CG1112 Engineering Principle and Practice
Semester 2 2019/2020

"Alex to the Rescue"
Design Report
Team: 3-3-1

Table 1: Role Delegation

Name	Student #	Sub-Team	Role
Samuel Paul Christopher	A0200667J	Software/Hardware	Hardware Engineer
Neo Yao Jie Joel	A0199695Y	Hardware	Hardware Engineer
R Ramana	A0197788X	Software	Software Engineer
Matthew Gani	A0204882A	Software	Software Engineer
Gunit Mittal	A0206438E	Software	Software Engineer

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

Figures

Tables

Introduction:

This report seeks to relay information regarding the construction of Alex, a robotic search and rescue device. The primary functionality of Alex is to navigate and map out the environment. Navigation will be done through the Master Control Programme (MCP) on the Raspberry Pi (herein referred to as Pi). This will be manually controlled by a user behind a screen. If time permits, we seek to design a user interface, by setting up a web server so as to ensure the ease of transmission of data, and navigation control. The MCP will relay the user input to the Arduino. Alex will be able to map its environment using the Simultaneous Localization and Mapping (SLAM) algorithm. The map produced will be sent back to the user.

Alex will carry out the following:

1. Mapping of the environment
2. Transmit back the relevant data of its current position to the user
3. Avoid obstacles and traverse in and out of rooms as quickly as possible
4. Being energy efficient in order to last a longer duration

In the subsequent sections of this report, we seek to elaborate on the system functionalities and architecture, as well as the component design of Alex.

Section 1: System Functionalities

This first section will outline our vision for Alex and delineate the essential and additional functionalities that we hope to build for Alex.

Alex and the Operator will be in two different rooms. There needs to be a two-way communication link between Alex and the Operator such that:

1. Alex can receive instructions from the Operator
2. Operator can see the map of the environment (generated by Alex) and output messages (these are information that will aid the Operator to make better decisions which include the distance covered by Alex)
3. Operator has a set of controls that he can use to control Alex

The above mentioned system functionalities are illustrated in Figures 1, 2 and 3 as shown below.

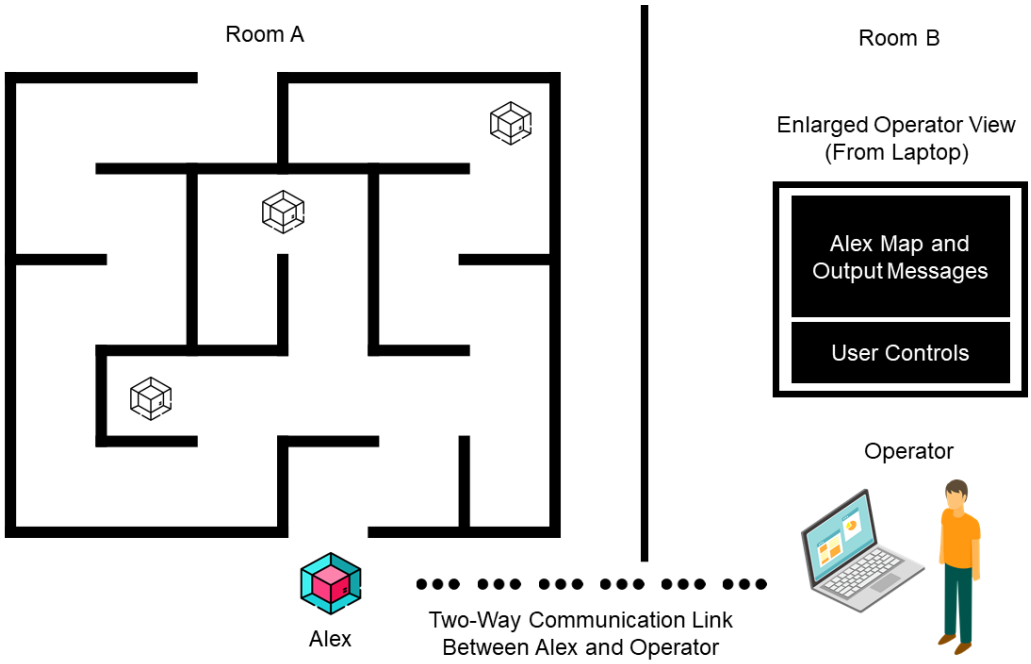

Figure 1: Tele-Operation
Source: Adapted from [1] [2]

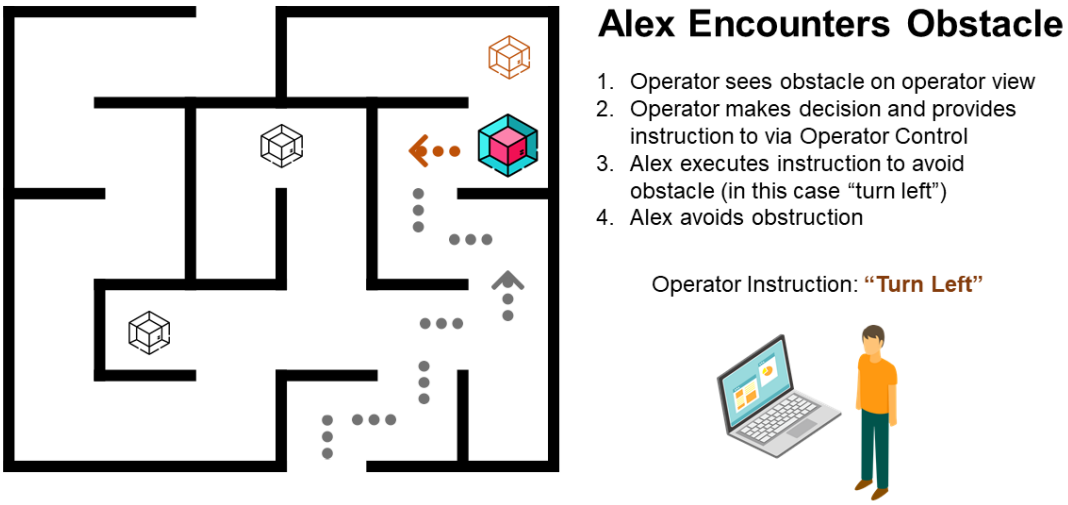

Figure 2: Overcome Obstacles
Source: Adapted from [1] [2]

The map produced by Alex should be an accurate representation of the environment and show outer wall dimensions as shown in Figure 3 below.

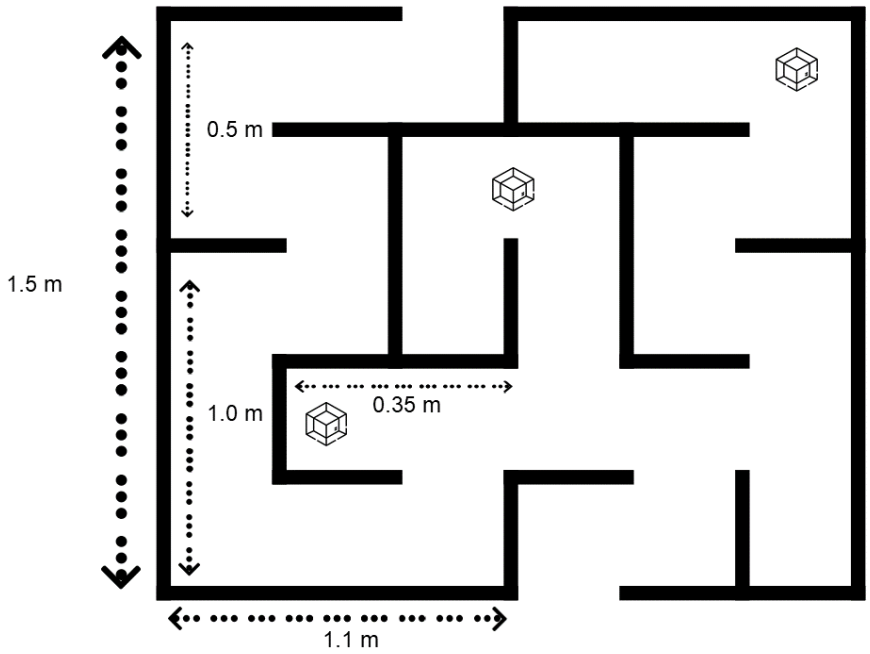

Figure 3: Accurate Environment Mapping
Source: Adapted from [1] [2]

Travelling Speed of Alex:

To be able to select a suitable (and the fastest) speed at which Alex should travel, the presence of latency is taken into consideration. Latency is the delay before the transfer of data begins after an instruction has been sent by the sender. This would mean that there might be several possibilities that obstacles might be sensed by Alex, but the data not being able to reach the receiver in certain short periods of time due to latency issues. Collisions might occur if Alex is travelling at a speed faster than the latency period. Therefore, to obtain a good balance between the accuracy of the sensors and the speed travelled by Alex, test runs will be implemented.

Usage of SLAM and LIDAR:

Simultaneous Localization And Mapping, or SLAM, is a collection of algorithms that enables the construction and mapping of an unknown environment, all while allowing the user to monitor the robot's position. SLAM is always used with several types of sensors. In this case, coupled with the infrared laser signals, the LIDAR emits laser signals in a circular manner around the surroundings. The laser bounces from the various obstacles and objects in the environment and is received back by the LIDAR to go through processing. The data collected goes through a digital signal process (DSP), which produces a graphical representation of the simulated environment. This visualization process provides an accurate depiction of the surrounding environment, making it easier for the user to control Alex that is traversing the terrain while avoiding any collisions.

Acknowledging the perks of SLAM, the SLAM combined with the visualization process consumes approximately 90% of the CPU load. This is clearly a downside as the large energy consumption will cause the battery to drain quickly. The energy inefficiency of Alex is also a big drawback.

Graphical User Interface (Additional Functionalities):

The Graphical User Interface (GUI) allows easier and more intuitive usage for the user. The user will be able to control the robot's direction, distance travelled as well as the angle of movement. The mapping and processing work will also be offloaded to the user's CPU which will allow for a more efficient energy usage of the Pi's CPU.

Power Savings (Additional Functionalities):

Power consumption is an area of consideration when designing Alex. Arduino has a built-in sleep function that allows the user to stop or turn off any unused modules in the Microcontroller, which in turn significantly reduces the power consumption. Together with interrupts, the function removes the need to continuously poll for instructions, enabling the Arduino to be more efficient in executing tasks.

Section 2: Review of State of the Art

In the previous section, we painted a picture of how we envision our teleoperated robot, Alex. In this section, we provide the findings of our research on two commercial teleoperated robots, namely TALON and SmokeBot, which helped us to craft our vision for Alex.

Section 2.1: TALON

TALON is an unmanned military robot (refer to Figure A.1 in Appendix A), produced by Foster-Miller [3], and is used in a myriad of situations such as first response, heavy lifting and rescue missions. They are built with high payload-to-weight ratio and a modular design, making them highly customizable with various sensors and components. The TALON robot can be controlled by a two-way radio or fibre-optic link [3], or even via either a laptop control unit or a tactical robotic controller [3].

Strengths

The TALON is simple to use, fast and sturdy [3]. It is customizable and hence suited for multi-purpose use in various environments. Sensors can be used to detect gas, chemical, radiation and so on, while a variety of weapons can also be attached [3]. The heavy-duty rotating shoulder is used for heavy-lifting [3]. The TALON can cost less ($230, 000) than what is normally spent on a soldier ($850, 000) per year [4].

Weaknesses

TALON relies heavily on constant and stable latency. Any delay due to unstable internet connection could possibly cause it to do wrong things at the wrong time, such as driving into wrong terrain [5]. TALON also uses two lead-acid rechargeable batteries which tend to have a lower capacity [6]. These batteries have a lower depth of discharge when compared to lithium-ion batteries [6]. These batteries also require frequent maintenance and tend to have lower life-spans, possibly resulting in higher costs in the long run [7].

Section 2.2: SmokeBot

SmokeBot seeks to address limitations that robots have in environments where sensor technologies can be affected due to dust and smoke [8]. The robot is built specifically to counter low visibility situations to help firefighters in search and rescue missions (refer to Figure A.2 in Appendix A) [8]. The robot thrives in a situation where humans are not able to do so. Using a range of sensors (thermal, radar, gas), the SmokeBot is able to perform in visibility impairing conditions. It is also able to detect harmful gases and this can help protect firefighters [8].

Strengths

As mentioned above, using a range of sensors, the SmokeBot is able to perform in visibility impairing conditions [8], and can even detect harmful gases and a possibility of gas explosions, helping to protect firefighters [9]. SmokeBot is able to tell if there is a possibility of a gas explosion by using data collected from its sensors [9]. In the case where the SmokeBot loses internet connection, it has the ability to automatically traverse back to the last place where it had a connection [9].

Weaknesses

SmokeBot cannot be used in immediate rescue efforts as it takes about 15-30 minutes to collect information [9]. Although it has the ability to automatically traverse back to the last place it had a connection, the algorithms used can be inefficient and can cause the SmokeBot to skip closer safe areas with stable internet connection [10].

Section 3: System Architecture

This section comprises of the simplified UML component diagram to show our design for Alex and how the different software and hardware components interact with on other to achieve our desired functionalities of Alex.

Referring to Figure 4 below, the ovals represent the software components, the rectangular boxes represent the hardware components. The arrows show the flow of data to and from the various hardware components, while the wires represent the relevant communication tools which will be used to transmit the data.

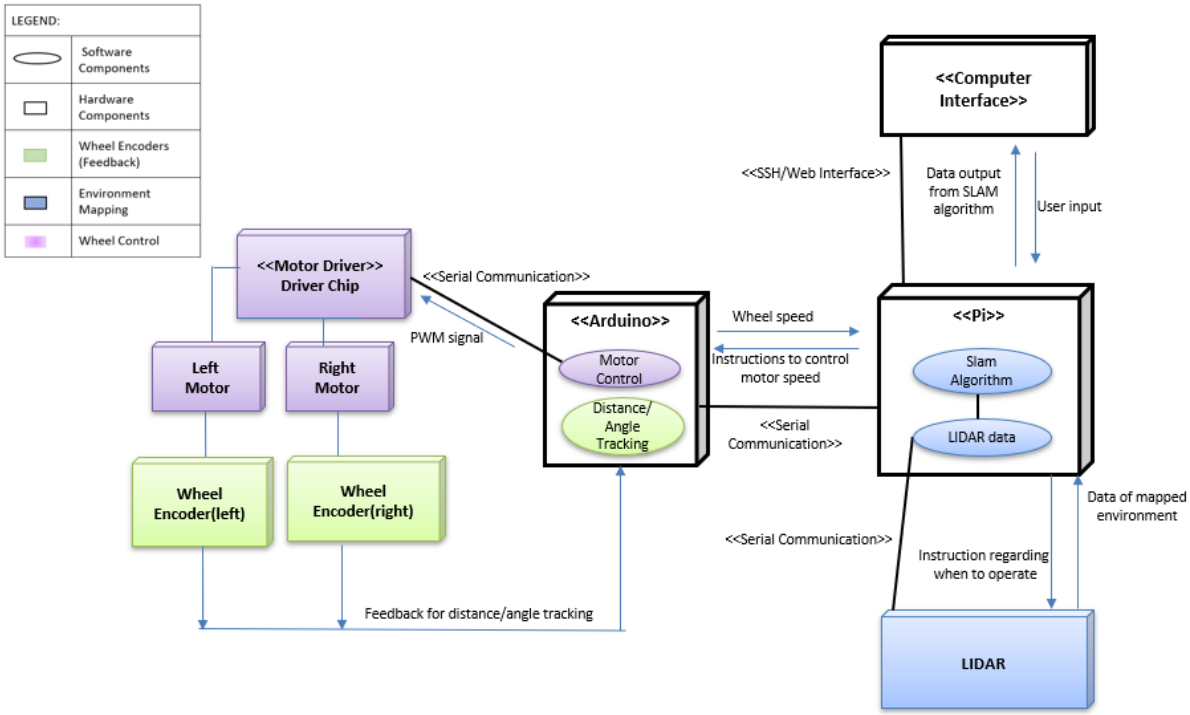

Figure 4: System Architecture

Section 4: Component Design

This section will cover our overall algorithm that we will use for Alex and will also include more detail about each step. The overall sequence of events is shown as a Flowchart (Figure 5).

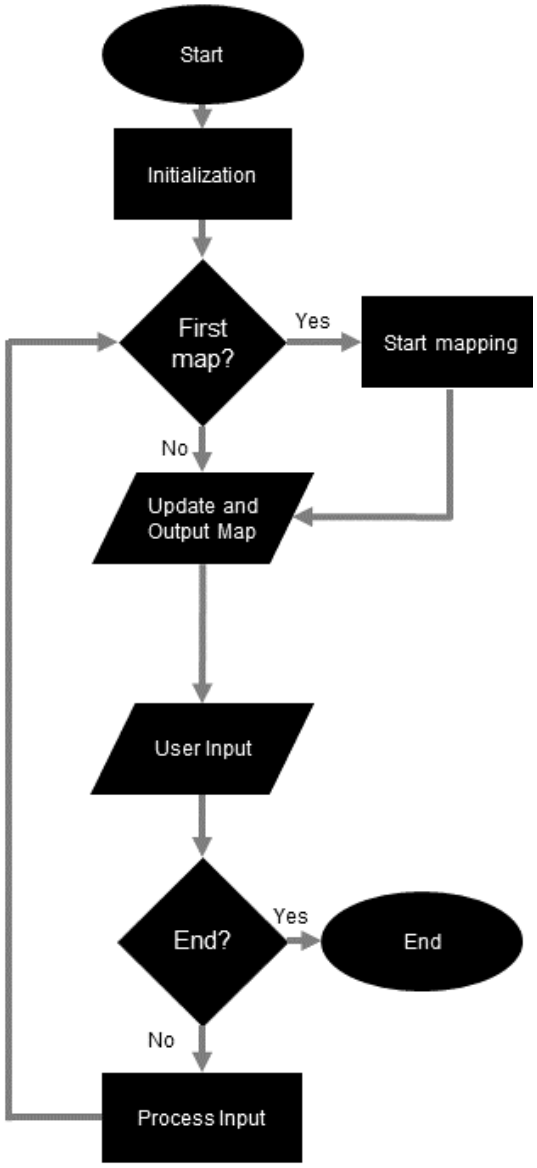

Figure 5: Overall Sequence of Events

High-Level Steps:

A: Initialization
B: Start / Update Mapping
C: Wait for User Input
D: Process Input

Go to either step B or End based on step D
End

A: Initialization

1. Test communication between Pi, Arduino and Laptop

    a. Ensure to set the PORT_NAME macro to the port where the Arduino is connected.
    b. Proceed to create a serial connection at 9600 bps.
    c. Pi creates a packet with message “Ready?”.
    d. Pi compiles message and serializes the Protocol Packet as a structure into a stream of bytes to Arduino.
    e. Arduino responds with an ACK package to Pi.
    f. The Arduino will then send back a packet to inform the Pi that it is done.
    g. The Pi implements Checksum to check that the data received is correct. The Pi will compare the Checksum from the sender side against an attached Checksum. If the data received matches correctly, the Pi will reply to the Arduino with an ACK packet. Else a NAK packet is sent instead.
    h. The Pi will deserialize the stream of bytes received by the Arduino back to structures, which is readable by the user.
    i. The user will receive an output message from Arduino titled “Yes”.
2. Test communication between Pi and Laptop (for ROS)
    a. Ensure the Pi and the laptop are connected to the same network
    b. Start a test program on the Pi that outputs messages to a topic
    c. Ensure that we are receiving the right messages on the Laptop’s end that is listening for the output messages at the topic.

B: Start / Update Mapping

1. For the first instance,
    a. Pi triggers Lidar to take the first set of readings.
    b. The Lidar takes readings and sends it to the Pi
    c. Lidar data is transmitted to the laptop using a ROS node
    d. Laptop runs HECTOR SLAM algorithm to generate the map
2. For updating the map,
    a. Pi triggers Lidar to take the readings
    b. Pi publishes Lidar readings to a Pi topic
    c. Laptop is running a ROS visualization node that is subscribed to the same topic that the Lidar data points are published to
    d. Laptop runs HECTOR SLAM algorithm to generate the map for user
3. Features of the map:
    a. Distances of walls
    b. Zero-degree marker (to know where the robot is facing)

Through this, we reduce CPU usage on Pi and this also allows for a higher refresh rate. Furthermore, we can process two maps on our laptop, one being the SLAM output and the other will be a graph of LiDAR points to see the environment better and to help us control the bot remotely. The dots will be in red so we can make it stand out and ensure that the points are clear. Finally, we are less reliant on latency stability as we are not transferring images from the Pi to our devices.

C: User Input

There will be a Graphical User Interface (GUI) for user to input actions (as shown in Figure 6)

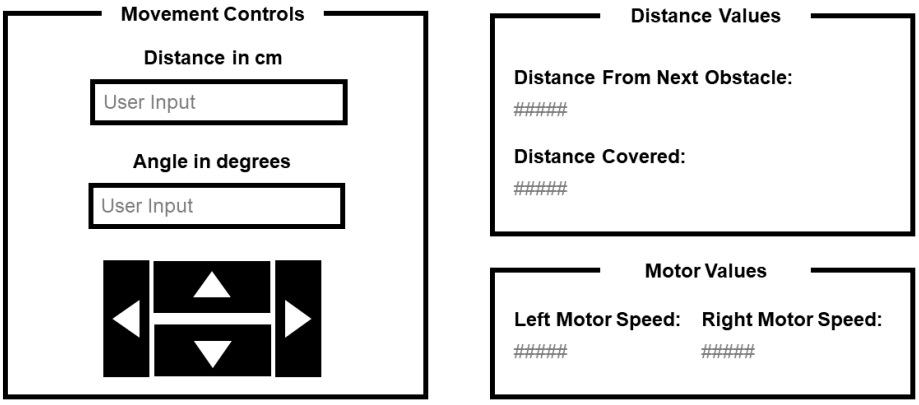

Figure 6: A Possible GUI layout

Using Ros Web Tools and ROSBridge we will be able to generate a webserver and we will use a node for the user interface that will publish actions to a topic and the Pi can listen for actions at that topic and also we can listen to messages from the Pi at specific topics.

Note, we also plan to enable keyboard mapping for the user controls such as forward, left, right and reverse. Apart from that, we will include a keyboard mapping for the "stop" command to stop all motors as a failsafe.

Distance covered:

1. Magnets are attached to the motors. Whenever the wheel completes one revolution, the encoders detects the falling edge of the end of the cycle, and causes an interrupt through the ISR.
2. For each revolution of the wheel, the interrupt records the respective direction (left/right) tick to increase by a certain amount.
3. Distance covered by one revolution of the wheel is recorded as WHEEL_CIRC cm angular distance.
4. When the forward function is called, distance travelled can be calculated using the following formula

$$\frac{(WHEEL\_CIRC \times total\ number\ of\ ticks)}{number\ of\ ticks\ for\ one\ wheel\ revolution}$$

D: Process User Input

The User will be able to:

a. Move forward by inputting a certain distance and clicking the up-arrow button as shown in Figure 6
b. Reverse by inputting a certain distance and clicking the down arrow button as shown in Figure 6
c. There are two options we would like to test out for turning function of Alex:
    i. input by x angles, then press left key once will cause Alex to turn left by the corresponding x degrees
        1. Turning algorithm: Refer to Appendix B.
        - A function called computeDeltaTicks is called, which computes an estimate of the number of wheel encoder ticks needed to turn Alex the angle specified by ang.
        - When Alex is moving in a straight line, the distance covered is WHEEL_CIRC cm forward (or backward) in one wheel revolution.

- To turn ang number of degrees, the number of wheel turns is given by the following formula, where AlexCirc is the circumference of the circle made by Alex turning on a dime

$$\frac{ang}{360.0} \times \frac{AlexCIRC}{WHEEL\_CIRC}$$

- The number of ticks is measured by

$$\frac{ang}{360.0} \times \frac{AlexCIRC}{WHEEL\_CIRC} \times COUNTS\_PER\_REV$$

- The user will simply decide on the angle at which the user wants Alex to turn through inputting a value into ang.

ii. press on left key to turn and hold until the desired angle then release. Although, theoretically, the problem with this method is that there might be delay when user clicks or releases the button, we hope to test this out to see if the user can train to work around the delay

d. Appropriate movement instruction is published on a ROS topic, the Pi listens to the topic and sends the instruction to the Arduino.
e. Arduino acknowledges instruction and executes instruction

Section 5: Project Plan

This section provides a week-by-week overview of our project and the milestones we hope to achieve the end of each week.

Table 2: Projected Timeline

Week	Milestones
8	1. Complete Design Report 2. Assemble Alex
9	1. Ensure all programmes (SSH and Arduino IDE) have been downloaded and tested. 2. Test run the implementation of the code on the various hardware components (Wheels, motor and encoder) to ensure that it is functional. 3. Gauge the proper estimates of the angle turned, distance travelled by Alex based on encoder readings. 4. Confirm that the basic movements of Alex are working (Left/Right turn, forward and reverse); Modify the code wherever needed. 5. Begin implementing our own Graphical User Interface to enhance the user experience Report Submission (16 March, Monday, 2359)
10	1. Integrate the LIDAR with Alex 2. Conduct test runs of the navigation of the environment with the LIDAR attached to Alex 3. Source for any areas of improvements CELC 1. Draft 1 Design Report (Submitted on 27 March, Friday, 2359)
11	1. Check accuracy of movement of Alex when it is remotely controlled. 2. Improve and modify the code to ensure that the environment is mapped out accurately by Alex. 3. Accuracy criteria: Ability of Alex to sense walls and objects and reproduce the distance accurately on the Laptop..
12	1. Enhance the performance of Alex by tweaking relevant hardware/software components. 2. Tackle power reduction challenge. Ensure that Alex is power efficient. 3. Finalise written report. CELC 1. Prepare a draft of Final Report (submit by 10 April, Friday, 2359.) For tutor's feedback.
13	Final Evaluation (DEMO + Presentation) Demo: 20%; Oral Presentation: 20%
RW	Submit Final Report (Submitted on 20 April, Monday, 2359)

References

[1] FavPNG, "FavPNG," [Online]. Available: https://favpng.com/png_view/flat-vector-laptop-laptop-icon-png/U0ErkhKA#. [Accessed 13 March 2020].

[2] FlatIcon, "FlatIcon," [Online]. Available: https://www.flaticon.com/authors/freepik. [Accessed 13 March 2020].

[3] Army Technology, "Army Technology," [Online]. Available: https://www.army-technology.com/projects/talon-tracked-military-robot/. [Accessed 6 March 2020].

[4] A. Etzioni and O. Etzioni, "Army University Press," May 2017. [Online]. Available: https://www.armyupress.army.mil/Journals/Military-Review/English-Edition-Archives/May-June-2017/Pros-and-Cons-of-Autonomous-Weapons-Systems/. [Accessed 6 March 2020].

[5] D. Quick, "News Atlas," 3 August 2011. [Online]. Available: https://newatlas.com/long-distance-tele-operation-of-ugvs/19423/. [Accessed 6 March 2020].

[6] J. Marsh, "EnergySage," 10 January 2019. [Online]. Available: https://news.energysage.com/lithium-ion-vs-lead-acid-batteries/. [Accessed 6 March 2020].

[7] P. Taylor-Parker, "Wholesale Solar," 23 October 2018. [Online]. Available: https://www.wholesalesolar.com/blog/lead-acid-vs-lithium-batteries. [Accessed 6 March 2020].

[8] CORDIS EU Research Results, "CORDIS EU Research Results," 5 December 2018. [Online]. Available: https://cordis.europa.eu/article/id/241251-stateoftheart-robot-to-better-assist-search-and-rescue-operations-in-lowvisibility-conditions. [Accessed 6 March 2020].

[9] Örebro Universitet, "Phys.org," 19 June 2018. [Online]. Available: https://phys.org/news/2018-06-smokebot-robot.html. [Accessed 6 March 2020].

[10] P. Fritsche, B. Zeise and P. Hemme, December 2017. [Online]. Available: https://www.smokebot.eu/deliverables/SmokeBot_D5.3.pdf. [Accessed 6 March 2020].

APPENDIX A

Figure A.1: TALON and the GUI
Source: Adapted from [3]

Figure A.2: SmokeBot
Source: Adapted from [8]

Appendix B

```
#include <math.h>

#define ALEX_LENGTH valuetobemeasured
#define ALEX_BREADTH valuetobemeasured
#define PI = 3.141592654

// compute & store this once
float AlexDiagonal = 0.0;
// turning circumference to be measured once
float AlexCirc = 0.0;

// Forward & Backward distances traversed
volatile unsigned long forwardDist;
volatile unsigned long reverseDist;
// Keep track if we have moved a commanded distance
unsigned long deltaDist;
unsigned long newDist;

// Keep track of turning angles
unsigned long deltaTicks;
unsigned long targetTicks;
void setup() {
   AlexDiagonal    =    sqrt((ALEX_LENGTH    *    ALEX_LENGTH)    +
(ALEX_BREADTH + ALEX_BREADTH));
   AlexCirc = PI * AlexDiagonal;
}
void forward(float dist, float speed) {
   // Code to tell us how far to move
  if (dist > 0)
     deltaDist = 9999999;
  else
     deltaDist = dist;

  newDist = forwardDist + deltaDist;
}
void reverse(float dist, float speed) {
   // Code to tell us how far to move
  if (dist > 0)
     deltaDist = 9999999;
  else
     deltaDist = dist;

  newDist = forwardDist + deltaDist;
}
unsigned long computeDeltaTicks(float ang) {
   unsigned long ticks = (unsigned long)((ang * AlexCirc *
COUNTS_PER_REV) / (360.0 * WHEEL_CIRC));
   return ticks;
}
void left(float ang, float speed) {
     int val = pwmVal(speed);
```

```
      dir = LEFT;

      if(ang == 0)
        deltaTicks = 9999999;
      else {
      deltaTicks = computeDeltaTicks(ang);
      }

      targetTicks = leftReverseTicksTurns + deltaTicks;
}
void right(float ang, float speed) {
      int val = pwmVal(speed);
      dir = RIGHT;

      if(ang == 0)
      deltaTicks = 9999999;
      else {
      deltaTicks = computeDeltaTicks(ang);
      }

      targetTicks = rightReverseTicksTurns + deltaTicks;
}

void loop() {
   if (deltaDist > 0) {
     if (dir==FORWARD) {
          if (forwardDist >= newDist) {
             deltaDist = 0;
          newDist = 0;
          stop();
          }
     }
     else if (dir==BACKWARD) {
          if (reverseDist >= newDist) {
               deltaDist = 0;
               newDist = 0;
               stop();
          }
     }
     else if (dir == STOP) {
          deltaDist = 0;
          newDist = 0;
          stop();
     }
   if (deltaTicks > 0) {
     if (dir==LEFT) {
          if (leftReverseTicksTurns >= targetTicks) {
          deltaTicks = 0;
          targetTicks = 0;
          stop();
          }
     }
     else if (dir==RIGHT) {
          if (rightReverseTicksTurns >= targetTicks) {
```

```
                deltaTicks = 0;
                targetTicks = 0;
                stop();
            }
        }
        else if (dir == STOP) {
            deltaTicks = 0;
            targetTicks = 0;
            stop();
        }
}
```