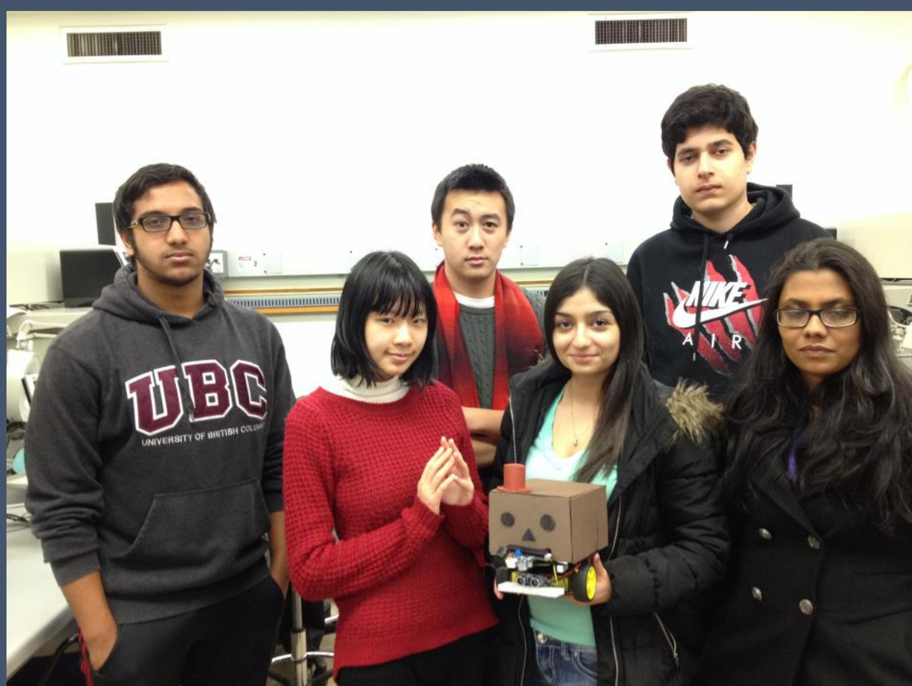
Autonomous Cleaning Robot



EECE 281 Section 202

Project 1

Lab section: L2C Group #: 6 Group's Lab-Bench #s: 4B and 5B

Student names, numbers and contribution percentages:

Catherine Lee	32556136	100/6 %
Sangeetha Kamath	49878135	100/6 %
Syed Mubashir Iqbal	47474135	100/6 %
Brian Chang	48671127	100/6 %
Srinjoy Chakraborty	14528137	100/6 %
Rosa Mohammadi	31856131	100/6 %

TABLE OF CONTENTS

1.0. CONTRIBUTIONS

Name	Tasks	Contribution
Catherine Lee	Wrote report, restructured code, prepared base charts, listed tasks to complete, created diagrams for report, switch between basic and additional functionality.	**100/6**
Sangeetha Kamath	Assisted with basic functionality implementation, wired LCD and temperature sensor, re-assembled robot to support second range sensor and light sensors.	**100/6**
Syed Mubashir Iqbal	Assisted with basic functionality implementation, expanded on area-calculating functionality, wrote maximum area coverage algorithm, created block diagram, calibrated right angle turns.	**100/6**
Brian Chang	Purchased components, soldering connections, assembled hardware, wired motors, connected external power source, power readings.	**100/6**
Srinjoy Chakraborty	Suggested area-calculating functionality and point-slope formula to reduce speed, calibrated right angle turns.	**100/6**
Rosa Mohammadi	Wrote and documented code, assisted with basic functionality implementation, researched IR sensing and line-following algorithm, made Fritzing schematic, created diagrams for report, revised report.	**100/6**

Figure 1.1: Contribution Summary

2.0. INTRODUCTION AND MOTIVATIONS

When we began this project, our aim was to create an impressive, user-oriented device. We were filled with ideas and very excited to create an excellent remote-controlled device. As we continued, we found that many of our ideas and plans translated to unrealistic goals with the given time, and continued to redefine our objectives and specifications.

Eventually, we decided on simplifying our ideas into a single cleaning robot. With bristles attached, this robot would start by cleaning one side of the area in random paths, then proceed to clean the other half of the area. If the robot is on a raised surface, it can also determine that it is approaching an edge, and resume its normal motion. Due to hardware limitations, both types of sensing are limited, but still perform both functionalities.

We have attached several files, which are listed below.

File Name	Description
cleaningRobot.ino	The complete Arduino code, consisting of all the functionality described in section 3.0.
cleaningRobot.fzz	The Fritzing Schematic corresponding to our project's wiring. Note: Schematic pins are not necessarily the same pins as those used in the code, but may easily be adjusted to follow our coded pins.

Figure 2.1. Project Attachments

3.0. PROJECT DESCRIPTION

Our robot is a cleaning robot that covers the maximum area on a surface while preventing itself from falling off edges. It covers maximum area by first focusing on one half of the surface, then focusing on the other half. Inspired by the iRobot Roomba, it moves along random paths, which ultimately produces the most effective algorithm. To accomplish this functionality, we connected a couple range sensors and photocells to alternate between the basic and additional functionality (Figure 3.1).

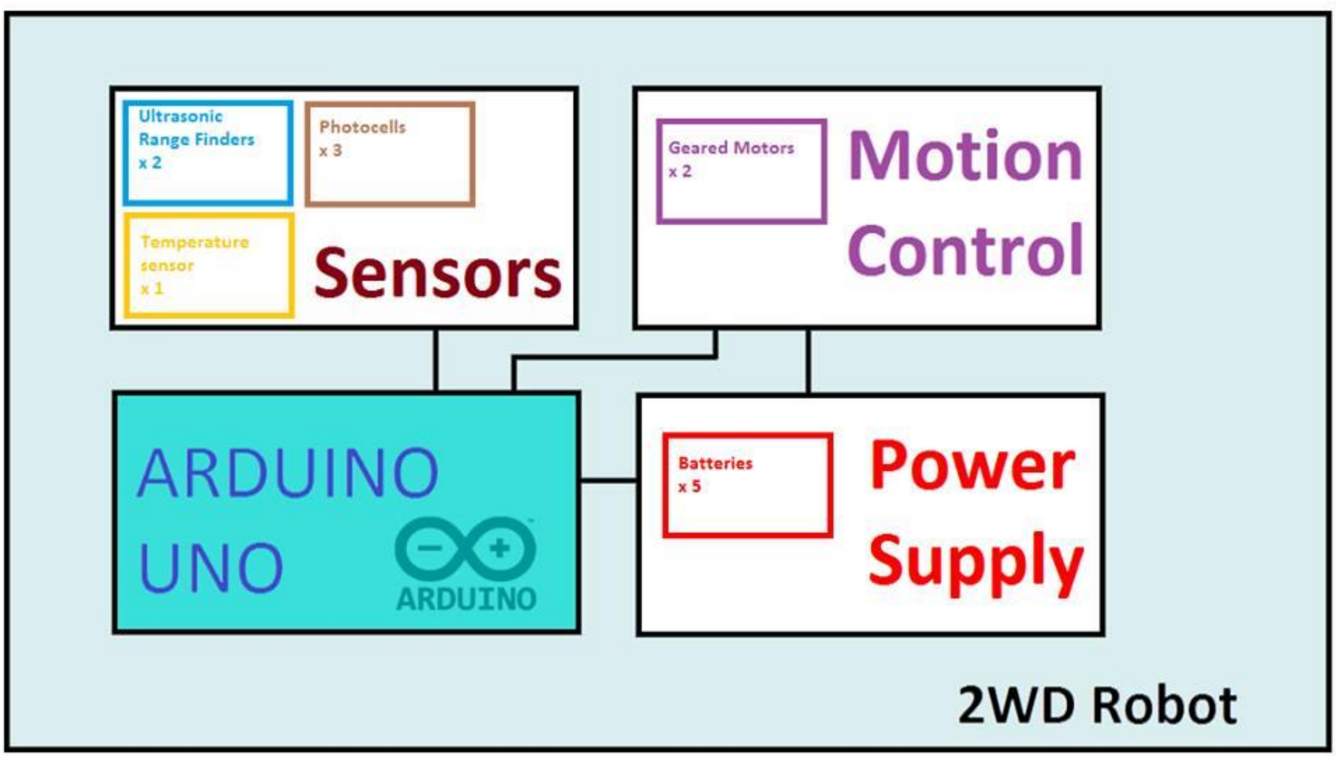
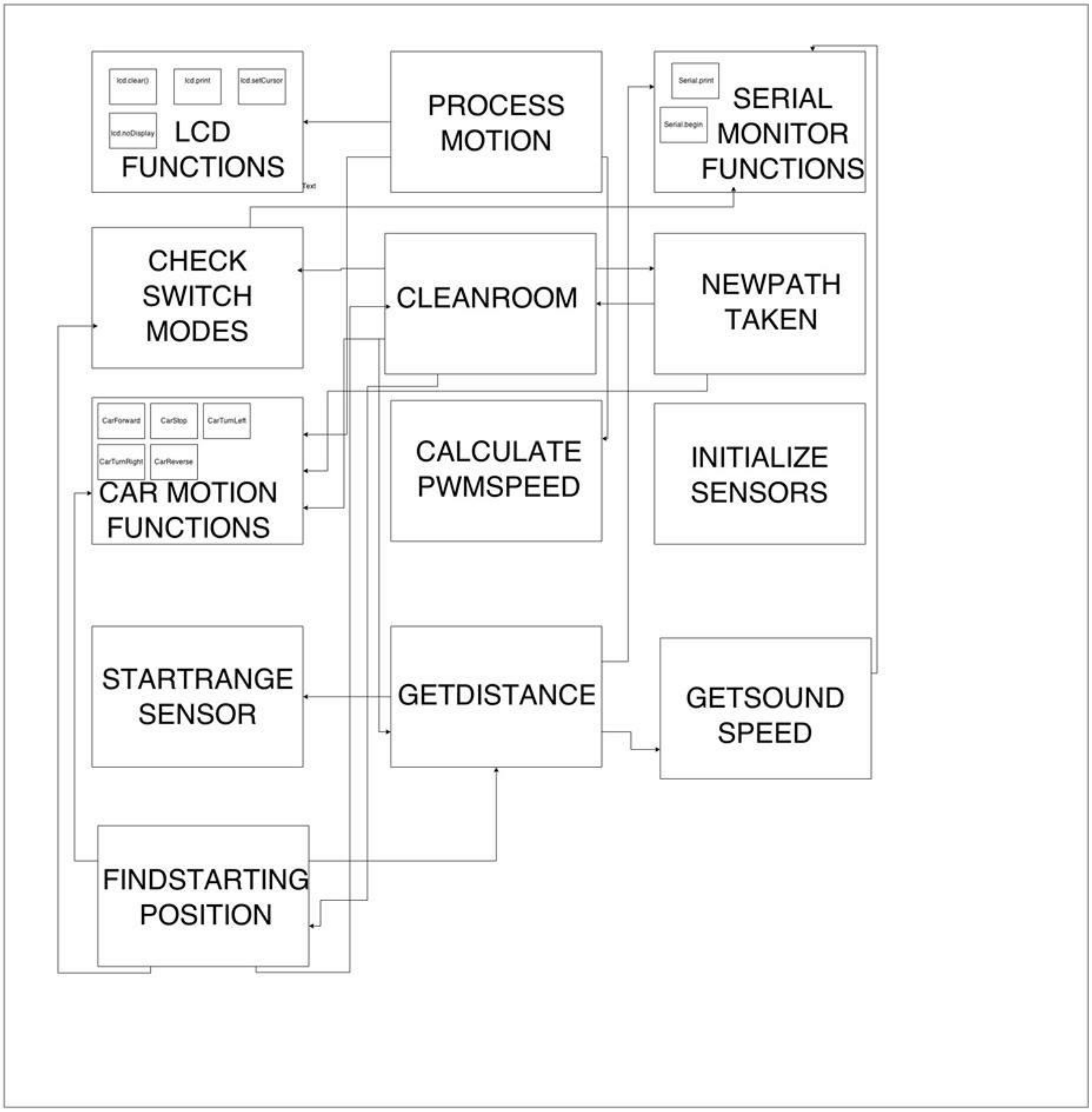


Figure 3.1. Hardware System Block Diagram


lcd.clear()
lcd.print
lcd.setCursor
lcd.noDisplay
LCD FUNCTIONS
Text
PROCESS MOTION
Serial.print
Serial.begin
SERIAL MONITOR FUNCTIONS
CHECK SWITCH MODES
CLEANROOM
NEWPATH TAKEN
CarForward
CarStop
CarTurnLeft
CarTurnRight
CarReverse
CAR MOTION FUNCTIONS
CALCULATE PWMSPEED
INITIALIZE SENSORS
STARTRANGE SENSOR
GETDISTANCE
GETSOUND SPEED
FINDSTARTING POSITION

3.1. Design and Implementation Procedures

3.1.1 Assembling the Robot and Testing Components

Following the Motor Shield Wiki [1] and the Mobile Platform Assembly Manual [2], we assembled the outer structure of the robot. Afterwards, we connected the motors to the Arduino (Figure A1) and used the code from the wiki [1] to test their motion. The motors moved, confirming that we were receiving instructions from the Arduino. Then, we then connected the motors to the robot, rewired the motors and observed that the device moved left, right, then forwards and backwards. From there, we connected a range sensor (Figure 3.2) and created a simple if-else loop (Figure A2) to move the car based on data sensed by the attached range sensor, based on our functions from Lab 5.

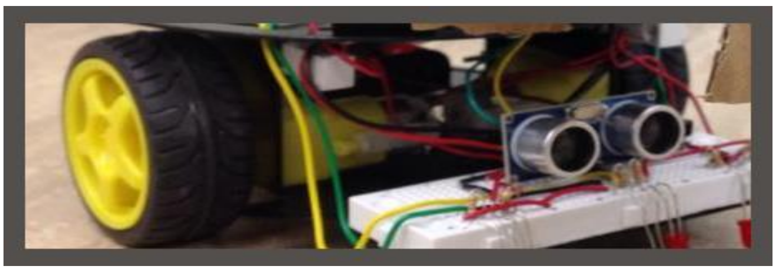


Figure 3.2. Range sensor placed in front of robot to sense objects ahead of it

To easily connect components to ground and power, we removed the power bus and ground bus on the large breadboard and attached them to the inner sides of the robot, near the motors, using double sided tape (Figure A3).

3.1.2 Using an External Power Source

To allow the robot to function without a USB cable, we moved the black plastic covers of the jumper cables (Figure A4) of the Arduino input pins to place the power source in series with the Arduino and other attached components. Then, the components were connected to ground through the negative terminal of the batteries. The switch attached to the robot was connected to the Arduino as an input; when the switch is on, the circuit is closed, and current is delivered to the board (Figure A5).

3.1.3 Approaching Nearby Objects

We decided to plot a few of our desired motor speeds with respect to the appropriate detected distance online and generate a formula

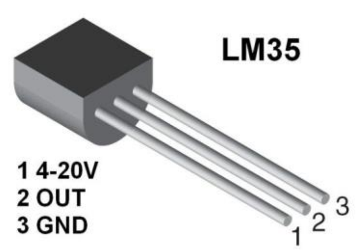


Figure 3.3 LM35 Temperature Sensor

(Figure A6) to gradually reduce the speed of the robot as it approaches obstacles. Using that formula, we created a function that would calculate the required speed when our robot was close enough to the object. To improve the accuracy of distance readings, we added a temperature sensor to obtain the environments speed of sound.

3.1.4 Edge Sensing

To detect edges of a raised surface, we attached a second Ultrasonic Range sensor to the robot. This sensor was extended further in front of the robot to sense the distance of the surface below the robot. To place the sensor, we initially used cardboard, but realized that it was a poor design choice and switched to metal (see section 3.2.3). To further extend the metal outwards, we realized that we could use the rounded metal frames and attach them onto the extended plate at the top (Figure A8).

We then measured the height of the range sensor to be approximately 13 cm and checked if the robot is about to leave level ground. When it does, the robot would reverse to make room for turning, then turned to a random side, based on angles between 20 to 100 degrees, to proceed.

3.1.5. Cleaning Maximum Area

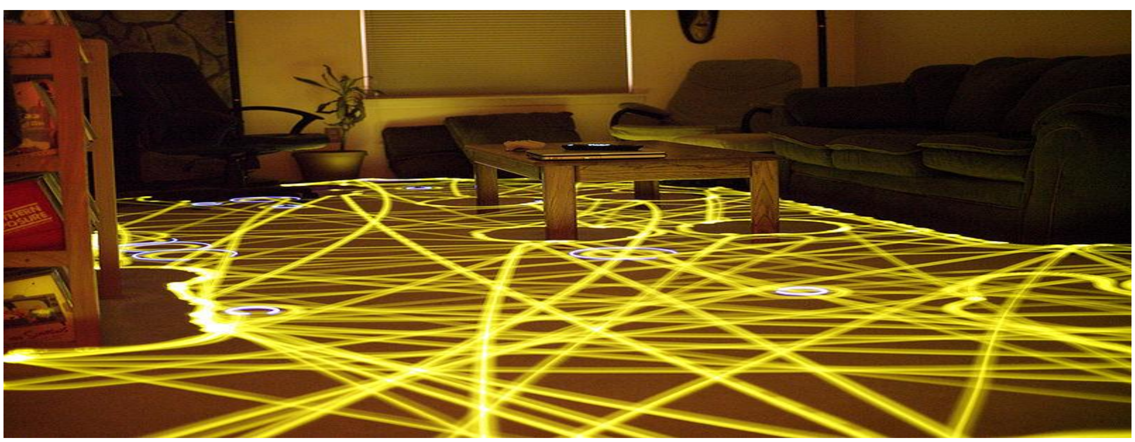


Figure 3.4 Results of our algorithm, inspired by the path taken by an iRobot Roomba vacuum. http://commons.wikimedia.org/wiki/File:Roomba_time-lapse.jpg

Originally, we implemented a row-by-row algorithm, but found it to be extremely inefficient, as it would take a long time to cover the area of the room (Figure A10/A11). To ensure we cover the most area of the surface we are cleaning, we implemented a slightly randomized algorithm (Figure 3.4), which covers areas of different shapes and sizes. Now, we start by assuming the area we are cleaning has at least one wall to detect, then the robot can be placed anywhere.

As shown in Figure A12, the robot will find the nearest wall of the area by first moving towards the first wall it sees, then turning right and moving in a randomized path, which keeps the robot on the right side of the area. The robot increments a count each time it is faced with an obstacle or edge, and continues on the right side till it has counted 9 times. Now that it has covered a sufficient area on one side of the area, so the robot moves to the opposite side by mapping its location to the original location it started in. The robot then performs in a similarly on the left side, cleaning random unobstructed paths. And continues to follow random paths afterwards.

3.1.6. Switching Between Modes

Due to few effective push buttons in our kit, we wired a photoce (see figure 3.5) to create a reliable switch that will allow us to switch between modes. We created a small box to cover the photocell, so that when the user would like the robot to perform its basic functionality, they could keep the box over the photocell. When the photocell detects a higher ambient light value, the box is off, and the robot will enter its cleaning mode.

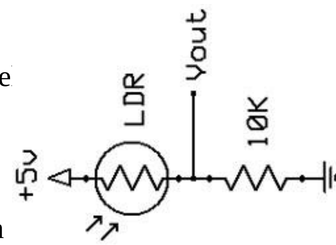


Figure 3.5 Photocell (LDR) schematic.

3.1.7. Displaying Data on the LCD

As the LCD is known for its heavy power consumption, we avoided using it for anything other than the basic functionality. The LCD will display on the first row the distance to an object in centimeters, and on the second row whether it is approaching an object, as well as the direction it is proceeding in (forward or left). Due to issues with external power, we had trouble with displaying our intended messages on the LCD (see 3.2.5).

3.2. Evaluation/Testing Procedures and Results

3.2.1. Calibrating Motion

When our robot moved along the ground, we noticed that it did not move in a straight line when powered by an external power source. Although the speed written to both motors were the same, this was likely a result of a minor difference in the current sent to each motor. Since we needed to ensure that both wheels moved at the same speed, we multiplied varying numerical factors, including 1.015 and 1.01, to the slower right wheel, to determine what value would enable the robot to move more linearly. Depending on current delivered to the motors, and the friction of a surface, the value would vary. When the motors did not move straight, we would test small multiples until we found a value that would allow the robot to move straight.

Although this is not an ideal method, we discovered an alternative far too late into the project (see 3.3.4) and could not implement it. However, the alternative would increase expenses and power consumption, while self-calibration could still get the robot to move fairly straight, just not as accurate as an encoder could. In the future, we should do more research into small details early into a project.

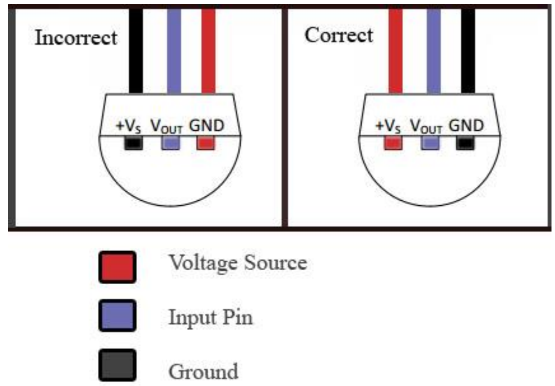


Figure 3.6 Incorrect temperature sensor wiring vs. corrected wiring.

3.2.2. Debugging Temperature Sensor

Initially, after wiring the sensor to the Arduino, we found that we received extremely high temperature readings of nearly 1000 degrees Celsius printed to the Serial Monitor. After further observation, we realized we had reversed the circuitry (Figure 3.6). With the temperature sensor correctly wired, we managed to detect lower and more reasonable temperatures, around 22 degrees Celsius, to calculate the distance. Considering the success of the Serial Monitor for debugging, we intend to continue using it to display uncertain values, as needed.

3.2.3. Testing Edge Sensing

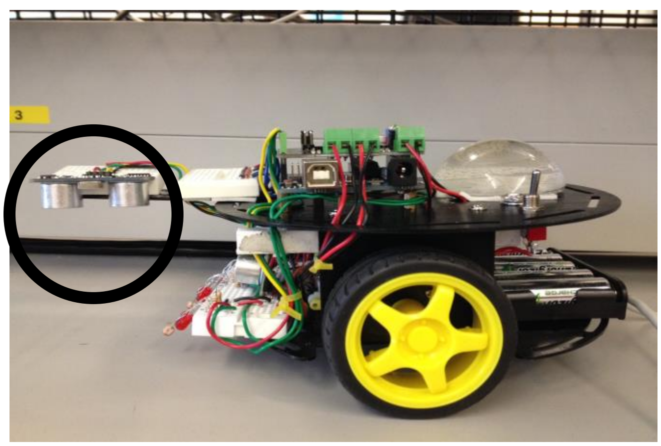


Figure 3.7 Range sensor for edge detection, extended in front of robot.

When we tested our edge sensing functionality, we initially found that the robot nearly fell off an edge when moving left and right to determine where it can move without falling off the raised surface. Due to that, we added a reverse function to provide the robot with additional space before turning.

After rewiring, we found that when the robot is approaching an edge at an angle, the range sensor might not detect the lower ground early enough to prevent itself from falling off (see section 3.3.5). A quick solution to this was to extend the range sensor further out from the robot to detect edges before the robot fell off (Figure 3.7). However, since we would like to keep the robot as compact as possible, we did not extend the range sensor too far out, as this would cause an imbalance in weight and the robot would increase in length.

3.2.4. Relocating Second Range Sensor

After successfully implementing edge sensing to our robot, we found that our basic functionality no longer worked as expected--our robot simply remained in the spot it was placed in and kept turning left in circles. To determine the issue, we added a few lines of code to print to the Serial Monitor (Figure 3.8), and found that the bottom range sensor detected distances correctly up to 13 cm.

```
Serial.print("Distance: ");

Serial.print(distance);

Serial.print(" Temp: ");

Serial.println(temperature);
```

Figure 3.8 Code to print testing values to the Serial Monitor

That was because the placement of our top range sensor was too low, and hence the bottom range sensor detected that instead of objects further away. As a result, our robot always thought that it had an object in front of it and quickly turned.

Originally, we considered mounting the sensor on a rod or a sturdier cardboard structure to resolve the issue. By using a piece of cardboard placed outside of the detection area of the bottom sensor, we found that our range sensors were detecting correctly--the top sensor would detect the lower surface and the bottom could detect objects much further ahead of it. However, cardboard was not an ideal nor professional solution. A peer from the other lab section saw this and suggested that we move the robot's metal structures to use pre-manufactured parts as a support for our top range sensor. Following this suggestion, we were pleased to find that we could improve the neatness of our wiring and overall appearance (Figure A8). Since not all members of the group were familiar with the assembly process of the robot, we did not know some parts could be moved around. From this, we learned a valuable lesson: we should always research provided components carefully and not expect to exactly follow images displayed in manuals, and all members should be equally familiar with each part of the project.

3.2.5. Correcting Incorrect Displays on the LCD

When we connected the LCD, we found that once the robot was connected to external power, it did not always display the expected output. Sometimes, it would show odd characters, that may have been a mix of Japanese and Greek letters. That was likely because the noise of the motors was creating an inconsistent voltage, leading to these displayed results. We could have prevented this with a bypass filter (see 3.3.3.), but due to time limitations, we settled with covering the wires with a piece of aluminum foil and grounding it to prevent magnetic fields caused by the induced currents from creating this error.

3.3. Design Considerations and Possible Alternatives

3.3.1. Power Considerations

To avoid consuming too much power and using up our batteries quickly, we took power readings from major components (Figure A9) and avoided using the LCD in our second functionality. In addition, we purchased rechargeable batteries since they provide 1.2 volts consistently; while disposable batteries start at 1.5 volts and gradually decrease until it they are dead.

3.3.2. Slowing Robot Before a Turn

A basic functionality of the project is for the robot to down gracefully as it approaches an object. Initially, we were able to slow down the robot near objects by using a loop and decementing the speed linearly. However, the motion was not very fluid, due to a constant decrease rate, regardless of the distance between the robot and the object. We later tried a linear function, but found the decrease in speed barely noticeable. Hence, we decided to create an exponential formula to decrease the rate depending on the robot's distance as it nears an object (see section 3.2.1).

3.3.3. Adding a Bypass Filter

One possible improvement we could add to our project is to solder 100 nF capacitors in parallel with both motors to stabilize the voltage slightly. As we were using an external power source, some ripple voltages would be expected, and may potentially cause the robot to not move straight. [3] Adding a capacitor would resolve this issue and slightly improve the accuracy of our motor speeds, as one motor sometimes moved slower than the other due to a non-ideal voltage source. The external voltage source would sometimes cause current to fluctuate slightly, resulting in slight curves in motion.

Unfortunately, by the time we learned of this from a peer, the deadline of the project was approaching soon, leaving us with nearly no time to implement this addition. In both cases, we learned that research should be done ahead of time, before starting to work on the project. By working at a slower pace, we would have saved time and improved performance in the long run.

3.3.4. Dynamically Correcting Inaccurate Speeds

To take calibration of the motors to the next step, we could add an IR emitter and IR sensor near each other for both motors so that we could determine the rotation of wheels and observe if one wheel is faster than the other, and correct this discrepancy accordingly. Once again, by the time we learned this from a peer, the deadline of the project was approaching soon, leaving us with little time to implement this addition.

3.3.5. Avoiding Obstacles and Edges at an Angle

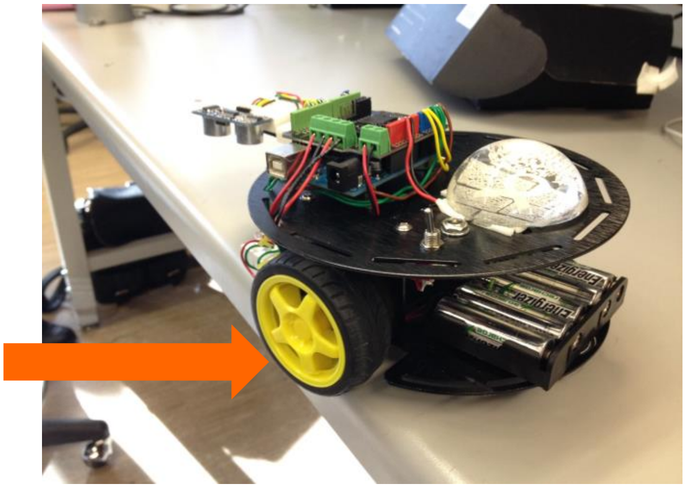


Figure 3.9 Robot wheel falling off before edge could be detected.

In most cases, our robot would detect objects and edges in front of it, and turn accordingly. However, at particularly steep angles, we found that our robot did not detect obstacles or edges quickly enough, and ended up colliding or falling (Figure 3.9). This was because the range sensor detected objects only within 30 degrees left and right and thought that it was much further from the object or edge than it was in reality. Hence, the robot would continue to move towards the object, believing that it still had room to move. We programmed our robot to turn much did to account for more of these angles, providing comfort that the robot will not collide with objects most of the time. In addition, we need to protect the edge sensor since it is a few inches in front of the robot. However, if we attached a servo, we may have been able to more efficiently detect and avoid objects at an angle.

3.3.6. Soldering vs. Breadboards

Although breadboards are meant to be used for prototypes, and are not ideal for a finished product, poor planning and a lack of research gave us a limited amount of time to consider soldering to boards. Soldering would certainly result in wires that are better connected and sturdier than a prototype. However, we realized this during the weekend before the deadline, so we could not locate the technician to assist us with cutting the circuit board into smaller pieces. In the future, we should always consider soldering instead of prototyping for a final product.

4.0. CONCLUSIONS

Transitioning from smaller three-person teams to a six-person team was not easy. Although many members of our team were motivated and everyone wanted to create a project we would all be proud of, we found that many of our goals and ideas were not necessarily realistic and each person's vision of the final product varied. Our team had plenty of ideas and people willing to suggest, but no team leader, leaving everyone working with no direction. Early on in the project, we focused on finishing the basic functionality, but did not truly read in between the lines of projects and did little research. At the same time, we continually discussed ideas and could not settle on any particular plan for our second functionality well into the second week. We went from an IR remote controlled robot, to a firefighting robot, to agreeing on a cleaning robot, only to find that our algorithm also didn't work. In the end, we had to settle on two simpler functionalities we had already implemented for a cleaning robot: following an efficient path and avoiding edges when our robot is close to falling off a raised surface. Near the end, we ended up restructuring the entire robot, reassembling components and rewiring each portion to focus on further polishing our basic functionality.

Although this project had its fair share of troubles, we learned a valuable lesson from it: we should start with each piece and test instead of writing too much code ahead of time. Also, we should have a solid plan early into the project to save time on changing ideas and discarding completed work. We did manage to successfully complete the project, but with better planning and time management, we believe we could achieve much better results.

LIST OF REFERENCES

[1] "Basic Kit for Turtle 2WD". Available: https://www.dfrobot.com/wiki/index.php?title=Basic_Kit_for_Turtle_2WD_SKU:ROB0118, [Mar 8, 2015]

[2] Mobile Platform - 2WD Turtle: Instruction Manual Booklet

[3] K. Ross. "Basic Circuits - Bypass Capacitors". Available: http://www.seattlerobotics.org/encoder/jun97/basics.html [Mar. 8, 2015]

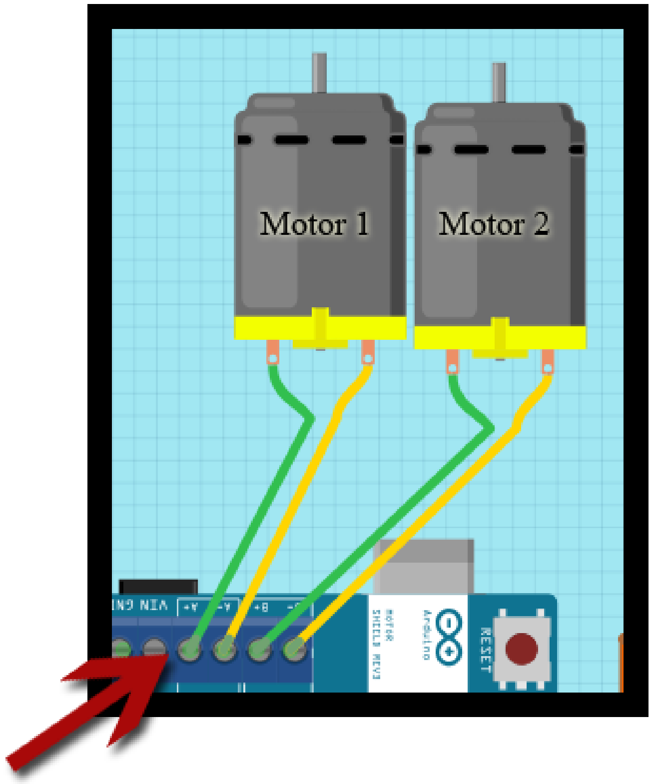
[4] Amarino. "Test Your Sensors for Line Following Robots". Available: http://www.buildcircuit.com/test-your-sensors-for-line-following-robots/ [Mar. 8, 2015]

[5] "Video of our Line Following Functionality". Available: https://www.youtube.com/watch?v=5LM3fTRRWE4&feature=youtu.be [Mar. 10, 2015]

[6] K. Shirriff. "Arduino - IR Remote Library". Available: https://github.com/shirriff/Arduino-IRremote [Mar. 8, 2015]

APPENDIX A: ADDITIONAL INFORMATION

A.1.Tables and Figures



Speed Pin 1: 5
Speed Pin 2: 6
Direction Pin 1: 4
Direction Pin 2: 7

Figure A1. Unscrew Arduino pins and wire according to diagram to implement motors

```
void processMotion(){

        if (distance >= NORMAL_DISTANCE){  // If object is 30+ cm away

                carForward(FAST_PACE, FAST_PACE); // Move forward
quickly

        }

    else if (distance > VERY_CLOSE_DISTANCE){  // Gradually decreasing
speed between 20-30 cm.

                int pwmSpeed = calculatePWMSpeed(distance);

                carForward(pwmSpeed, pwmSpeed); // calcPWMSpeed is
calculated based on a formula

}

        else { // Car is 20 or less cm away, stop briefly, then turn left

                carStop();  // Stop

                carTurnLeft(TURN_PACE, TURN_PACE);  // Turn left

        }

}
```

Figure A2. If else loop determining car motion. Function calls in blue are calls to functions provided by the Motor Shield Wiki [1].

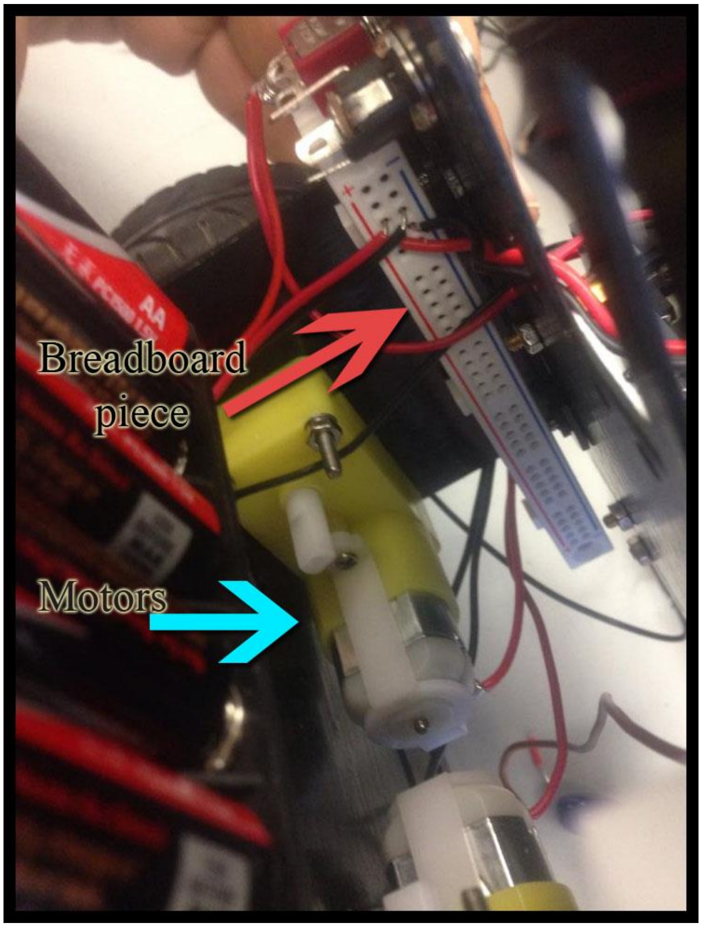


Figure A3: Breadboard pieces near motor

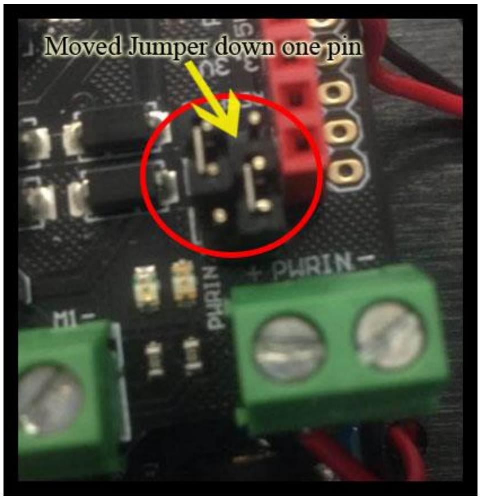


Figure A4. Moving Jumper Pins to enable external power

Figure A5. External Power connected through switch

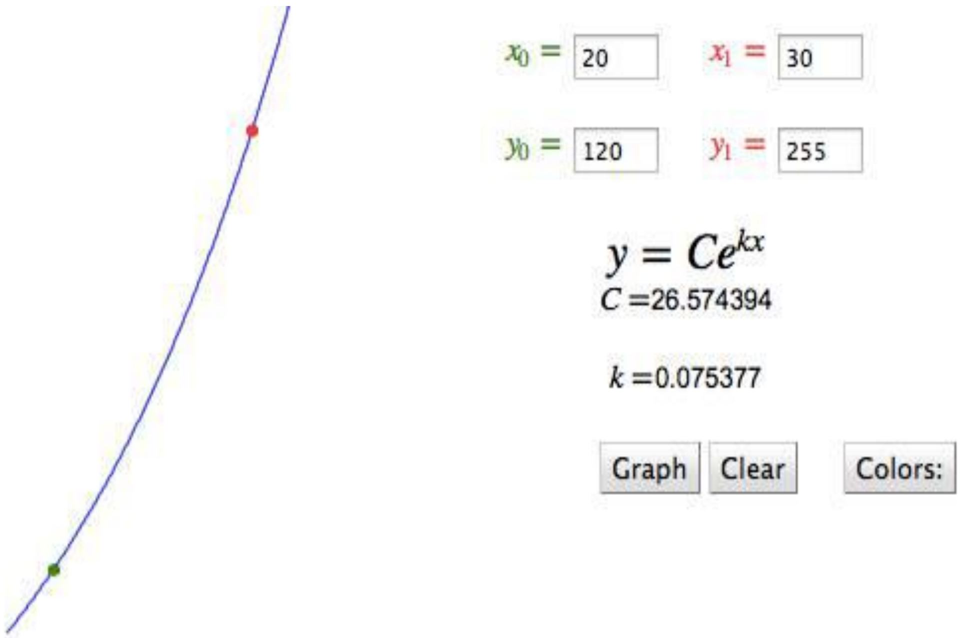


Figure A6: Exponential Function used for slowing robot before a turn

.

```
void edgeDetect(){

	int distanceBelow = getDistance(ECHO2, TRIG2); //Obtain the distance from the range sensor parallel to the edge sensor.


	if (distanceBelow > 18){ //If the distance from the ground is significantly lower than level ground.

		carStop(); //Stop the car.

		delay(250);

		carReverse(FAST_PACE, FAST_PACE); //Turn back to make room for turning.

		delay(500); // Reverse to avoid falling off edge

		carTurnLeft(TURN_PACE,TURN_PACE); //Then turn to remain on table.

	}

}
```

Figure A7. Edge sensing code with a call to carReverse().

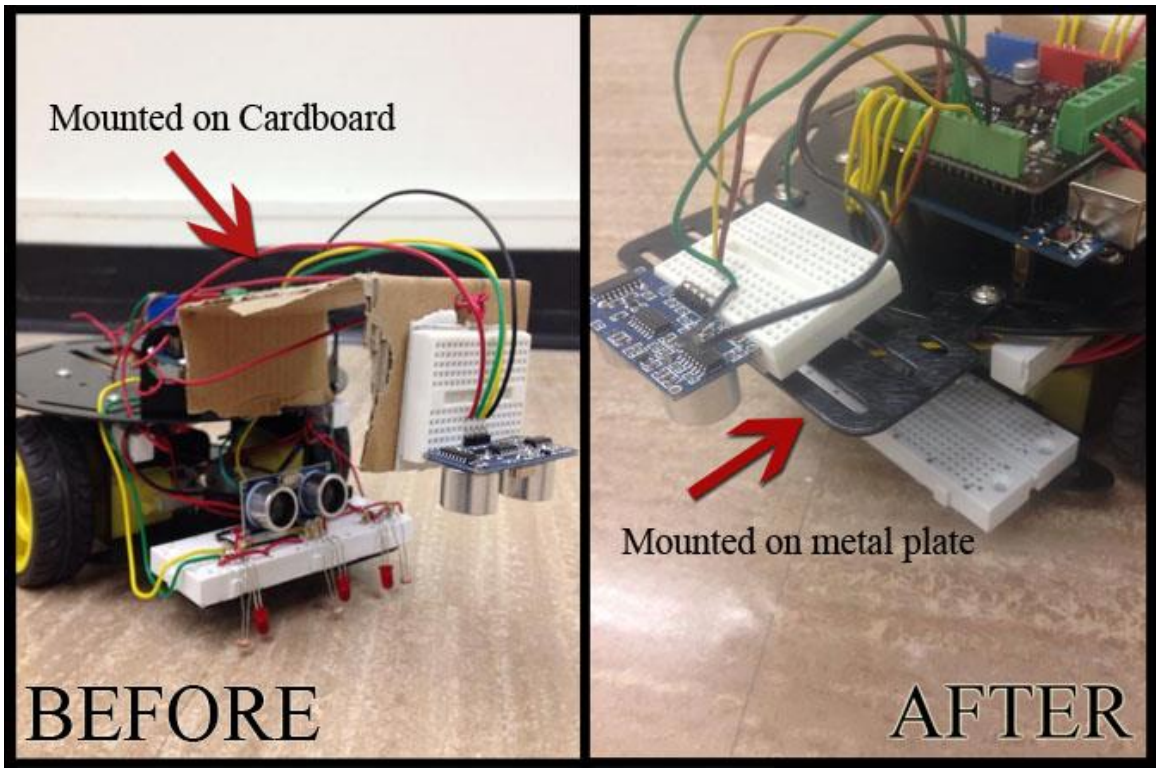
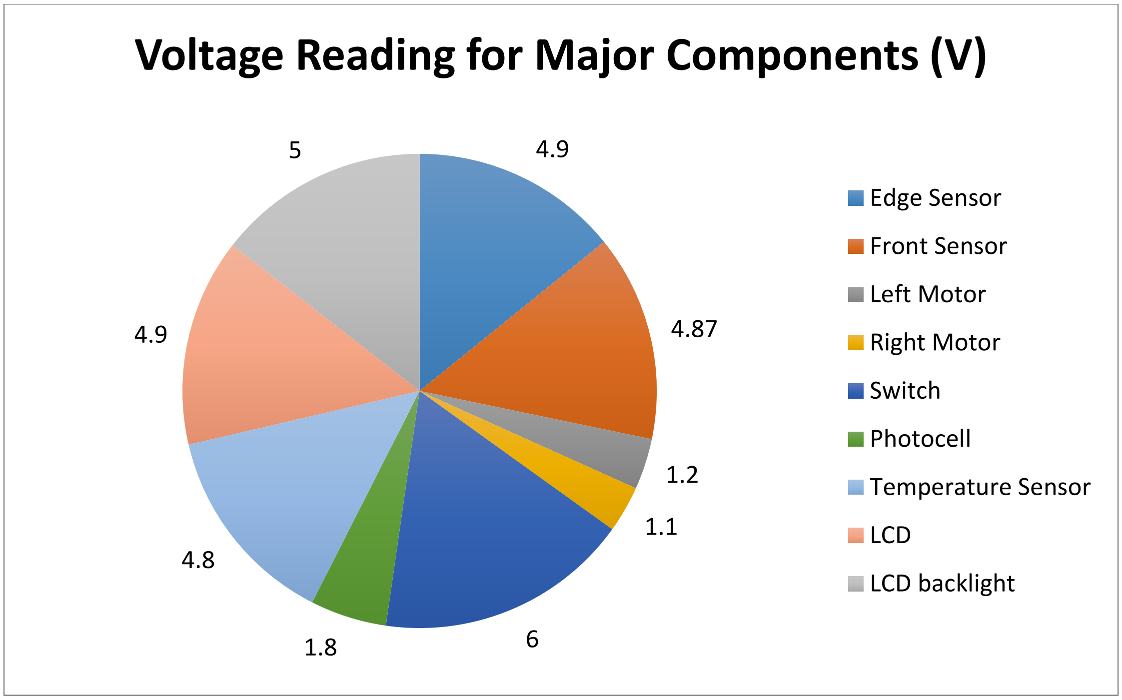
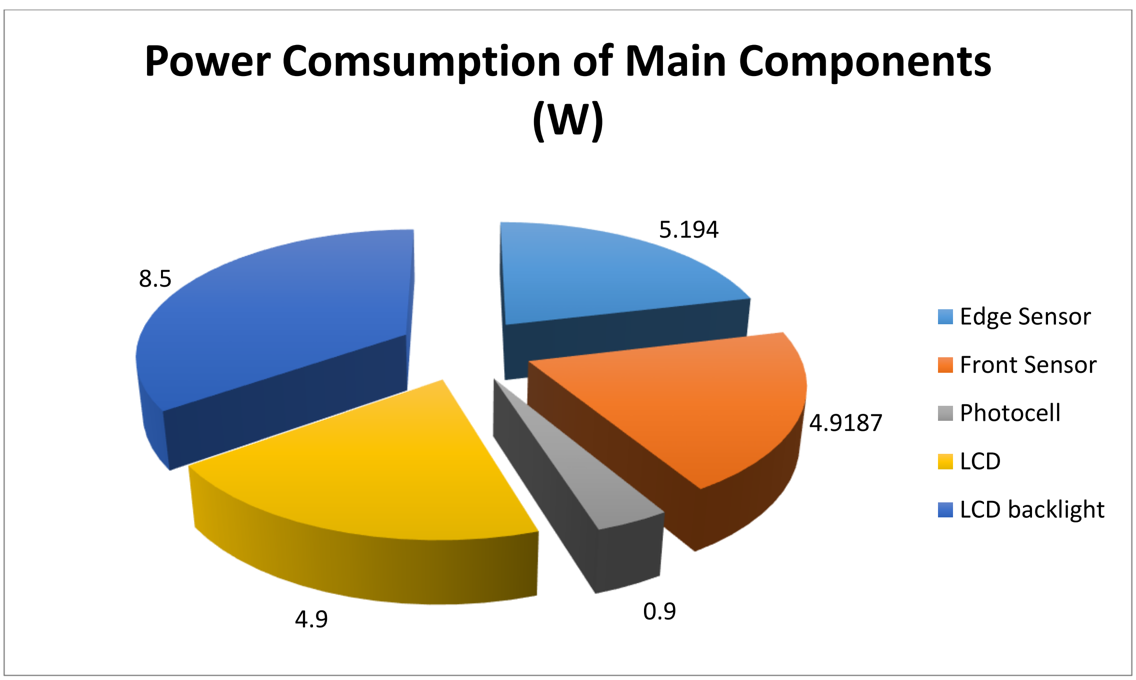


Figure A8. Comparison between robot before and after moving structure.



a)



b)

Figure A9. a) Voltage readings and b) power consumption of major components.

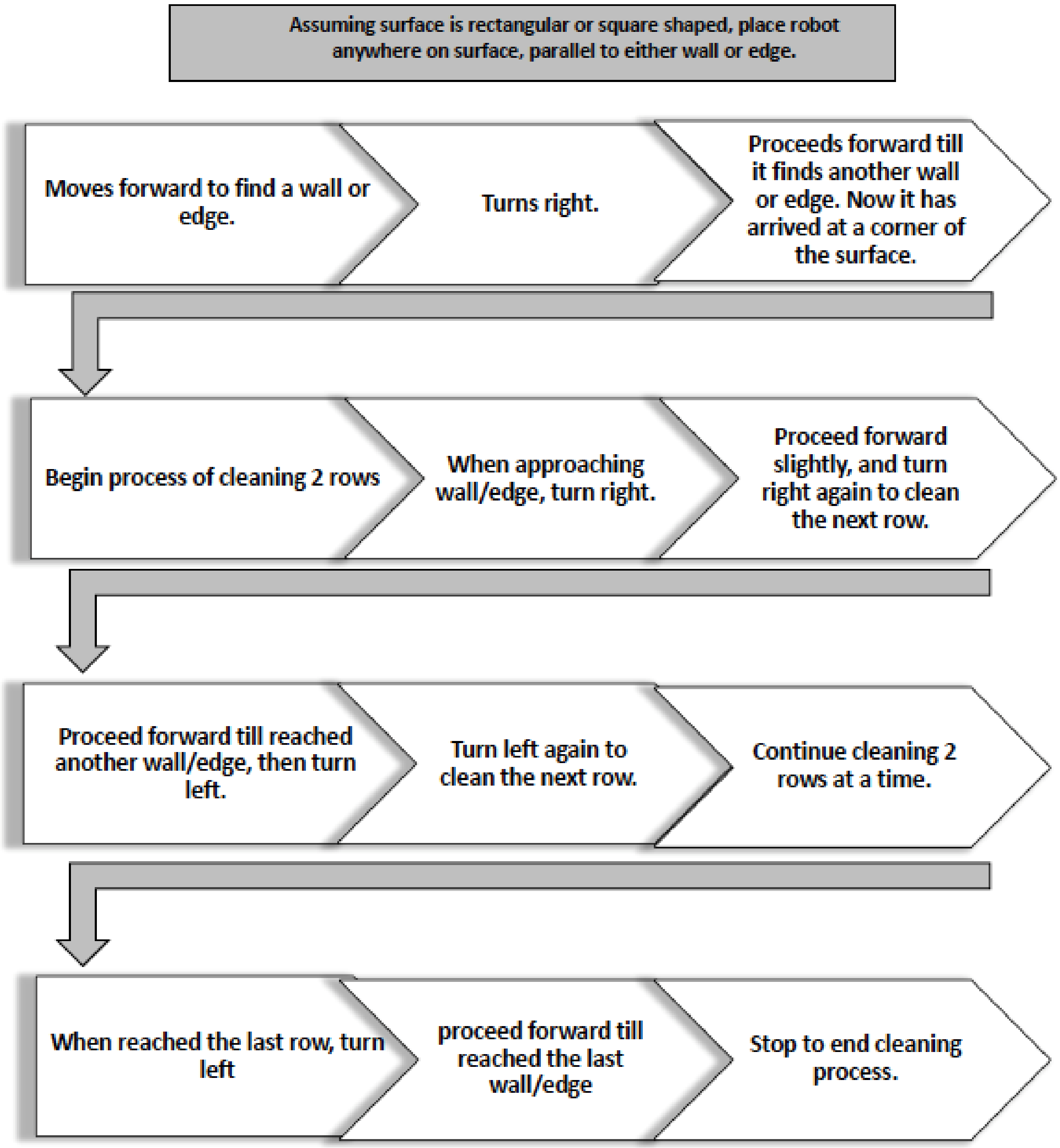


Figure A10. Previous inefficient algorithm for cleaning maximum area of a surface.

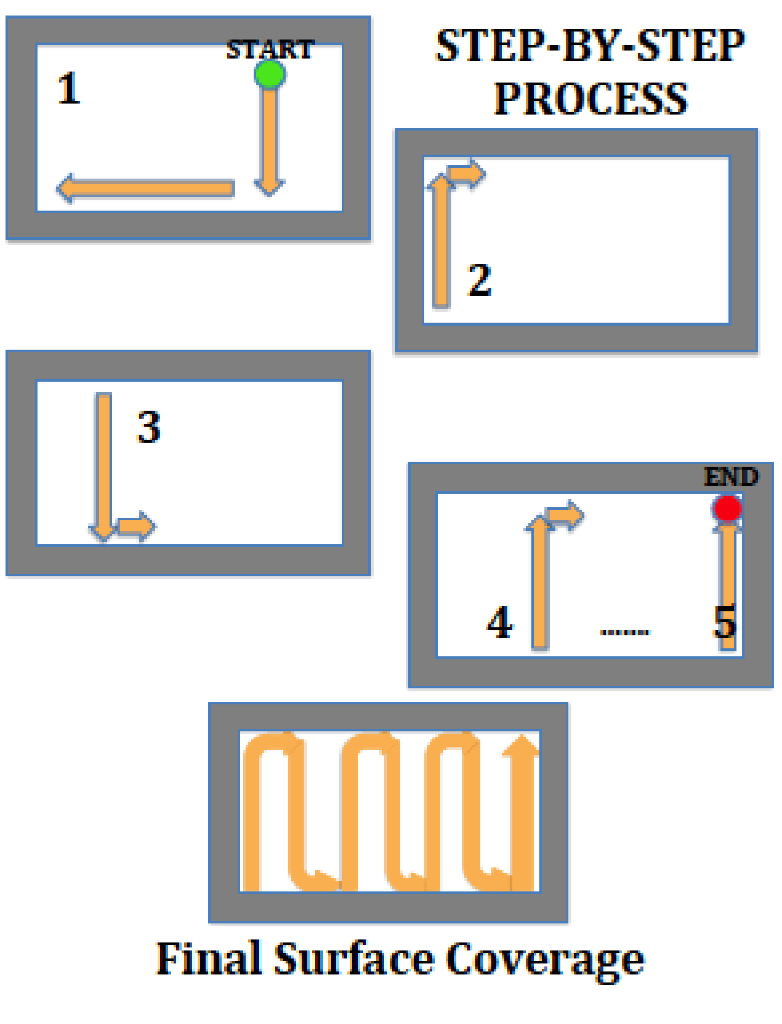


Figure A11. Previous process of cleaning a surface.

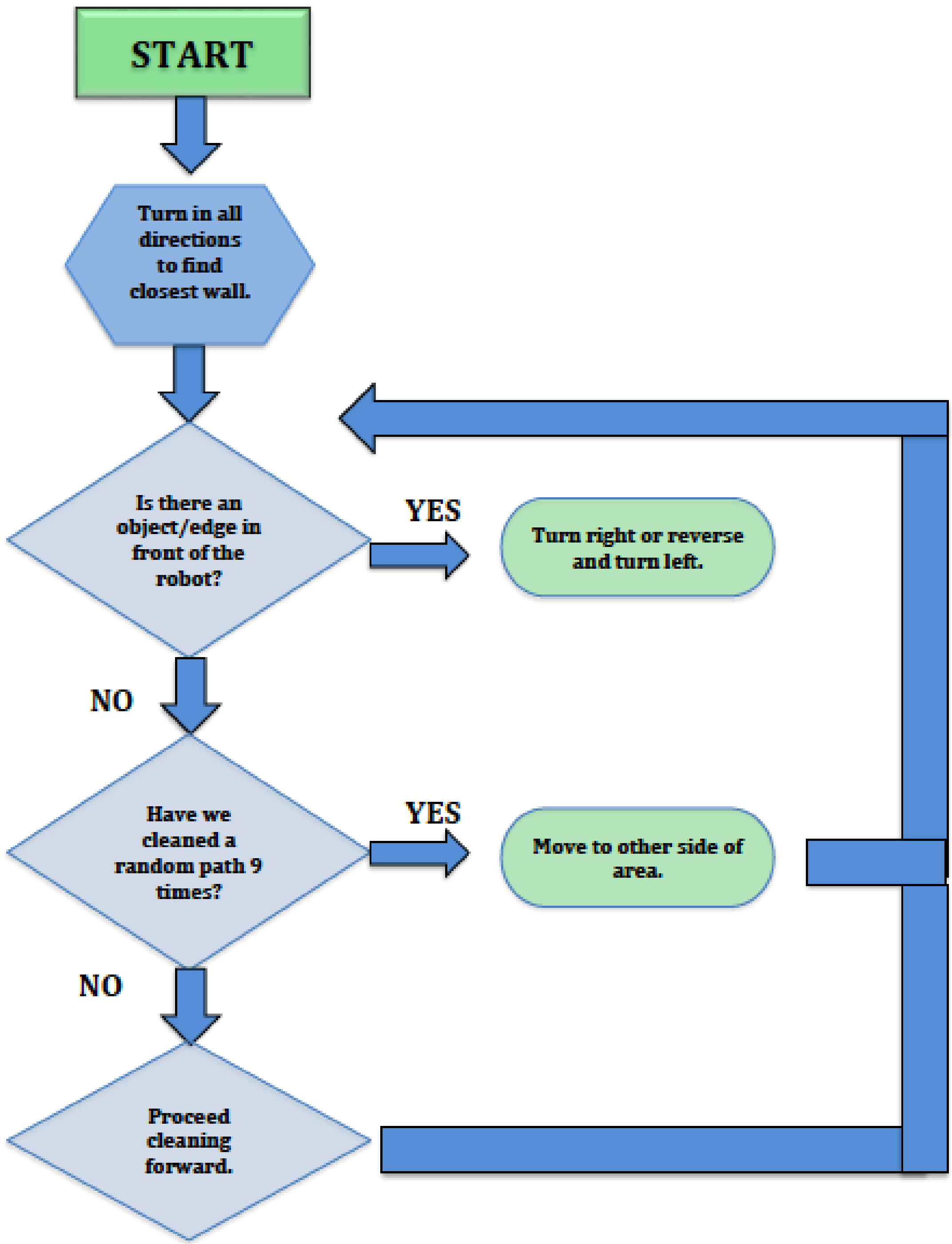


Figure A12. Randomized cleaning algorithm.

A2. List of Components

Component ID	Component	Purpose
HC-SR04 x2	Range/Edge sensors	Detects objects in front and distance to the floor.
LM35DZ	Temperature sensor	Calibration of distance.
TC1602A-01T	LCD	Display motion information.
3386F-103TLF	Potentiometer	Adjusting the contrast of LCD.
PO502 x2	Left and right motors	Move robot in directions with varying speed.
PDV-P5001	Photocell (LDR)	Sensing bright light for switching modes.

A3. Attempted Second Functionalities

A.3.1. Line Following

Inspired by the implementation of a line-following robot [4], we wired three photocells each with LEDs beside them (see figure A13). To allow a robot to follow a dark line along the ground, we took advantage of a darker color's tendency to absorb light and brighter color's tendency to reflect light (see figure A14). A white patch sensed using this method would generally provide a reading of 700 to 850 lux, while a black patch would result in a reading of about 650 lux. These values indicate that there is a detectable difference between darker and lighter colors. Since the LEDs must be on for the entire time we aim to sense an object, we simply connected them to the voltage source, in parallel.

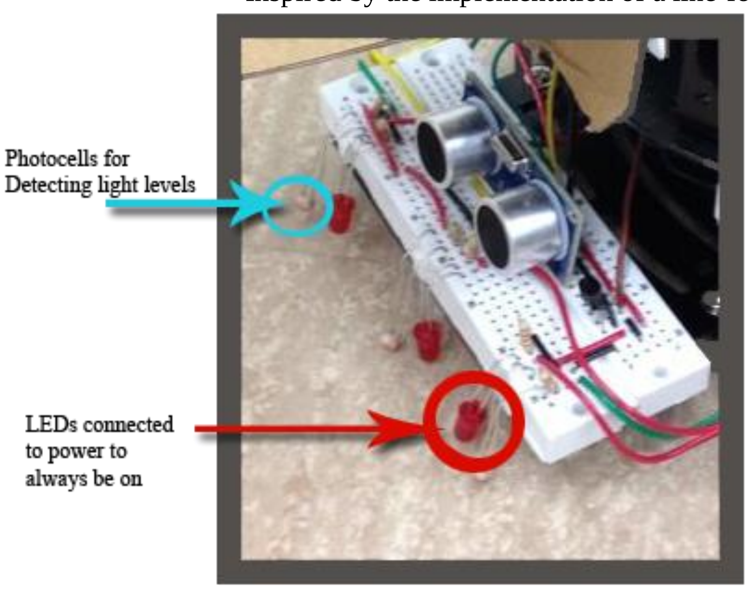


Figure A13: Photocell and LEDs for color sensing based on property that darker colors (ie. black) absorb light and light colors (ie. white) reflect light.

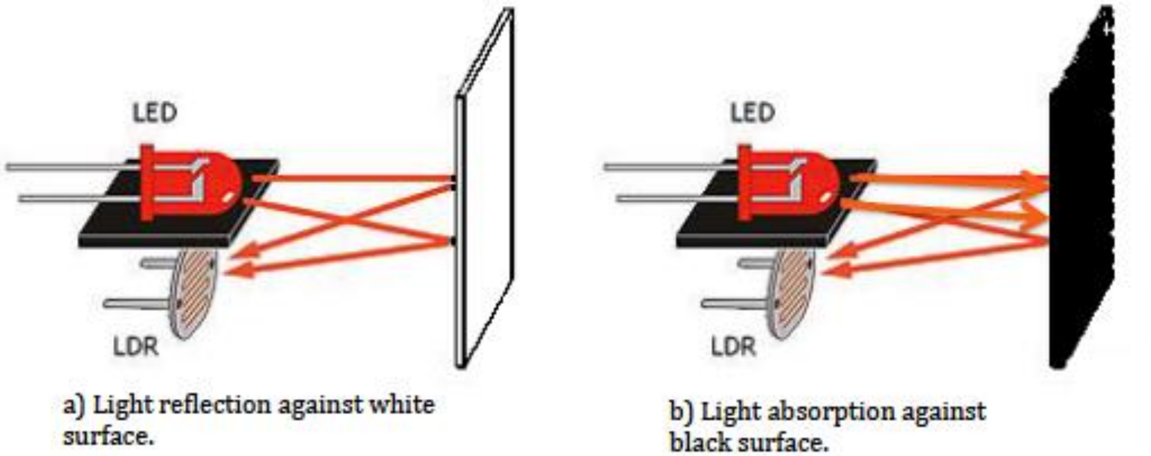


Figure A14. a) Light reflection   b) Light absorbance

Although this method consumes more power, LEDs generally have lower power consumption than most of our other components (see figure A9), and this reduces the number of Arduino pins we use by three.

Using this data, we checked the values detected by the photocells every 1 millisecond and decided if the robot needed to move left or right, or remain in the center (see figure A15 and A16). Due to the fact that photocells are not the most accurate method of detecting contrast between colours, we used solid black lines, approximately 1-2 inches wide for the most effective results, against a white sheet of poster paper to check if this functionality worked as expected.

Unfortunately, after successfully following black lines for a few days, this functionality no longer worked as expected. We were able to take a video of this functionality before it malfunctioned, which is linked in [5].

Line Following Algorithm

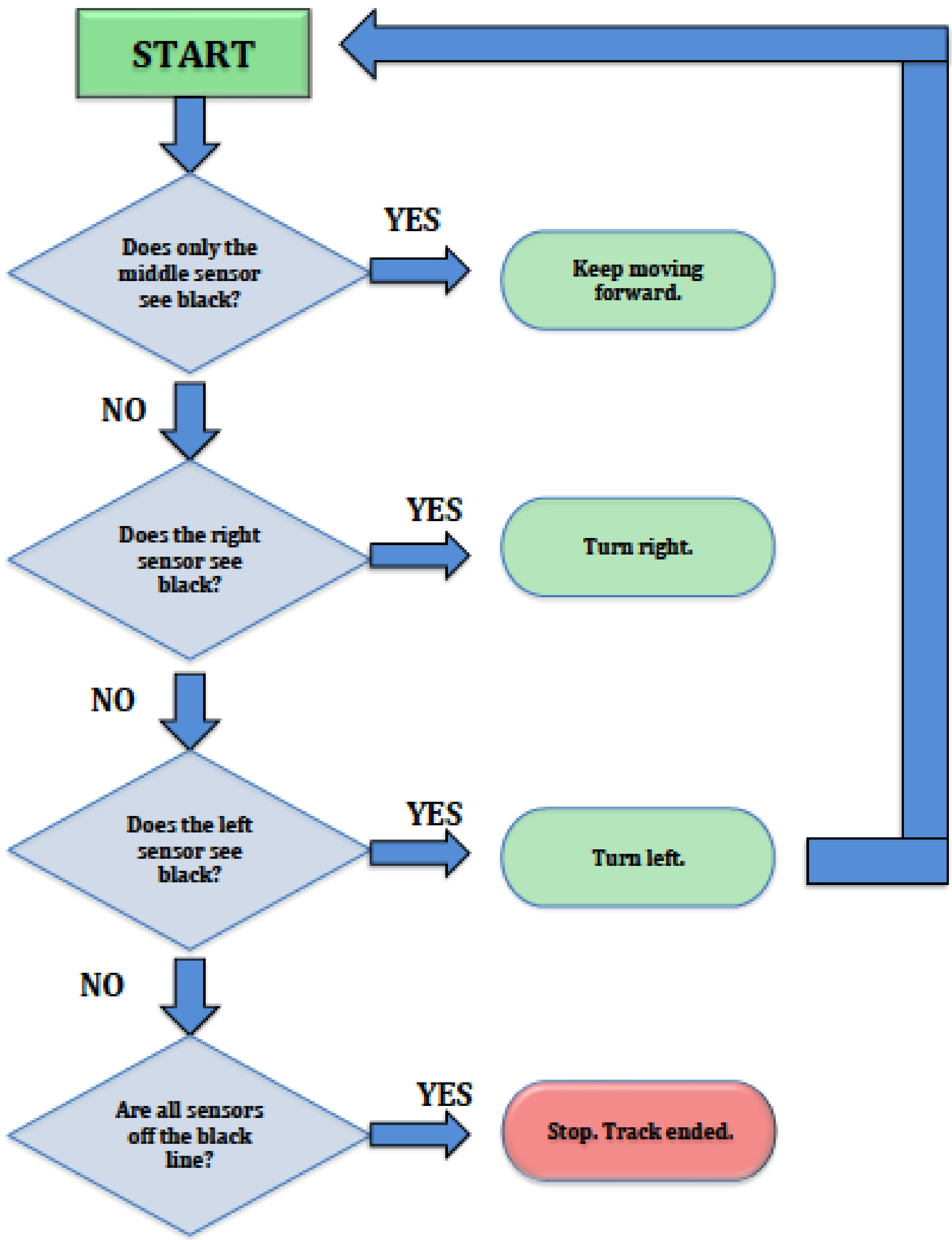


Figure A15. Line following algorithm.

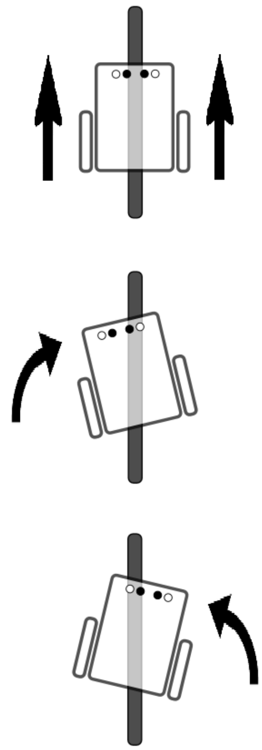


Figure A16. Robot decides to proceed forward, turn right, or turn left.

A.3.2. Use IR to Switch Between Modes

To switch from the basic functionality and our second functionality, and as a second functionality, we considered using an infrared sensor along with a Universal Remote Control. Although the idea seemed simple, especially with the use of the Arduino IR library [6], we found that various buttons on the remote gave out the same encoding, based on distance from the receiver. Although we could simply program the Arduino to switch between modes as soon as any infrared signals were detected, it would be wasteful to use a remote control that has far more keys. Since we would not be using the full potential of IR sensors and emitters with this functionality, we did not go through with this option.

APPENDIX B: COMPLETE ARDUINO CODE

```
/*
* Assignment: Project 1
* Authors: Catherine Lee, Sangeetha Kamath, Syed Iqbal,
*          Rosa Mohammadi, Brian Chang, Srinjoy Chakraborty
*
* Purpose: The robot performs 2 main functionalities, a basic and
*          additional functionality.
*
*          In the basic functionality,
*          the robot proceeds forward at full speeds till it
*          approaches an object, when it slows down and
*          turns left at a right angle.
*
*          In the additional functionality, the robot performs
*          a maximum area coverage algorithm for cleaning a room,
*          whether it contains elevated surfaces or not. It covers
*          maximum area by first going to the right side of the area,
*          then cleaning paths in angles of 20-100 degrees. Then it
*          recalibrates its location, to move to the left side of
*          the room, where it performs similarly.
*/


/*The LCD circuit:
 * LCD RS pin to digital pin 2
 * LCD Enable pin to digital pin 3
 * LCD D4 pin to digital pin 15
 * LCD D5 pin to digital pin 14
 * LCD D6 pin to digital pin 8
 * LCD D7 pin to digital pin 9
 * LCD R/W pin to ground
 */
```

```
// include the library code:
#include <LiquidCrystal.h>

//initialize the library with the numbers of the interface pins
LiquidCrystal lcd(2, 3, 15, 14, 8, 9);

/*
-----------------------------------------------------------------
Connected Pins
-----------------------------------------------------------------
*/
//Motor Control Pins
int M1_SPEED_PIN = 5;     //M1 Speed Control - left wheel
int M2_SPEED_PIN = 6;     //M2 Speed Control - right wheel
int M1_DIRECTION_PIN = 4;     //M1 Direction Control - left wheel
int M2_DIRECTION_PIN = 7;     //M2 Direction Control - right wheel
//Pins for Range Sensor 1 (Frontal Object Detection)
#define TRIG1 10 //Trig of range sensor to digital I/O pin 10
#define ECHO1 11 //Echo of range sensor to digital I/O pin 11
//Pins for Range Sensor 2 (Ground Level Detection)
#define TRIG2 12 //Trig of range sensor to digital I/O pin 12
#define ECHO2 13 //Echo of range sensor to digital I/O pin 13
#define TEMP_SENSE A5 //Temperature Sensor calculating distance
#define PHOTOCELL_BUTTON A2 // Photocell for switching modes
/*
-----------------------------------------------------------------
Distances and Speeds
-----------------------------------------------------------------
*/
#define FAST_PACE 255  // Full PWM speed
#define TURN_PACE 255  // PWM Turning speed
#define EDGE_DETECT_PACE 90 //A slower pace for detecting edges

//Notable Distances in cm
#define NORMAL_DISTANCE 35 //The threshold distance, in cm, between a detected object and the
range sensor that is considered safe to approach without slowing down.
```

```
#define VERY_CLOSE_DISTANCE 30 //The threshold distace, in cm, between a detected object and the range sensor before the robot must stop and perform a turn.


#define WHEEL_SPEED_OFFSET 1.07  // Constant to multiply to the right motor, since less power is delivered to that motor.

#define TURNING_TIME 117  // Length of time the car should turn to produce right angles

#define CLEAN_TURN 170  // Additional turning time since the cleaning function operates at a slower speed


#define OBJECT_IN_FRONT 20 // The distance from the object when the robot should turns. 20 cm since edge sensor enlarges robot front length.

#define SURFACE_BELOW 13 // The distance to surface below, when the robot is not going to fall off edge


#define FLASHLIGHT_LUX 700  // When the photocell value is below or above this, change the mode (basic or additional).


#define RANDOM_PATHS 9 // The number of paths we clean on each side of room


int counterForCleaning;  // Counter for keeping track of how many paths robot has cleaned

int mode = 0;  // Basic functionality mode


/*

-----------------------------------------------------------------

Basic Arduino Functions - Setup functions and superloop

-----------------------------------------------------------------

*/

/*

* Name: setup

*/

void setup(){

       Serial.begin(9600); // Initialize the Serial Monitor

        lcd.begin(16,2);  // Set up the LCD

        lcd.clear();  // Make sure the LCD is clear


       initializeSensors(); //Initialize the range sensors and temperature sensor

        pinMode(PHOTOCELL_BUTTON, INPUT); // Initialize the photocell for switching modes


}
```

```
/*
* Name: initializeSensors
* Purpose: Declares the range sensor trig and echo pins, as output and input.
*          Also declares the temperature sensor as an input.
*/
void initializeSensors(){
       //Initialize Range Sensor 1 for front detection
       pinMode(TRIG1, OUTPUT);  // Activate write for trig of range sensor
       pinMode(ECHO1, INPUT);  // Activate read for echo of range sensor

       //Initialize Range Sensor 2 for edge detection
       pinMode(TRIG2, OUTPUT); // Activate write for trig of range sensor
       pinMode(ECHO2, INPUT);  // Activate read for echo of range sensor

        //Initialize Temperature Sensor for added accuracy
       pinMode(TEMP_SENSE, INPUT);  // Initialize the temperature sensor as an input
}

/*
* Name: loop
* Purpose: The superloop that continually runs all the described functionality.
*           It continuously checks which mode should be performed, basic or
*           additional functionality.
*/
void loop(){

   lcd.clear();  // Clear the lcd
   mode = checkSwitchModes(mode);  // Check the photocell for which mode should be entered

   if(mode == 0){  // If the photocell is dark, do basic functionality
     Serial.print("Basic functionality");  // Print to serial monitor that we are entering basic
functionality
     processMotion();  // Turn left upon reaching obstacles
   }

   else{ // Otherwise, photocell is bright, so start cleaning
```

```
     lcd.noDisplay(); // Do not use the LCD for additional functionality

     Serial.print("Clean functionality"); // Print to serial monitor that we are entering
cleaning functionality

     findStartingPosition(0);  // Perform cleaning at position 0.

   }


}




/*

* Name: checkSwitchModes

* Purpose: Checks the state of the environment through a photocell.

*          Depending on whether the environment is dark or bright,

*          return a special value.

*/

int checkSwitchModes(int currentMode){

    int photocellVal = analogRead(PHOTOCELL_BUTTON);  // Read from photocell


    if(photocellVal > FLASHLIGHT_LUX){  // If bright, return 1

      return 1;

    }

    else{  // Otherwise return 0 to indicate darkness

      return 0;

    }


}

/*

-----------------------------------------------------------------

Basic Functionality

-----------------------------------------------------------------

*/

/*

* Name: processMotion

* Purpose: For the basic functionality. Determines

*          the speed and direction the car should move in.

*          If the object is 35+ cm away, the car proceeds

*          at its fastest speed. Otherwise, it gradually
```

```
*          slows down till it is 20 cm away from the object,
*          then it stops, and turns left at a right angle.
*/
void processMotion(){
        int distance = getDistance(ECHO1, TRIG1); // Get the distance in front

          lcd.setCursor(0, 0);  // Begin writing on first row of LCD
          lcd.print("Distance: ");  // Print the distance
          lcd.print(distance);

        if (distance >= NORMAL_DISTANCE){  // If object is 30+ cm away
               carForward(FAST_PACE, FAST_PACE); // Move forward quickly

                lcd.setCursor(0,1);  // Write the state of motion on LCD
                lcd.print("Approaching");  // State is approaching object
        }

      else if (distance > VERY_CLOSE_DISTANCE){  // If object is approaching, reduce speed
               int pwmSpeed = calculatePWMSpeed(distance);  // Gradually decrease speed using an
exponential function.
               carForward(pwmSpeed, pwmSpeed); // calcPWMSpeed is calculated based on a formula

                lcd.setCursor(0,1);  // Write the state of motion on LCD
                lcd.print("Slowing Down");  // State is slowing down
}
        else { // Car is 20 or less cm away
               carStop();  // Stop briefly
                delay(250);
               carTurnLeft(TURN_PACE, TURN_PACE);  // Turn left at a right angle

                lcd.setCursor(0,1);  // Write state of motion on LCD
                lcd.print("Turn Left");  // State is turning left
        }
}



/*
```

```
-----------------------------------------------------------------

Update and Retrieve Data

-----------------------------------------------------------------

*/

/*

* Name: startRangeSensor

* Purpose: Initializes the range sensor to begin

*          detecting pulse widths by starting the trigger.

*/

void startRangeSensor(int trig){

        digitalWrite(trig, LOW);  // Set up the sensor to start detecting range

        digitalWrite(trig, HIGH); //Send a pulse to the sensor.

        delay(.01);  // 10 usec delay

        digitalWrite(trig, LOW); //Turn off power to make this a 10 usec pulse.

        delay(.2);  // Pause for the 8 bursts to end

}

/*

* Name: getDistance

* Purpose: Prepares the range sensor for gathering data

*          and obtains an echo pulse width duration to compute

*          the distance from the range sensor to

*          an object (in cm).

*/

float getDistance(int echo, int trig){

        startRangeSensor(trig);  // Start the trigger and wait for 8 bursts from ECHO1

        // Gather range sensor echo data to assist in calculating distance

        int duration2 = pulseIn(echo, HIGH);  // Returns length of pulse in microseconds or 0 if
no pulse

        float speedOfSound = getSoundSpeed();  // Obtain the current speed of sound


        int distance = (duration2 / speedOfSound) / 2;  // Formula to calculate a more accurate
distance


        Serial.print("Distance: ");  // Print to serial monitor the distance

        Serial.println(distance);

        Serial.println("cm");  // Print the units of distance, cm
```

```
        return distance;  // return the distance from the specified sensor
}


/*
* Name: getSoundSpeed
* Purpose: Using the voltage recieved from the temperature
*          sensor, calculates the temperature(in C) and then the
*          speed of sound(us/cm) using provided formulas.
*/
float getSoundSpeed(){
        float voltage = analogRead(TEMP_SENSE);  // Read the voltage from temperature sensor


        // Formula relating the voltage to temperature(C) for this temperature sensor
        float temperature = (voltage/1024.0)*500;// Convert the voltage to degrees Celsius


        Serial.println("Temperature ");
        Serial.println(temperature);
        Serial.println("degrees Celsius"); // Print the units of temperature to serial monitor


        float speedOfSound = 331.5 + 0.6*temperature;  // Given formula to calculate speed of
sound(m/s) using temperature
        speedOfSound = speedOfSound/ 10000; // Convert speed of sound to cm/us
        speedOfSound = 1 / speedOfSound; // Convert speed of sound to us/cm to use in given
distance formula.


        return speedOfSound; // return the speed of sound of current environment
}
/*
-----------------------------------------------------------------
Calculate PWM Speed
-----------------------------------------------------------------
*/


/*
* Name: calculatePWMSpeed
* Purpose: When the robot is approaching an object,
*          reduce the speed gradually using an
```

```
*          exponential formula.
*/
int calculatePWMSpeed(int distance){
        int pwmSpeed = 26.57*pow(2.72, distance*0.075377);  // Exponential decay to decrease
speed as object gets closer. Decrease as a function of distance.


       return pwmSpeed;  // Return the decayed speed
}


/*
* Name: findStartingPosition
* Purpose: Find the nearest wall when cleaning, and
*          then return to face that wall, then proceed
*          towards it, and turn right to reach a corner.
*/
void findStartingPosition(int directions) {
    int distance, shortDistance, height;
    int counter = 3;  // How many walls to look for, assuming rectangular room

    shortDistance = getDistance(ECHO1, TRIG1);  // Calculate the current distance

    carTurnLeft(TURN_PACE, TURN_PACE);  // Turn left to check if the other wall is closer
    delay(CLEAN_TURN);  // Additional delay to produce right angle turn
    carStop();  // Stop each time
    distance = getDistance(ECHO1, TRIG1); // Check the distance of this wall



    mode = checkSwitchModes(mode);  // Check if we need to switch modes
    if(mode == 0){  // If environment is dark
       return;  // Exit cleaning mode
      }

    if (distance < shortDistance) { // If this wall is nearer
      shortDistance = distance;  // Use this distance to compare to next wall
      counter = 2;  // Set count to 2, so robot returns to this position if it is closest
    }
```

```
carTurnLeft(TURN_PACE, TURN_PACE); // Face the next wall
delay(CLEAN_TURN);  // Additional delay to produce right angle turn
carStop();  // Stop briefly
distance = getDistance(ECHO1, TRIG1);  // Check the distance to this wall

if (distance < shortDistance) {  // If this wall is closer
  shortDistance = distance;  // Use this distance to compare to next wall
  counter = 1;  // Set count to 1, so robot returns to this position if it is closest
 }

carTurnLeft(TURN_PACE, TURN_PACE); // Face the next wall
delay(CLEAN_TURN);  // Additional delay to produce right angle turn
carStop();  // Stop briefly
distance = getDistance(ECHO1, TRIG1);  // Check the distance to this wall

if (distance < shortDistance) {  // If this wall is closer
  shortDistance = distance;  // Use this distance to compare to next wall
  counter = 0;  // Set count to 0, so robot returns to this position if it is closest
}

 mode = checkSwitchModes(mode);  // Check if we need to switch modes
 if(mode == 0){  // If environment is dark
   return;  // Exit cleaning mode
  }

  for(int i = 0; i < counter; i++) { // Turn back to the nearest wall
    carTurnRight(TURN_PACE, TURN_PACE);  // Turn right to get back to the original positons
    delay(CLEAN_TURN);  // Additional delay to get right angle turns
  }

 mode = checkSwitchModes(mode);  // Check if we need to switch modes
 if(mode == 0){  // If environment is dark
   return;  // Exit cleaning mode
  }

distance = getDistance(ECHO1, TRIG1);  // Check distance in front
```

```
while (distance > OBJECT_IN_FRONT) {  // If there are no objects
  carForward(120,120);  // Proceed forward
  delay(100);  // For a short time
  distance = getDistance(ECHO1, TRIG1);  // Check if no objects in front still
}

if (directions == 0) {  // If the first wall was nearest
  carTurnRight(TURN_PACE, TURN_PACE);  // Go to the right side of area we are cleaning
  delay(CLEAN_TURN); // Additional delay for right angle turn
  carStop(); // Stop briefly
  }

else { // Otherwise
  carTurnLeft(TURN_PACE, TURN_PACE); // Turn left
  delay(CLEAN_TURN); // Additional delay for right angle turn
  carStop(); // Stop briefly
}

 mode = checkSwitchModes(mode);  // Check if we need to switch modes
 if(mode == 0){  // If environment is dark
   return;  // Exit cleaning mode
  }

  height = getDistance(ECHO2, TRIG2);  // Make sure no edges are approaching

  mode = checkSwitchModes(mode);  // Check if we need to switch modes
 if(mode == 0){  // If environment is dark
   return;  // Exit cleaning mode
  }

while (height <= SURFACE_BELOW) { // If no edges upcomming
  carForward(120,120); // Proceed forward
  delay(100); // For a short time
  height = getDistance(ECHO2, TRIG2); // Check for no edges again
}
```

```
    carReverse(120,120);  // If there was an edge, reverse
    delay(500);  // Reverse for 500 us

    carTurnRight(TURN_PACE,TURN_PACE); // Then go to the right side
    delay(CLEAN_TURN); // Additional delay for right angle turn
    carStop(); // Stop for 2 seconds
    delay(2000); // To make it obvious we are beginning to clean the right side of area

     mode = checkSwitchModes(mode);  // Check if we need to switch modes
     if(mode == 0){  // If environment is dark
       return;  // Exit cleaning mode
     }

    cleanRoom(); // Begin cleaning room algorithm
}

/*
* Name: cleanRoom
* Purpose: Constantly check if there are obstacle or edges,
*          so the robot can avoid them when cleaning.
*          As long as there are no warnings, robot will
*          proceed forward, and in angles randomly between
*          20 and 100 degrees for maximum area coverage.
*
*/
void cleanRoom() {
    boolean warning = false;  // Start with no warnings originally
    int height;  // The height below robot
    int frontDistance; // The distance in front of robot

    if (counterForCleaning > RANDOM_PATHS ) { // Clean right side of area with 9 random paths
      findStartingPosition(1); // Recalibrate to find the left side of room
    } else if(counterForCleaning >= 2*RANDOM_PATHS){ // Make sure we checked the other side of
area as well
      carStop();  // Stop
      delay(3000);  // Stop for 3 seconds to make obvious it has finished cleaning
```

```
}

int heightOfTopSensor = getDistance(ECHO2, TRIG2);

 mode = checkSwitchModes(mode);  // Check if we need to switch modes
 if(mode == 0){  // If environment is dark
   return;  // Exit cleaning mode
  }

while (!warning) { // As long as no warnings
  carForward(120,120); // Proceed forward
  delay(100); // Proceed forward for a short time

  height = getDistance(ECHO2,TRIG2); // Continuously check for edges
  frontDistance = getDistance(ECHO1, TRIG1); // Continuously check for obstacles in front


 mode = checkSwitchModes(mode);  // Check if we need to switch modes
 if(mode == 0){  // If environment is dark
   return;  // Exit cleaning mode
  }

  else if (height > SURFACE_BELOW) {
    carStop(); // Stop the car
    delay(200); // For a brief time
    carReverse(120,120); // Then reverse to get away from edge
    delay(500); // Reverse for 500 us
    warning = true; // Set warning to true
    newPathTaken(); // Keep track of paths taken and turn left

  } else if (frontDistance < OBJECT_IN_FRONT) { // If objects are too close in front
    carStop(); // Stop
    warning = true; // Set warning to true
    newPathTaken(); // Keep track of paths taken and turn left
  }
```

```
     mode = checkSwitchModes(mode);  // Check if we need to switch modes
     if(mode == 0){  // If environment is dark
       return;  // Exit cleaning mode
      }

    }
    carStop(); // Stop the robot after cleaning everything
    delay(1000); // Stop robot for 1 second
}



/*
* Name: newPathTaken
* Purpose: When reaching an edge during cleaning,
*          keep track that the number of paths has
*          now increased, since we are turning left
*          onto a new path. Then delay for a random
*          time to turn in different angles around the
*          area we are cleaning.
*
*/
void newPathTaken() {
    counterForCleaning++; // Increment number of paths
    int rand = random(10, 270); // Generate a random number for delaying angle
    carTurnLeft(TURN_PACE,TURN_PACE); // Turn left
    delay(rand); // Turn left at a random angle to cover more area
    cleanRoom();
}

/*
-----------------------------------------------------------------
Car Motion Functions
-----------------------------------------------------------------
*/
// Note: Due to mechanical problems, the car moves slightly to
//       the right rather than in a straight line. To correct this,
```

```
//       we adjusted the right motor speed to move faster by a constant
//       than the left motor.
/*
* Name: carStop
* Purpose: Stops the motor by reducing left and
*          right motor speeds to 0 speed PWM.
*/
void carStop(){ // Stop the motor
       digitalWrite(M1_SPEED_PIN, 0);  // 0 speed to stop left wheel
       digitalWrite(M2_SPEED_PIN, 0);   // 0 speed to stop right wheel
       digitalWrite(M1_DIRECTION_PIN, LOW); // Left wheel off
       digitalWrite(M2_DIRECTION_PIN, LOW);  // Right wheel off
       delay(100);  // Delay briefly to show the car has stopped
}
/*
* Name: carForward
* Purpose: Moves the car forward by directing
*          the left/right wheels forward and
*          adjusting speed to speed desired.
*/
void carForward(int leftSpeed, int rightSpeed){  //Move forward
       digitalWrite(M1_DIRECTION_PIN, HIGH); // Left wheel direction forward
       digitalWrite(M2_DIRECTION_PIN, HIGH); // Right wheel direction forward
        analogWrite(M1_SPEED_PIN, leftSpeed);   // Adjust left wheel speed
       analogWrite(M2_SPEED_PIN, rightSpeed*WHEEL_SPEED_OFFSET);   // Adjust right wheel speed
}
/*
* Name: carReverse
* Purpose: Moves the car in reverse by directing
*          the left/right wheels backwards and
*          adjusting speed to speed desired.
*/
void carReverse(int leftSpeed, int rightSpeed){  //Move backward
       digitalWrite(M1_DIRECTION_PIN, LOW);    // Left wheel direction backward
       digitalWrite(M2_DIRECTION_PIN, LOW);  // Right wheel direction backward
       analogWrite(M1_SPEED_PIN, leftSpeed); // Adjust left wheel speed
```

```
        analogWrite(M2_SPEED_PIN, rightSpeed*WHEEL_SPEED_OFFSET);  // Adjust right wheel speed
}
/*
* Name: carTurnLeft
* Purpose: Turns the car left by directing the
*          left wheel backwards and right wheel
*          forwards and adjusting speed to speed desired.
*/
void carTurnLeft(int leftSpeed, int rightSpeed){  //Turn left
        digitalWrite(M1_DIRECTION_PIN, LOW);  // Left wheel direction backward
        digitalWrite(M2_DIRECTION_PIN, HIGH);  // Right wheel direction forward
        analogWrite(M1_SPEED_PIN, leftSpeed);  // Adjust left wheel speed
        analogWrite(M2_SPEED_PIN, rightSpeed);  // Adjust right wheel speed
        delay(TURNING_TIME);  // For our turning pace, 110 ms delay will produce a right angle
turn
}
/*
* Name: carTurnRight
* Purpose: Turns the car right by directing the
*          right wheel backwards and left wheel
*          forwards and adjusting speed to speed desired.
*/
void carTurnRight(int leftSpeed, int rightSpeed){  //Turn right
        digitalWrite(M1_DIRECTION_PIN, HIGH);  // Left wheel forward
        digitalWrite(M2_DIRECTION_PIN, LOW);  // Right wheel backward
        analogWrite(M1_SPEED_PIN, leftSpeed);  // Adjust left wheel speed
        analogWrite(M2_SPEED_PIN, rightSpeed);  // Adjust right wheel speed
        delay(TURNING_TIME);  // For our turning pace, 110 ms delay will produce a right angle
turn
}
```