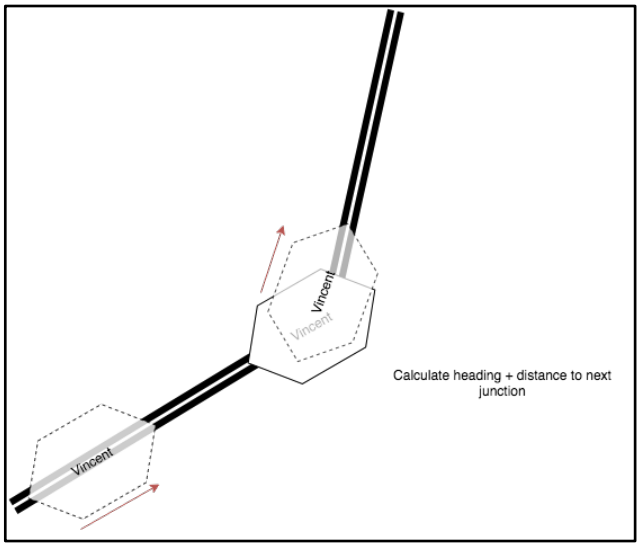

Figure 9. A* algorithm illustration

Section 7- Conclusion

7.1 Important Lessons learnt

Firstly, time management was a key issue that plagued the team's performance during the process of generating Vincent, coupled with division of workload being ambiguous. Initially, the team delegated the workload in such a manner that most of the work would be software centred. However, after testing with the hardware pieces provided, the team quickly realised that numerous parts of the hardware would have to be replaced in order to achieve optimum functionality.

Secondly, the hardware provided, including the chassis for the robot, the battery pack, as well as the magnetometer, required either complete replacement or heavy calibration and additional complementary hardware for optimum results.

7.2 Two Greatest Mistakes

One of the biggest mistakes that was made was to follow the specifications of the project too closely, as they often changed during the course of the project. Hence, quick adaptations to the tentative plans and material decision had to be made in order to fit the requirements. Moreover, the shortest path algorithm could be implemented to fulfill phase two of the project. However, after further clarification Vincent would need to pass through all the markers during the backtracking stage. As a result, A* shortest path algorithm has been repurposed to an additional feature in our project.

Due to the team's marking algorithm, Vincent has a heavy reliance on the LiDAR generated map. Although this gives a 0.05m precision according to the LiDAR resolution, the markers were lost as soon as the map warps or shifts out of origin.

References

[1]R. Association, "Robotics Online", *Robotics Online*, 2018. [Online]. Available: https://www.robotics.org/content-detail.cfm?content_id=5537. [Accessed: 16- Mar- 2018].

[2]"Meet Japan's Earthquake Search-and-Rescue Robots", *Popular Science*, 2018. [Online].Available:https://www.popsci.com/technology/article/2011-03/six-robots-could-shape-future-earthquake-search-and-rescue. [Accessed: 16- Mar- 2018].

[3]"Videos: Super-mobile rescue robot Quince", *TechCrunch*, 2018. [Online]. Available: https://techcrunch.com/2010/05/04/videos-super-mobile-rescue-robot-quince/. [Accessed: 14- Mar- 2018].

[4]E. Rohmer et al. 2010. Integration of a sub-crawlers' autonomous control in Quince highly mobile rescue robot. 10.1109/SII.2010.5708305.

[5]"Finding shortest paths on real road networks: the case for A*", *zenodo.com,* 2018. [Online]. Available: https://zenodo.org/record/979689#.WtrG9NNubOQ. [Accessed: 15- Apr- 2018].

[6]"Fukushima’s Radioactivity-Proof Cleanup Robot", *nippon.com*, 2018. [Online]. Available: https://www.nippon.com/en/views/b00904/. [Accessed: 15- Mar- 2018].

[7]T. Yoshida, K. Nagatani and S. Tadokoro, *Improvements to the Rescue Robot Quince Toward Future Indoor Surveillance Missions in the Fukushima Daiichi Nuclear Power Plant*. 2018.