Recommendations for an Autonomous Cleaning Robot

RFP-ECE 338-202101

Submitted to:

Dr. Ilamparithi Thirumarai-Chelvan

Submitted by:

Aiden Jenner V00962200

John Burgess V00934703

Duncan McCracken V00959946

Aly Mooltazeem V00962689

March 17, 2021

Aiden Jenner, Aly Mooltazeem, Duncan McCracken, John Burgess
University of Victoria
P.O. Box 1700
Victoria B.C.
V8W 2Y2

March 17, 2021

Ilamparithi Thirumarai-Chelvan, Position
University of Victoria
P.O. Box 1700
Victoria B.C.
V8W 2Y2

Dear Dr. Ilamparithi Thirumarai-Chelvan

Please accept the following report entitled “Recommendation for an Autonomous Cleaning Robot”, responding to your request for a proposal. We are a team of four engineering students from the University of Victoria that have created three prototype robots to fulfill the needs of your request.

Inside we have attached an in-depth description of each of the prototype designs that include technical descriptions and a comparison chart between each of the designs. After comparing our prototypes we have concluded that the Shoddy Scanner is the prototype we recommend.

Due to COVID-19 we were unable to complete testing for the robots and we have included how we have overcome our, less than ideal, circumstances. Each of the robots still meet the needs and are more than capable of the tasks at hand.

We want to extend our thanks to you for taking the time to read our report and we look forward to hearing back from you. If you have any questions or concerns feel free to contact us at any time.

Sincerely,

Aiden, Aly, Duncan, John

Attachment - RecommendationsForAnAutonomousCleaningRobot.pdf

Table of Contents

List of Figures and Tables

Executive Summary

The University of Victoria is requesting an autonomous robot that is able to detect and eliminate COVID-19 from the Electrical Computer Engineering lab surfaces. In this report our members created three prototypes that fit the criteria supplied by the University. The criteria stated that the robot must search in the ‘x’ and ‘y’ direction, use an infrared radiation (IR) emitting device to eliminate the virus, scan a 15 centimetre by 10 centimetre area, scan various test squares, start and finish at the same position, only use VEX Robotics’ parts, and then cost less than $300 USD (excluding the microcontroller). The AA Robot has a rectangular base with a horizontal track that allows the robot's extendable arm to move in the ‘x’ direction. The extendable arm reaches out and in to eliminate the virus in the ‘y’ direction. The Shoddy Scanner has a C-shaped base and two perpendicular rails equipped with motors to shift a sensor and IR emitter along cartesian coordinates. The Minesweeper consists of 2 long rails on the ‘y’ and ‘x’ axis, with the scanner carriage being connected to both. By using motors attached to the rails, it allows the scanner to reach all possible coordinates in the possible search area. Using an IR scanner the robot scans the search area for dirty areas and cleans them.

Glossary

IR Sensor - is an electronic device, that emits in order to sense some aspects of the surroundings [2]

5-Bit Digital Counter - a device that stores the data from the IR sensor

Finite State Machine (FSM) - is a mathematical model of computation.

1. Introduction

Due to the impact of the COVID-19 pandemic, social distancing regulations have changed how students and professors interact with the university laboratories. The laboratories must be cleaned after each use with a minimal amount of students using the laboratory at a time. However, cleaning each station by hand would be time consuming and inefficient, as the stations must be cleaned in the time between classes which in most cases is 10 minutes. The Electrical and Computer Engineering department of the University of Victoria has released a Request for Proposals for autonomous robots that are capable of locating and disinfecting high-touch areas and workbench equipment, using radiation such as infrared light rays.[1] By using autonomous robots the laboratories are able to be sanitized quickly, and without the need for human interaction. As a response to this RFP our team has developed 3 prototypes that are ready for comprehensive testing and evaluation. These robots are capable of meeting the following objectives and constraints as stated in the RFP[1]:

- Objectives
    - Disinfect high-touch area (15cm x 10cm)
    - Disinfect all workstations.
- Constraints
    - Design should be built only using VEX robotics parts
    - The design should complete the task in under 10 minutes.
    - The design should cost under $300 (not including VEX kit)
    - The design should not tip over.

In order to test these objectives and evaluate our prototypes we have outlined a Weighted Objectives Matrix (WOM) later in the report that evaluates each robot and their test results against the others. This method gives a definitive answer about how each robot performs and meets the objectives and constraints as stated before.

2. Technical Description

Technical Description: Our team (Figure 1) built this robot using VEX parts [2]. This sits on top of the surface (i.e. white paper), where it scans. The robot consists of the following parts:

1. Base Structure
2. Horizontal Tracks
3. Motors, shafts & worm gear
4. Carriage for the IR sensor
5. Limit switches

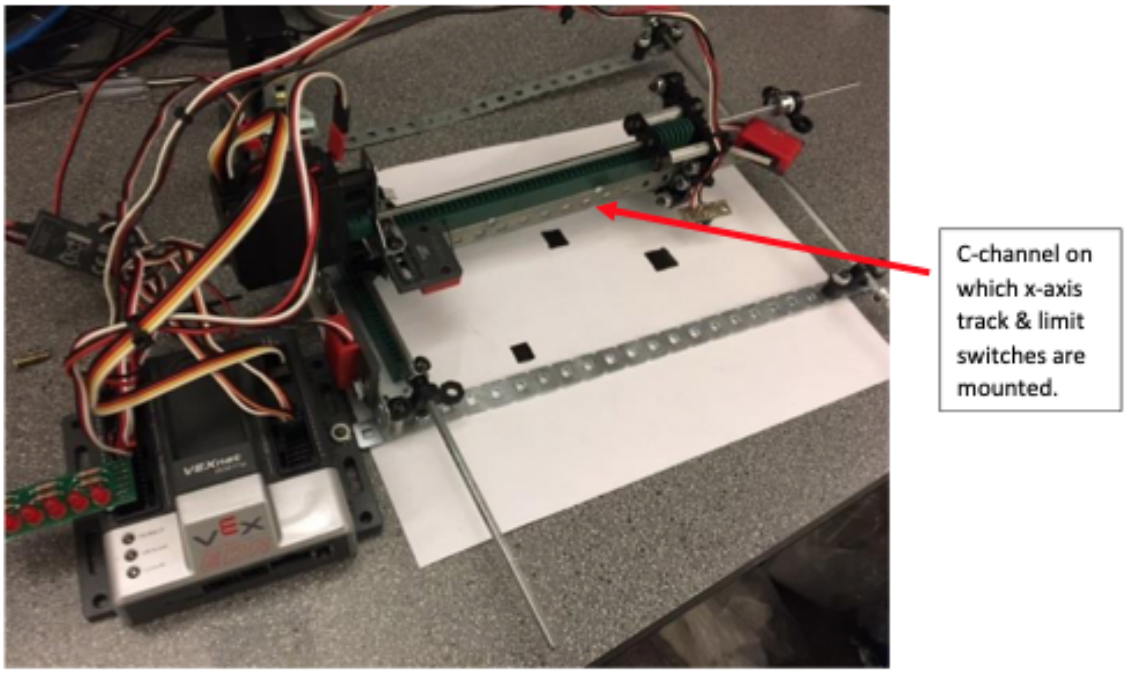


Fig. 1. Pre-built robot looking for black squares on a white paper

The base structure, built with C-channel, has two horizontal tracks. One for the X-axis (horizontal) and the other for Y-axis (vertical). The Y-axis angle has its standard orientation (i.e. the vertical part of the angle is pointing upwards) whereas the X-axis C-channel has its inverted orientation. Finally, we attached a long shaft at one end to support the angle for the X-axis track, to ensure level placement of the X-axis track.

On the X-axis and Y-axis tracks, the robot moves with the help of worm gears, driven by **motors**. The gussets hold two **motors** (x-axis and y-axis motor) mounted at one end of the X-axis and Y-axis frames. The carriage holds this worm gear of the X-axis track, as well as the IR sensor module.

The X-axis motor moves the carriage (as well as the sensor) to the right when it rotates in the positive direction . The Y-axis motor will carry the worm gear upwards when it is in the positive

direction. When the motor reverses direction, both of these motions also reverse. **Figure 2** below illustrates this motion.

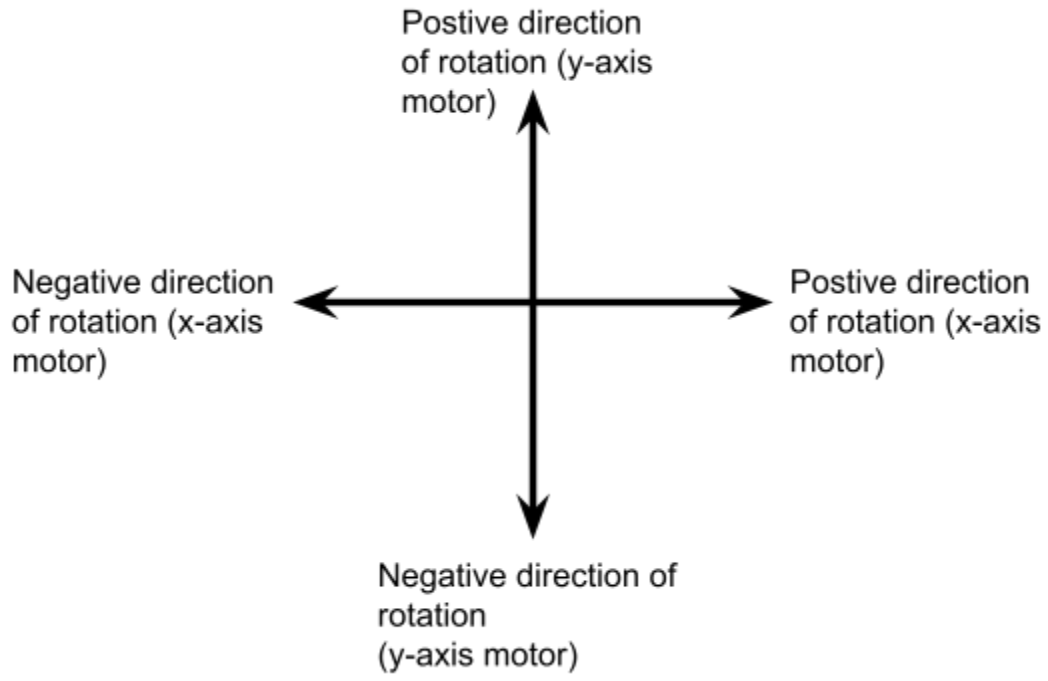


Figure 2: The graph showing the direction of linear movement of the robot by the motor

The microcontroller uses the Cortex variant. It has multiple ports that facilitate connection with sensors, buttons, switches, and motors. It also contains 6-LEDs which indicates successful detection and counting black squares. **Figure 3** shows the image of the Cortex based microcontroller.

This Cortex microcontroller consists of the following ports:

- 8 analog ports
- 12 digital ports
- 4 data ports
- 10 motor control ports

Figure 3: Image of Cortex based microcontroller used with the pre-built robot.

We have established the following connections, as listed below:

1. **I2C port** connected to **X-axis motor**, then daisy-chained to **Y-axis motor**
2. **X-axis motor** connected to **Motor Port 2**
3. **Y-axis motor** connected to **Motor Port 3**
4. **Y-Min** (Y-axis minimum position switch) connected to **Digital Port 1**
5. **Y-Max** (Y-axis maximum position switch) connected to **Digital Port 2**
6. **X-Min** connected to **Digital Port 3**
7. **X-Max** connected to **Digital Port 4**
8. Optical transceiver (IR sensor's output part) connected to **Analog Port 1**
9. 6-LEDs connected to **Digital Ports 6-11**

2.1 AA Robot

The Robot begins in a motionless state in the origin (0,0) position. Starting at the minimum 'X' position, the robot moves forward in the 'X' direction and goes until it pushes the maximum 'X' limit. Once pushed, the program triggers the robot to move down 1 'Y' unit, based on a timer configuration. Once the robot moves down for 1 second, the robot begins to move backwards in the 'X' direction. If the robot moves down and triggers the limit switch, it turns the robot into the 'off' state. The motor causes the scanner to move backwards in the 'X' direction from the maximum, to the minimum 'X' location. Once the minimum 'X' limit switch is pushed in, the robot will move down for 1 second again in the 'Y' direction, stop and then proceed to go in the 'X' direction once more. The cycle continues until one of the limits is triggered after the 'Y' direction is reached. If the robot reaches the 'Y' limit switch, it will return to its original starting position and turn to the 'off' state. This will end the process.

Throughout the process the robot uses an **IR sensor** to detect black squares and capture the information. As the IR sensor scans the area it progressively stores digital values between 0 and 4095. If the value reaches above 2300 (threshold value) then it will store the information as a black square and collect that data using a LED bar. This bar acts as a **5-bit digital** counter to count the amount of black squares detected. Depending on how many squares are detected the LED lights flash accordingly.

Figure 4 below shows an example of a path the robot may take to scan the surface of the paper.

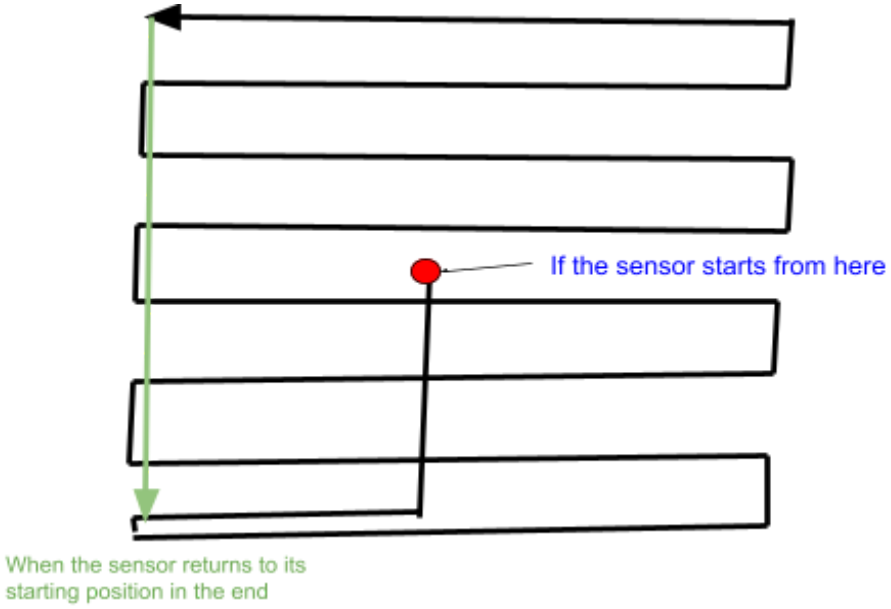
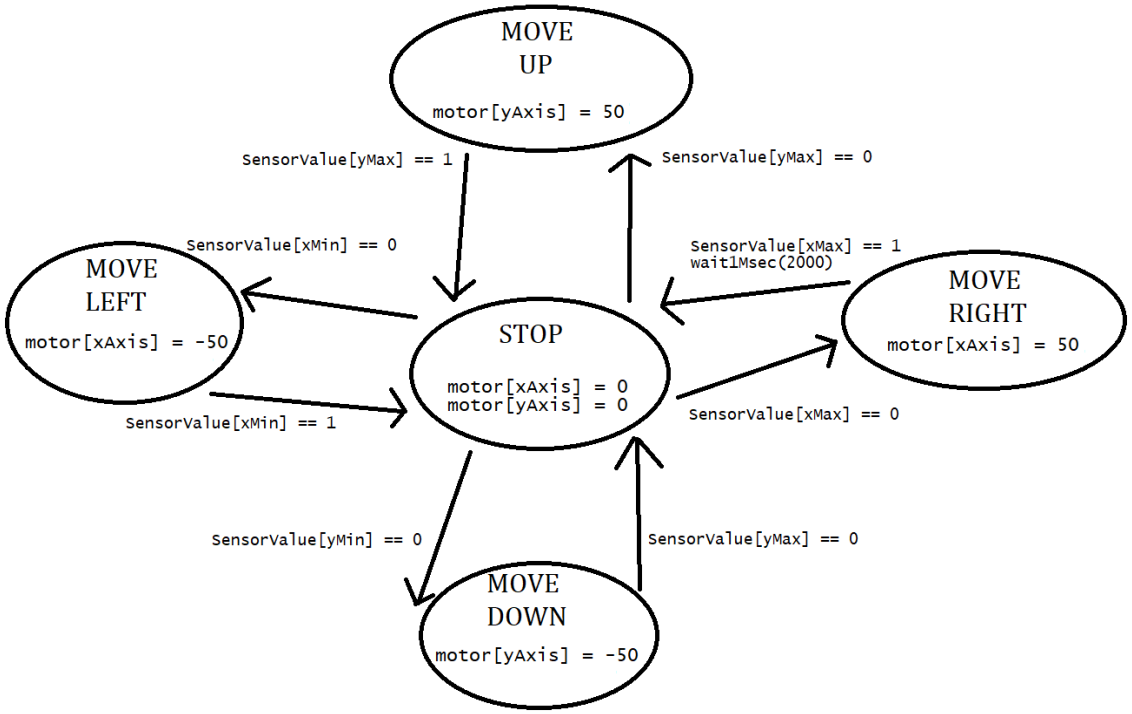


Figure 4: Example scanning path

2.2 Minesweeper

In our program design, the robot begins by returning itself to the (0,0) coordinates at the bottom left of the given scanning area, by using the limit switches to indicate when it returns to the origin. To scan the search area, the program begins by moving the scanner in the positive Y direction, scanning the underlying surface for black squares. Once the robot triggers the limit switch at the top of the Y axis, the program moves the scanner a small distance to the right in the X axis. The robot then proceeds to move the scanner in the negative Y direction until it hits the limit switch at the bottom of the Y axis. This process continues until the entire searchable area is scanned. In order to detect the black squares in the search area, an IR sensor mounted to the robot scans below it, returning a value between 0 and 4095. In our program, a value above 2800 counts as a black square. To reduce the double scanning of the same square, the scanner must detect white, or less than 2800 in the IR sensor, before it will register another square into the counter keeping track of the number of squares. Once the robot triggers both the limit switch at the top of the Y axis and the limit switch at the far right of the X axis, the robot then displays the number of black squares counted by flashing an LED as many times as there are squares. If there were 10 squares, the LED would blink 10 times. Below is a **Finite State Diagram** of the programming used in our robot.



Robot's scanner carriage starts motionless and moves all the way left, then all the way down to starting position.
To search the area, it
Moves to the top along the y-axis, then right by 1 unit.
Moves to the bottom along y-axis, then right by 1 unit.
This is repeated until the sensor reaches as far right as possible.

Figure 5: Finite State Diagram for Minesweeper

2.3 Shoddy Scanner

This robot starts at the top left of the scannable area and starts scanning along the X axis (horizontal axis) until it reaches the right side. It then moves down along the Y axis for 1.5 seconds and begins scanning along the X axis in the opposite direction. This process repeats until the robot reaches the bottom Y limit, where it will scan along the X axis one final time before it stops.

The robot detects the black squares by constantly reading the colour values underneath the IR sensor. The colour values are interpreted as digital values between 0 and 4095. A received digital value over 3000 corresponds to a colour of black below the IR sensor. To reduce false positives during scanning, a 1 second timer starts after every detection of black. The counter increments by one only if a black reading remains after 1 second. This 1 second timer can be adjusted based on the speed of the robot. An increased scanning speed would correspond to a shorter window of time where the robot detects a black square; therefore, the timer would have to be shorter as well. The leftmost LED on the LED bar serves as the indicator. Every time the robot detects a black square, the indicator will periodically light up. The 5 other LEDs on the LED bar serve as a **5-bit digital counter** to display the total count of detected squares; this counter updates every time the indicator lights up. Below is a FSM of the Shoddy Scanner.

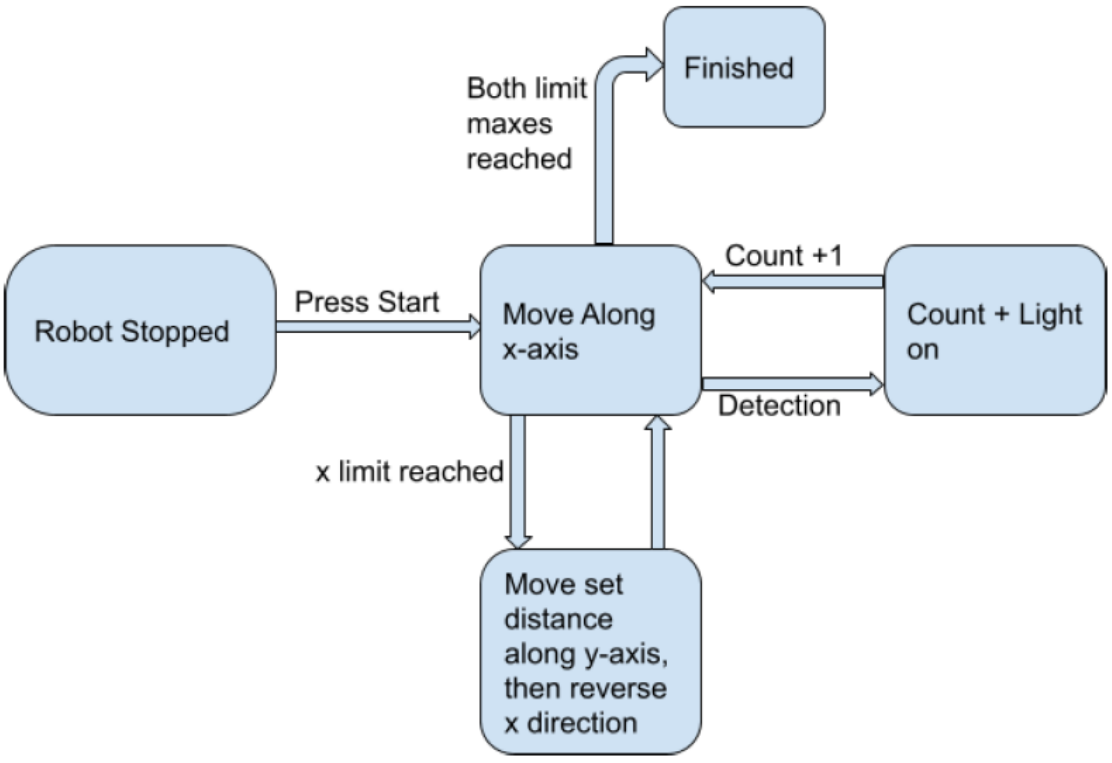


Figure 6: Finite State Diagram for Shoddy Scanner

3. Analysis of Designs

In order to accurately evaluate our designs we have developed 6 different criteria based on importance:

1. Build Quality
2. Completion Time
3. Accuracy
4. Overall Cost
5. Energy Consumption

In our testing, we found that build quality was the most important aspect for a successful robot under these conditions. As these robots need to complete their tasks remotely without human intervention it is paramount that these robots fail or have mechanical issues very rarely. As if they break down a human is required to either repair the robot or replace it entirely. If a robot has a very high build quality it will be able to run continuously without breaking down for a very long time without human intervention. Whereas if the build quality is very low, then the robot will break often and require human intervention very often.

Time taken to complete the task of cleaning the workstations is also one of the most important factors in determining whether one robot is better than another. These robots need to be able to clean the workstations in a limited amount of time. For example, they would have to complete the task in under 10 minutes so that they can be used between lab sessions for the ECE department. If the robot completes the task quickly, then it will be able to be used more times and within less downtime. If it completes the task very slowly, it might not be able to clean the workstation fully by the time the next lab session begins. Therefore it is of utmost importance that the robot is able to clean the area quickly.

The accuracy of the robots is also an important metric for comparing the different robots. In order to clean the workstations efficiently, the robots need to be able to accurately detect the contaminated area of the workstation and disinfect the area without cleaning extraneous parts of the work area. If the robot is more accurate then it will be able to clean the area more efficiently, if it is less accurate then it will take longer and will be more prone to errors in the cleaning program. Being accurate will make the robots faster and less prone to mechanical faults.

Cost was mentioned by the ECE department as to not exceed $300 USD. Cost is a relatively important factor considering that these designs are for use in the public sector. However not as important as other factors listed as costs can change over time due to varying circumstances, for example supply, and newer versions of robots. The cost breakdown below outlines the overall cost for the robots. As each group used the same basic design of robot, the same overall cost is used for the testing of each robot.

Table 1: Cost Breakdown

Item	Price	Quantity	Cost	
VEX Starter Kit	$249.99	1	$249.99	Note: This is not included with the total cost, as the RFP states is it provided
Advanced Gear Kit	$29.99	1	$29.99	
Steel Rail Kit	$22.99	2	$45.98	
Limit Switches (2-pack)	$19.99	2	$39.98	
IR Sensor	$19.99	1	$19.99	
Total			**$135.94**	

Energy consumption also factors into the choice between the different designs. If a robot draws too much power it would be inefficient to run constantly between lab sessions. This factor is less important in a university setting where the power used by the robots creates less of an impact on the cost.

These criteria have been developed through extensive testing with the given robots in order to best represent the different aspects that create a successful design. Included in our Testing Results section is our completed WOM with our values from testing our designs. See Appendix B for a more detailed breakdown of the parameters of each criteria.

4. Testing Results

The testing of each robot resulted in the WOM shown below in **Table 1**.

Table 2: Weighted Objectives Chart

Criteria	Weight (score/10)	% Weight	AA Robot		Minesweeper		Shoddy Scanner	
			Score	Value	Score	Value	Score	Value
Build quality	10	0.256	1	0.256	1	0.256	1	0.256
Completion time	9	0.231	3	0.692	5	1.154	4	1.154
Overall cost	7	0.179	5	0.897	5	0.897	5	0.897
Accuracy	9	0.231	5	1.154	2	0.462	5	1.154
Energy consumption	4	0.103	4	0.410	3	0.308	5	0.513
Final Value				3.410		3.077		3.7444

From our testing, the Shoddy scanner received the highest score based on our criteria and parameters. While it took longer than other designs to complete the task, this design was highly accurate and had a very low energy consumption compared to the other designs. The high accuracy was a result of careful programming and optimization from the group that designed it. The low energy consumption was also a result of this as it was able to complete the task in a very efficient manner leading to low overall energy costs. High scores in these two criteria were the reason why the Shoddy Scanner ended up receiving the highest total score of our designs. In practice, this design proves most useful, as it completes the task with a high accuracy, with a low impact on energy usage, while also completing the task in a reasonable time frame.

Recommendation

Each robot proved to successfully complete the objectives. The Minesweeper had the fastest completion time but had issues with accuracy. The AA Robot was accurate and cost effective, but consumed more energy due to high repetition of tasks. The Shoddy Scanner did not complete the fastest, yet it was the most optimized and consistent, earning it the highest score. As a result, the Shoddy Scanner is the robot we recommend.

Our criteria was based on the importance we felt was highest to the client. The Shoddy Scanner is the most tested design and was able to exceed the other designs. This compacted robot design fully meets and exceeds expectations and will work in most situations it is put in. All in all, after discussion and testing, the Shoddy Scanner fits the ECE lab expectations best. We suggest the ECE lab moves forwards with the following steps if the robot is approved:

1. Immediate contact with our team stating their approval
2. Create a contract between the two parties
3. Sketch up a project timeline
4. Create tangible objectives that the robot can meet
5. Include any modifications if necessary
6. Work alongside our team to finalize the robot

References

[1] T. Ilamparithi, “REQUEST FOR PROPOSALS .” Department of Electrical & Computer Engineering, University of Victoria, Victoria, BC, Sep-2021.

[2] *VEX Robotics*. [Online]. Available: https://www.vexrobotics.com/. [Accessed: Feb. 25, 2021].

[3] Agarwal, T. (2020, November 16). *IR Sensor : Circuit Diagram, Types Working with Applications*. ElProCus - Electronic Projects for Engineering Students. https://www.elprocus.com/infrared-ir-sensor-circuit-and-working/#:%7E:text=An%20infrared%20sensor%20is%20an,called%20a%20passive%20IR%20sensor.

[4] Wikipedia contributors. (2021, April 3). *Finite-state machine*. Wikipedia. https://en.wikipedia.org/wiki/Finite-state_machine

Appendices

Appendix A: Work Logs

The Gantt chart below shows each task being allocated to a specific amount of time:

Table 3: Gantt Chart of project schedule

Tasks	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
CAD Design												
Finite State Machines												
Programming												
Proposal Document												
Progress Report												
Project Preparation												

Appendix B: Objectives Scoring Tables

Table 4: Build Quality Parameters

Parameters	**Score**
less than 50% success rate	0
50% to 60% success rate	1
60% to 70% success rate	2
70% to 80% success rate	3
80% to 90% success rate	4
90% to 100% success rate	5

Table 5: Completion Time Parameters

Parameters	**Score**
<10mins	5
10mins - 15mins	4
15mins - 20mins	3
20mins - 25mins	2
25mins -30mins	1
>30mins	0

Table 6: Overall Cost Parameters

Parameters	**Score**
cost<$150	5
150<cost<200	4
200<cost<250	3
250<cost<300	2
cost>300	1

Table 7: Energy Consumption Parameters

Parameters	Score
<300W	1
250W<E<300W	2
200W<E<250W	3
150W<E<200W	4
150W<E<200W	5

Robot Code

```
#pragma config(Sensor, in1,    IROut,          sensorReflection)
#pragma config(Sensor, dgtl1,  yMin,           sensorTouch)
#pragma config(Sensor, dgtl2,  yMax,           sensorTouch)
#pragma config(Sensor, dgtl3,  xMin,           sensorTouch)
#pragma config(Sensor, dgtl4,  xMax,           sensorTouch)
#pragma config(Sensor, dgtl6,  Red1,           sensorLEDtoVCC)
#pragma config(Sensor, dgtl7,  Red2,           sensorLEDtoVCC)
#pragma config(Sensor, dgtl8,  Red3,           sensorLEDtoVCC)
#pragma config(Sensor, dgtl9,  Red4,           sensorLEDtoVCC)
#pragma config(Sensor, dgtl10, Red5,           sensorLEDtoVCC)
#pragma config(Sensor, dgtl11, Red,            sensorLEDtoVCC)
#pragma config(Motor,  port2,           xMotor,
tmotorServoContinuousRotation, openLoop)
#pragma config(Motor,  port3,           yMotor,
tmotorServoContinuousRotation, openLoop, reversed)
//*!!Code automatically generated by 'ROBOTC' configuration wizard
!!*//

int results(int squares);
void moveRight();
void moveLeft();
void moveUp();
void moveDown();
void startPos();
void startPos2();
void scanSameSurface(int num);


int turnVar = 0;
int threshold = 2300;
int squares = 0;
int speed = 125;
int array[15] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

task main()
{


      //startPos2();

      SensorValue[Red] = 0;
      SensorValue[Red1] = 0;
```

```
SensorValue[Red2] = 0;
SensorValue[Red3] = 0;
SensorValue[Red4] = 0;
SensorValue[Red5] = 0;
int temp = 0;
int row = 0;
int rowsWithSquares = 0;



startPos();

while (SensorValue[yMax] == 0) {


      if (SensorValue[xMin] != 0) {
            turnVar = 1;
      }
      if (SensorValue[xMax] != 0) {
            turnVar = 2;
      }

      temp = squares;

      switch (turnVar) {
            case 1:
            moveRight();
            if (squares > temp) {
                  rowsWithSquares++;
                  array[row] = 1;
            }
            row++;


            break;

            case 2:
            moveLeft();

            if (squares > temp) {
                  rowsWithSquares++;
                  array[row] = 1;
            }
            row++;
```

```
                break;

                default:
                break;
          }
          moveUp();

     }
     moveRight();

     bool sameSurface = true;

     startPos();

     wait1Msec(5000);

     if (sameSurface == true){
          scanSameSurface(rowsWithSquares);
     }


}


void moveRight() {
     while(SensorValue[xMax] == 0) {
                    motor[xMotor] = speed;

                    /*if (SensorValue[IROut] == threshold){
                         squares++;
                         results(squares);
                    }
                    */
                    if (SensorValue[IROut] > threshold) {
                         //clearTimer(T2);
                         while(SensorValue[IROut] > threshold){
                              SensorValue[Red] = 1;
                         }
                         SensorValue[Red] = 0;
                         wait1Msec(3000);
                         squares = squares + 1;
                }
                    results(squares);

                }
```

```
                motor[xMotor] = 0;
                wait1Msec(1000);
                turnVar = 0;
}

void moveLeft() {
while(SensorValue[xMin] == 0) {
                    motor[xMotor] = -speed;

                    /*if (SensorValue[IROut] == threshold){
                         squares++;
                         results(squares);
                    }
                    */
                    if (SensorValue[IROut] > threshold) {
                         //clearTimer(T2);
                         while(SensorValue[IROut] > threshold){
                              SensorValue[Red] = 1;
                         }
                         SensorValue[Red] = 0;
                         wait1Msec(3000);
                         squares = squares + 1;
                    }
                    results(squares);

                }

                motor[xMotor] = 0;
                wait1Msec(1000);
                turnVar = 0;
}

void moveUp() {
     clearTimer(T1);
          while (time1[T1] < 1000) {
               motor[yMotor] = speed;
          }
          motor[yMotor] = 0;

          wait1Msec(500);
}

void moveDown() {
     clearTimer(T1);
          while (time1[T1] < 900) {
```

```
                motor[yMotor] = -speed;
           }
           motor[yMotor] = 0;

           wait1Msec(500);
}

void startPos() {
while (SensorValue[yMin] ==0) {
           motor[yMotor] = -speed;
     }
     motor[yMotor] = 0;

     while (SensorValue[xMin] == 0) {
           motor[xMotor] = -speed;
     }
     motor[xMotor] = 0;
     motor[yMotor] = 0;
     wait1Msec(1000);
}

void startPos2() {
while (SensorValue[yMax] ==0) {
           motor[yMotor] = speed;
     }
     motor[yMotor] = 0;

     while (SensorValue[xMin] == 0) {
           motor[xMotor] = -speed;
     }
     motor[xMotor] = 0;
     motor[yMotor] = 0;
     wait1Msec(1000);
}

int results(int squares) {
     if (squares == 1) {
                                SensorValue[Red1] = 1;
                                SensorValue[Red2] = 0;
                                SensorValue[Red3] = 0;
                                SensorValue[Red4] = 0;
                                SensorValue[Red5] = 0;
                           }

                      if (squares == 2) {
```

```
            SensorValue[Red1] = 0;
            SensorValue[Red2] = 1;
            SensorValue[Red3] = 0;
            SensorValue[Red4] = 0;
            SensorValue[Red5] = 0;
      }

if (squares == 3) {
            SensorValue[Red1] = 1;
            SensorValue[Red2] = 1;
            SensorValue[Red3] = 0;
            SensorValue[Red4] = 0;
            SensorValue[Red5] = 0;
      }
if (squares == 4) {
            SensorValue[Red1] = 0;
            SensorValue[Red2] = 0;
            SensorValue[Red3] = 1;
            SensorValue[Red4] = 0;
            SensorValue[Red5] = 0;
      }

if (squares == 5) {
            SensorValue[Red1] = 1;
            SensorValue[Red2] = 0;
            SensorValue[Red3] = 1;
            SensorValue[Red4] = 0;
            SensorValue[Red5] = 0;
      }
if (squares == 6) {
            SensorValue[Red1] = 0;
            SensorValue[Red2] = 1;
            SensorValue[Red3] = 1;
            SensorValue[Red4] = 0;
            SensorValue[Red5] = 0;
      }

      if (squares == 7) {
            SensorValue[Red1] = 1;
            SensorValue[Red2] = 1;
            SensorValue[Red3] = 1;
            SensorValue[Red4] = 0;
            SensorValue[Red5] = 0;
      }
```

```
if (squares == 8) {
     SensorValue[Red1] = 0;
     SensorValue[Red2] = 0;
     SensorValue[Red3] = 0;
     SensorValue[Red4] = 1;
     SensorValue[Red5] = 0;
}
if (squares == 9) {
     SensorValue[Red1] = 1;
     SensorValue[Red2] = 0;
     SensorValue[Red3] = 0;
     SensorValue[Red4] = 1;
     SensorValue[Red5] = 0;
}

if (squares == 10) {
     SensorValue[Red1] = 0;
     SensorValue[Red2] = 1;
     SensorValue[Red3] = 0;
     SensorValue[Red4] = 1;
     SensorValue[Red5] = 0;
}

if (squares == 11) {
     SensorValue[Red1] = 1;
     SensorValue[Red2] = 1;
     SensorValue[Red3] = 0;
     SensorValue[Red4] = 1;
     SensorValue[Red5] = 0;
}

if (squares == 12) {
     SensorValue[Red1] = 0;
     SensorValue[Red2] = 0;
     SensorValue[Red3] = 1;
     SensorValue[Red4] = 1;
     SensorValue[Red5] = 0;
}

if (squares == 13) {
     SensorValue[Red1] = 1;
     SensorValue[Red2] = 0;
     SensorValue[Red3] = 1;
     SensorValue[Red4] = 1;
     SensorValue[Red5] = 0;
```

```
                }

                if (squares == 14) {
                      SensorValue[Red1] = 0;
                      SensorValue[Red2] = 1;
                      SensorValue[Red3] = 1;
                      SensorValue[Red4] = 1;
                      SensorValue[Red5] = 0;
                }

                if (squares == 15) {
                      SensorValue[Red1] = 1;
                      SensorValue[Red2] = 1;
                      SensorValue[Red3] = 1;
                      SensorValue[Red4] = 1;
                      SensorValue[Red5] = 0;
                }

                if (squares == 16) {
                      SensorValue[Red1] = 0;
                      SensorValue[Red2] = 0;
                      SensorValue[Red3] = 0;
                      SensorValue[Red4] = 0;
                      SensorValue[Red5] = 1;
                }
                return squares;
           }

void scanSameSurface(int num) {

squares = 0;
SensorValue[Red] = 0;
SensorValue[Red1] = 0;
SensorValue[Red2] = 0;
SensorValue[Red3] = 0;
SensorValue[Red4] = 0;
SensorValue[Red5] = 0;
int rowNum = 0;
int switchVar = 0;
     startPos();

     for (int i = 0; i<num; i++) {

     if (SensorValue[xMin] != 0) {
          switchVar = 1;
```

```
	}
	if (SensorValue[xMax] != 0) {
		switchVar = 2;
	}

	switch (switchVar) {
		case 1:
		moveRight();
		rowNum++;
		break;

		case 2:
		moveLeft();
		rowNum++;
		break;

		default:
		break;
	}

		moveUp();
		if (array[rowNum] == 0) {
			moveUp();
			rowNum++;
		}
	}
	moveRight();
	startPos();
}
```