Bean Pirate – Autonomous Robot for Toxic Can Disposal

By

Denise Cruise

Harshal Charhate

Senthil Chandrasegaran

Tim Sullivan

Abstract

The report contains a basic introduction to the autonomous robot developed for mechatronics lab. The task assigned was to develop an autonomous robot to locate five barrels of toxic chemicals, and carry them to a safe disposal site. The task was simulated in a 7ft by 7ft field using a 12in by 12in robot. This report details the approach taken, experiments and results while building the robot. Arduino code, circuit diagrams, state diagrams and state transition tables are shown in appendices. The financial report about the materials used in developing robot is also included.

Introduction

The main purpose of the autonomous robot would be to pick up some filled cans and deposit them at a specified location. It was supposed to be a small size working model for disposing toxic drums. A track had been provided with black lines that divided the different regions. Another task provided was to measure the temperature of the cans and identify a cold can among them. The cold can was supposed to be deposited after all other cans had been deposited.

We used a crane type model with a gripper at end of arm to pick up the cans. The cans were collected and stacked on the robot. We used line sensors to provide the ability to travel autonomously inside the track. A temperature sensor was attached to the body near where the cans were picked up to register the temperature of each can. The cold can was placed in a different place than the hot cans to allow it to be deposited last.

Body

Robot 1.0

The robot initially designed was supposed to work on the principle of electromagnets. The electromagnet would have attracted the can and held it while the boom would move and deposit the can on top of the robot, which would act as a carousel for all of the cans.

Two DC brushed motors were used to allow the robot to travel. They have a gear ratio of 1:75, and have a large amount of torque. This was important to us because we needed the robot to continue to travel even with all five cans (five pounds) loaded on the carousel. Both motors were able to move in forward and reverse, which gave us the option of moving the robot forward, moving in reverse, and spinning the motors in different directions to make 0 radius turns.

The line follower had been designed using 3 line sensors placed at right angles to each other. One line sensor was placed in the front part of the robot, and the other two sensors were placed on the same line as the drive motors, one near the left motor and the other near the right motor. It was a very simple scheme, and it assumed that the robot followed the line accurately. We realized that if the robot did not follow the line accurately, we would need additional line sensors to let it follow the same line.

The Herkulex servo would be attached to the boom to allow it to rotate between 0 to 320 degrees. This gave us a specific range where the cans could be deposited onto the body of the robot. The front end of the robot would be in a V-arc shape to gather the cans while they moved inwards into the robot. An IR sensor was used to sense if a can had reached proper position. The moment the can got in correct position it would be lifted.

The robot would be following the direct route to get to the collection zone. It would then travel along the black line that is the entire perimeter of collection zone. Then it would take an indirect route back to the deposit point.

Robot 2.0

The electromagnet was replaced by a gripper. The gripper would grab the can tightly and a servo motor with a pulley attached would lift it. Another servo motor, the Herkulex, would cause the boom to rotate and the can would then be deposited at a specified angle. Our strategy was to gather all of the cans, place them onto the body of the robot, travel to the hole, and then deposit them all into the hole at once.

An additional goal of the project was to identify the cold can, and to place it last into the hole. We planned to identify the cold can with an IR temperature sensor, and then we would place that can in a specific place on the carousel so that it could be the last can off of the carousel.

The line follower was updated to use 5 line sensors. Apart from the initial 3 sensors, two additional sensors were added on the left and right of the front sensor. The additional two sensors dramatically improved the robot's ability to do line corrections.

The original V-arc shape at the front section of the robot was modified so that only half of the robot had the arc for receiving the cans. The change was made since only half of the robot was inside the track at any time for collection. This was a positive change because the angle would be smaller, therefore the cans would move inward faster which would be more efficient.

Robot 3.0

During the testing phase of the robot design, we determined that we needed more line following sensors in order to enable the robot to follow straight lines and to make ninety degree turns. An array of six line sensors was added, and further testing was completed. We had a total of 13 line sensor available for our use, but in the final robot, we used nine of them to maximize control.

The Herkulex servo motor was replaced with the Dynamixel servo motor. This servo was very similar to the Herkulex, so it was an easy replacement, and still worked on the same general principle.

Originally we had planned to use an Arduino Uno as our microcontroller for the robot, but we were having some issues getting the Herkulex servo to work properly with everything else

connected. The Herkulex functioned via serial communication, and we had no problems with it when it was the only thing running on the Arduino, but when everything was connected, there were problems. To alleviate this problem, we changed from an Arduino Uno to an Arduino Mega. This ended up being a very positive change because it made serial communication very simple, there were specific pins for SCL and SDA, which is what the temperature sensor needed, and there were significantly more digital and analog pins available for our use.

The robot was powered by an 11.1 Volt Lipo battery. We had performed all testing using a power supply at 9 Volts, and initially planned on using six AA batteries, but because all of the motors were pulling such a large amount of current, we knew that we would be better off using Lipo batteries. The change from 9 Volts to 11.1 Volts was acceptable, because all of the components were able to run off of that voltage; however, we did see some changes from the higher voltage, such as the robot struggling with line following because it was now moving at a faster speed. Overall, we felt that the Lipo batteries were a good choice, but we should have completed more of the testing phase using those batteries rather than the power supply that we did use.

The final design can be seen in Figure 1.

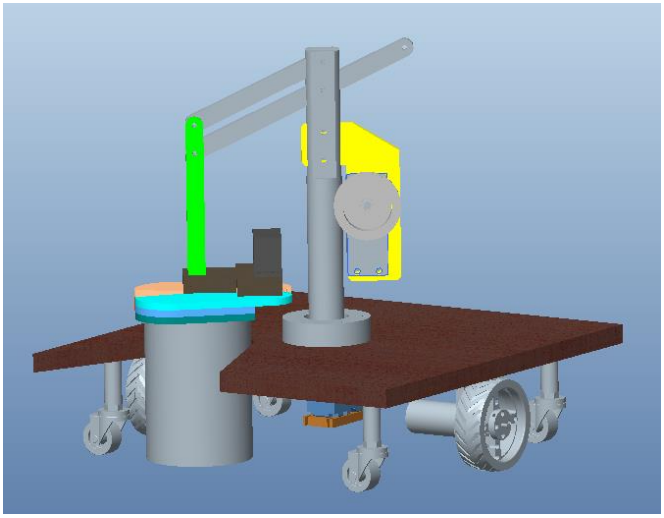


Figure 1: CAD Model showing final design of robot

Results

The testing phase of our robot building process included testing each component individually as well as testing the integrated product. This phase was extremely crucial to the overall success of the robot, and it took up a majority of the time spent on the robot.

When we received the electromagnet, we immediately tested it to make sure that it could pick up a can. Supplying the voltage specified on the datasheet, we powered the magnet, and put it over a can. It was not strong enough to pick it up. We initially tried supplementing the electromagnet

with several small permanent magnets in order to increase the lifting power. This helped significantly, and we were able to lift a can several times, but it was not consistent, and it was very sensitive to the location of the magnet on the can. The magnet had to be exactly in the center where the metal was flat without ridges, or it was not strong enough to pick it up. Because we were planning on holding the can with the magnet while we rotated to place the can, we decided that the magnet may not be the best approach. We took an alternative route of using the gripper with the servo motor as a way to grab and lift the cans.

We had initially determined specific angles that the cans would be placed at on the robot to ensure that they all would fit properly; however once we attached the Herkulex, we found that some angles did not work because the motors would run into the cans on the carousel during rotation. Because of this, we had to go through iterations of testing to determine which angles we would place the cans at on the carousel. The Herkulex was capable of 320 degrees of rotation, with a range of -160 to 160, but we found that we only needed 305 degrees for our application. The final angles that we decided upon are shown in Figure 2.

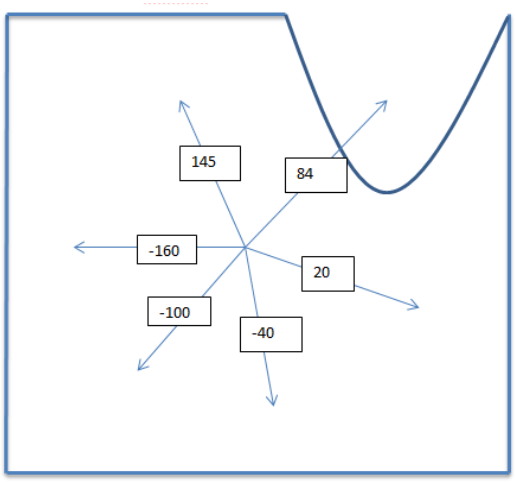


Figure 2: Can Placement Angles for Herkulex Servo

After our Herkulex motor stopped working, and we switched to the Dynamixel, we had to retest the angles of can placement. The Dynamixel had a slightly smaller range of motion, only being able to move 300 degrees. The input was a number between 0 and 1023, so we converted all of the angles that we had found from the Herkulex to this new range, and ran initial tests. It worked very well, so we were able to use these specific angles shown in Figure 3 for the final product.

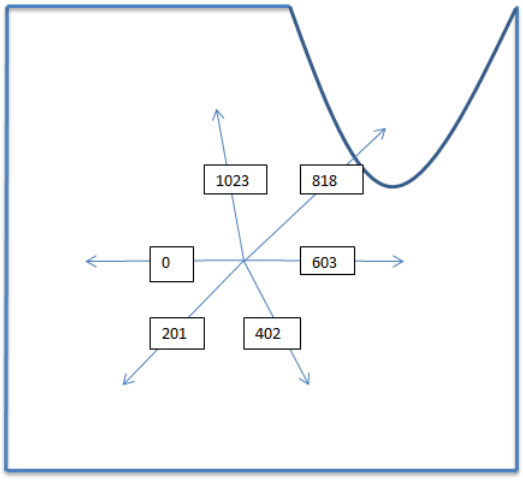


Figure 3: Can Placement Angles for Dynamixel Servo

In addition to testing for the angles of can placement, we also needed to test the winch motor to determine which values to use for the various heights of the boom. For example, we needed a specific height to pick up the boom, a height to travel with the can to its position, and then the necessary height to place the can onto the carousel. This process was very iterative and took a long time to perfect, but the final results of this testing can be seen in Table 1

Table 1: Positions for Winch Servo

Position Name	Position Description	Input to Servo
Initial Position	This position is parallel to the ground and ensures our robot fits within the 12" height requirement	38
Low	This position is used to pick up cans from the ground	45
Mid	This position is used to place cans on the carousel	35
High	This position is used during rotations when there is not a can in the gripper	30
High with Can	This position is used during rotations when there is a can in the gripper	23

The process of getting our robot to follow lines was extremely iterative. We had to decide if we wanted a scheme that would follow a line quickly with very little oscillation, but struggle to return to the line if it happened to wander, or a scheme that oscillated back and forth more frequently, resulting in more robust line following, but slower travel. In the end we decided to allow our robot to oscillate, therefore travel more slowly, because we thought it was more important to have extremely robust line following abilities.

The final line sensor layout is shown below in Figure 4. Having the two sensor in front of the array of line sensor improved our ability to determine if we had drifted from the line, and then the array of sensors could tell the robot how to correct. In addition to testing to determine how to place the sensors, we also completed a significant amount of testing to determine the proper

threshold values for the sensors. Each sensor had to be tested individually to find its threshold because none of them were exactly the same. The results from this testing can be seen in Table 2.

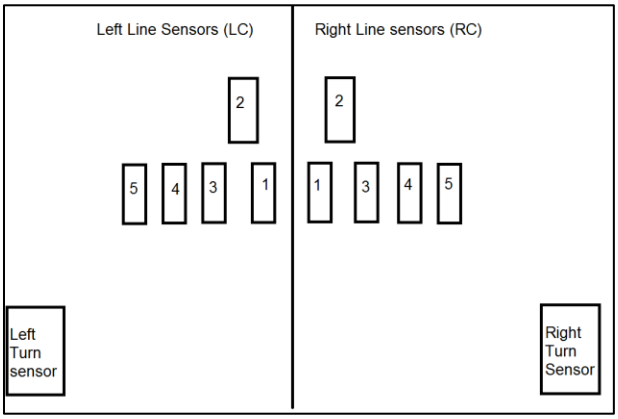


Figure 4: Line Sensor Layout

Table 2: Line Sensor Testing Results

Line sensors Active	Left Motor PWM	Right Motor PWM	Left Motor Direction	Right Motor Direction
LC4	200	200	Fwd	Bkw
LC3 LC4	200	100	Fwd	Bkw
LC2 LC3 LC4	200	0	Fwd	Fwd
LC2 LC3	200	50	Fwd	Fwd
LC1 LC2 LC3	200	100	Fwd	Fwd
LC1 LC2	200	150	Fwd	Fwd
LC1 LC2 RC1	200	175	Fwd	Fwd
LC1 LC2 RC1 RC2	200	200	Fwd	Fwd
LC1 RC1 RC2	175	200	Fwd	Fwd
RC1 RC2	150	200	Fwd	Fwd
RC1 RC2 RC3	100	200	Fwd	Fwd
RC2 RC3	50	200	Fwd	Fwd
RC2 RC3 RC4	0	200	Fwd	Fwd
RC3 RC4	100	200	Bkw	Fwd
RC4	200	200	Bkw	Fwd

We built our robot and completed all testing in the Spira Labs where two of our team members work. This was extremely convenient because a lot of equipment was available for our use; however, this also meant that the testing was completed in different conditions than the final competition would take place in. We did go to the location of the final competition to do testing, and found that many things had to change in order for our robot to function properly.

We had predicted that some code would have to change due to different conditions for the final competition, but we did not expect design changes to be necessary. When we initially tested on the final competition arena, we found that the friction force was much greater than it had been in our testing area, so the cans were getting knocked over by the robot. To account for this, we added to pieces of wood to the front of the robot, so that if the robot hit a can, it was would just push it forward instead of knocking it over.

Discussion

Most lessons learned from such project-based courses are more from the things that did not work than the things that did. Our project was no different. We had several issues, problems that, in hindsight, could have been avoided, but as a first attempt at a prototype, were inevitable. Some of these issues are listed below.

Human error: Due to the power required for the robot, and the speed at which our AA batteries were being consumed, all tests were performed by connecting the robot to a power supply unit. Hitting the wrong button on the unit resulted in a high voltage supplied, which may have resulted in the Herkulex motor failing.

Line following: Our unfamiliarity with PID controllers resulted in us writing a line following code that was closer to a proportional control. This required several long days of fine-tuning, and was inherently unstable. It became necessary to implement further checks and balances in our code to compensate for this instability, which increased the complexity and debugging time for the code. This instability and sensitivity to the operating conditions also resulted in having to completely re-tune the line following code when (a) we moved the robot to the demonstration arena, which, as mentioned earlier, had completely different surface properties, and (b) when we replaced the power supply unit with the powerful Lipo battery.

Last-minute setbacks: The Herkulex motor failing at an inopportune moment required us to look for other alternatives, and we settled on the Dynamixel AX-12A motor. Working out the required circuitry, such as the HC 1026 and HC-04 chips, and code for the motor set us back by an an entire day, which proved critical given how close to the project deadline this occurred.

Piecewise testing vs. system testing: We had different segments of code and different circuits set up for testing the temperature sensing, the can pick-up servo coordination, and the line following. These performed well individually, but when integrated, ran into problems. While some of the minor problems could be solved with a restructuring of the code, other issues such as noise to the digital pins, especially for winch and gripper servos, could not be solved. We narrowed the problem down to two issues – a library conflict between the Arduino standard servo library, and the library for the Dynamixel servo, or sensor noise, resulting in unwanted signals being sent to the winch and gripper servos. We believe the failure of our temperature sensor in the integrated system was due to similar factors.

It has to be noted that in spite of all these issues arising at the final hours, we still managed to overcome most of them, and our best run was prevented being a perfect run at the very end due to the earlier mentioned problem with the winch servo. This relative success was due to the detailed planning, design and testing that we conducted earlier. It is easy to ignore the problems we *did not* have, but it is critical to make note of them. We did not have problems of fit or interference during assembly because the person responsible for the manufacturing was also involved in the design process, and ensured manufacturability of the components. We also performed detailed checks for interference and range of motion, especially since our design relied heavily on the boom linkage working as planned. Bought-out components, including sensors, motors, etc. were checked carefully for power requirements and compatibility, and these resulted in many aspects of the build going very smoothly.

Conclusion

This project of building an autonomous robot to execute a specific task introduced us to the complexity of building mechatronic systems, and to the world of electromechanical prototyping using the Arduino board. We put the concepts learned in the classes and the labs to good use in our final project, and encountered the challenges of working in a multi-disciplinary team. Apart from technical lessons learned, we experienced the importance of planning and early system integration and testing for a successful outcome. In spite of the setbacks that our team faced, we were still appreciated for our design and were voted as “the best in show” by our peers.

Appendix 1 – Arduino code

```
//////////////////////////////////////////////
// ME 588 Final Project : Bean Pirate       //
// Code by:                                 //
//      Denise Cruise                       //
//      Tim Sullivan                        //
//      Senthil Chandrasegaran              //
//////////////////////////////////////////////

#include <Servo.h>  // libraries for winch and gripper servos
#include <i2cmaster.h> // library for temperature sensor
#include <DynamixelSerial1.h> // library for boom servo

// Define the winch and gripper servos
Servo servoMain; // gripper
Servo servoWinch; // winch

// This is the spatial arrangement of the line following sensors:
// LCn, RCn, L, R, and RT are identifiers for the sensors in the code
// ------------------------------------
//          FRONT OF ROBOT
// ------------------------------------
//            LC2     RC2
//  LC5 LC4 LC3 LC1 RC1 RC3 RC4 RC5
//                             RT
//
//
//
// L                                R
// ------------------------------------
//           BACK OF ROBOT
// ------------------------------------
// Two line sensors up front -- LC2 and RC2
int LeftControlPin2 = A3;
int RightControlPin2 = A4;

// Array of line sensors LC1, LC3 - LC5, RC1, RC3 - RC5
int LeftControlPin5 = A8;
int LeftControlPin4 = A9;
int LeftControlPin3 = A10;
int LeftControlPin1 = A11;
int RightControlPin1 = A12;
int RightControlPin3 = A13;
int RightControlPin4 = A14;
int RightControlPin5 = A15;

// turning pins for sensing intersections
// for sensing the beginning and end of turn (used for turn 3)
int RightTurnSensorPin = A0;
int LeftSensorPin = A1;
int RightSensorPin = A2;
```

```
// Drive motor pins -- these motors drive the robot
int LeftEnablePin = 6;
int RightEnablePin = 7;
int LeftMotorPin1 = 5;
int LeftMotorPin2 = 4;
int RightMotorPin1 = 3;
int RightMotorPin2 = 2;

// LEDS -- these were used for indicating the number of turns done
// to the viewer. Counting is done in binary.
// Pins are ordered as follows:
//        0      1     2      3     -- LEDs numbered on the Robot
//      pin4    pin1  pin2   pin3   -- Corresponding Pin variables
// Turn counts:
//        0      0     0      1     -- Turn 1
//        0      0     1      0     -- Turn 2
//        0      0     1      1     -- Turn 3
//        0      1     0      0     -- Turn 4
//        0      1     0      1     -- Turn 5
//
int ledpin1 = 24;
int ledpin2 = 26;
int ledpin3 = 28;
int ledpin4 = 22;
int pin1 = LOW;
int pin2 = LOW;
int pin3 = LOW;
int pin4 = LOW; // pin4 is marked 0 on the robot

// variables for reading line sensor values.
int LC1sensor;
int LC2sensor;
int LC3sensor;
int LC4sensor;
int LC5sensor;
int RC1sensor;
int RC2sensor;
int RC3sensor;
int RC4sensor;
int RC5sensor;
int RTsensor;
int Lsensor;
int Rsensor;

// Calibrated threshold values for line sensors.
// A value above this threshold means the sensor is over a black line
int LC1Threshold = 456;
int LC2Threshold = 600;
int LC3Threshold = 519;
int LC4Threshold = 504;
int LC5Threshold = 581;
int RC1Threshold = 433;
int RC2Threshold = 600;
int RC3Threshold = 426;
int RC4Threshold = 444;
```

```
int RC5Threshold = 594;

int RTThreshold = 918;
int LThreshold = 982;
int RThreshold = 982;

// speeds in line following. This is the variable passed to
// The lineFollow() function.
int followspeed;
// drive motor speeds
int turningspeed = 65; // motor speed when robot is taking a turn
int speedReduction = 0; // dynamic speed reduction for turn 3
int travelspeed; // the variable used inside the lineFollow() function,
//                  corresponding to the followspeed variable.

int TurnsDone = 0;  // Number of turns done by the robot
int InTurn = 0;     // Check variable for turn 3.
int TT = 0;
int d = 0;
int count = 0;
// TT, count, and d are variables for checking if a robot has finished a
// turn and is consistently back to the lineFollow() state. This is
// required since the robot switches back and forth between turning and
// linefollowing at the end of a turn, making the turn count unreliable


// debouncing variables. Check filteredRead() function for details.
int reading;
int lastReading;
int actualReading;
int stable;
long lastDebounceTime = 0;
long debounceDelay = 15;

//can sensing & placing variables
int can_number = 1; // initialize can count
int cold_can_done = 0; // setting switch for when cold can is done
int cans_left; // counting down during can placement
int can_temp;  // Will identify which can is cold
int placement_angle = 0; // position of can placed on the robot
int pickup_angle = 818;  // boom orientation for picking up can
int travel_time = 0; // herkulex travel time with can
int travel_time_empty = 0; //herkulex travel time without can

// variables for winch servo positions.
// lower numbers mean higher positions
int high = 30; // default position
int highwithcan = 23; // high position when a can is being carried
int mid = 35;         // can placing/ pickup on the robot base
int initialpos = 38;  // keeps boom horizontal at start
int low = 45;         // can placing/ pickup from the floor
```

```
// variables for gripper servo positions
// a lower number means a higher angle between the gripper jaws.
int closes = 125;
int opens = 85;

// Temperature sensing variables
int sensorpin = 45; //pin for the photoresistor
int senseval; //variable that will hold result of photoresistor
int warm = 72;
int cold = 30;
int Temperature,Voltage,Position;
int dev = 0x5A<<1;
int data_low = 0;
int data_high = 0;
int pec = 0;

void setup(){
    delay(1000);
    pinMode(LeftControlPin5, INPUT);
    pinMode(LeftControlPin4, INPUT);
    pinMode(LeftControlPin3, INPUT);
    pinMode(LeftControlPin2, INPUT);
    pinMode(LeftControlPin1, INPUT);
    pinMode(RightControlPin1, INPUT);
    pinMode(RightControlPin3, INPUT);
    pinMode(RightControlPin4, INPUT);
    pinMode(RightControlPin5, INPUT);
    pinMode(RightControlPin2, INPUT);
    pinMode(RightSensorPin, INPUT);
    pinMode(RightTurnSensorPin, INPUT);
    pinMode(LeftSensorPin, INPUT);
    pinMode(LeftMotorPin1, OUTPUT);
    pinMode(LeftMotorPin2, OUTPUT);
    pinMode(RightMotorPin1, OUTPUT);
    pinMode(RightMotorPin2, OUTPUT);
    pinMode(LeftEnablePin, OUTPUT);
    pinMode(RightEnablePin, OUTPUT);
    pinMode(ledpin1,OUTPUT);
    pinMode(ledpin2,OUTPUT);
    pinMode(ledpin3,OUTPUT);
    pinMode(ledpin4,OUTPUT);
    servoMain.attach(33); // servo on digital pin 33
    servoWinch.attach(31); // winch on digital pin 31
    servoMain.write(closes);
    servoWinch.write(initialpos);
    delay(1000);
    servoWinch.write(mid);
    //Setup i2c communication
    i2c_init(); //Initialise the i2c bus--Pins 20 (SDA) and 21 (SCL).
    PORTC = (1 << PORTC4) | (1 << PORTC5);//enable pullups
    //Setup Dynamixel
    Serial.begin(9600); //Begin Serial Communication

    //Initialize the servo at 1Mbps and Pin control 13
    Dynamixel.begin(1000000,13);
```

```
    //empty boom moves over to pickup point
    Dynamixel.moveSpeed(1,pickup_angle,600);
}

void loop()
{
    // make sure the gripper jaws remain closed. Need to pass this in
    // every iteration of the loop to maintain them closed.
    servoMain.write(closes);

    // Make sure the Dynamixel stays un the pickup angle (over the slot
    // on the base) at all times, except when boom is in action
    Dynamixel.moveSpeed(1,pickup_angle,600);

    // Make sure the winch servo is in the mid position at all times
    // except when the boom is in action.
    servoWinch.write(mid);

    // Conditions for turning LEDs on
    if (pin1 == HIGH) {digitalWrite(ledpin1, HIGH);}
    else if (pin1 == LOW) {digitalWrite(ledpin1,LOW);}

    if (pin2 == HIGH) {digitalWrite(ledpin2, HIGH);}
    else if (pin2 == LOW) {digitalWrite(ledpin2,LOW);}

    if (pin3 == HIGH) {digitalWrite(ledpin3, HIGH);}
    else if (pin3 == LOW) {digitalWrite(ledpin3,LOW);}

    if (pin4 == HIGH) {digitalWrite(ledpin4, HIGH);}
    else if (pin4 == LOW) {digitalWrite(ledpin4,LOW);}

    // Read line sensor values using the function
    // for debouncing the readings.
    Rsensor = filteredRead(RightSensorPin, RThreshold);
    Lsensor = filteredRead(LeftSensorPin, LThreshold);
    LC5sensor = filteredRead(LeftControlPin5, LC5Threshold);
    LC4sensor = filteredRead(LeftControlPin4, LC4Threshold);
    LC3sensor = filteredRead(LeftControlPin3, LC3Threshold);
    LC2sensor = filteredRead(LeftControlPin2, LC2Threshold);
    LC1sensor = filteredRead(LeftControlPin1, LC1Threshold);
    RC1sensor = filteredRead(RightControlPin1, RC1Threshold);
    RC2sensor = filteredRead(RightControlPin2, RC2Threshold);
    RC3sensor = filteredRead(RightControlPin3, RC3Threshold);
    RC4sensor = filteredRead(RightControlPin4, RC4Threshold);
    RC5sensor = filteredRead(RightControlPin5, RC5Threshold);
    RTsensor = filteredRead(RightTurnSensorPin, RTThreshold);

    // Initialize line following speed. For all straights,
    // use the same value (120), but for the short straight after
    // the second turn, use a lower speed (better for stability of
    // line following and accuracy of turning)
    if (TurnsDone == 2) {
        followspeed = 75;
    }
```

```
else {
    followspeed = 120;
}

// STOP WHEN ROBOT REACHES THE HOLE, DUMP CANS AND STOP COMPLETELY
//
// TT == 0 only when the previous turn is completely done and the
// robot has been line following for a while.
// To see the part of the code that sets TT to 0, check the
// lineFollow() function
if ((TurnsDone == 5) &&
  (LC5sensor > LC5Threshold ||
   RC5sensor > RC5Threshold) &&  // these sensors sense the hole
  (TT == 0)) {
    dumpCans();
    while (1<2){}; // stop completely after dumping cans
}

// IF 4 TURNS ARE DONE AND FRONT LINE SENSORS ALL GO OFF,
// TAKE A LEFT TURN (THIS IS THE 5th TURN, TURNING LEFT)
else if ((TurnsDone == 4) &&
       (LC1sensor < LC1Threshold &&
        LC2sensor < LC2Threshold &&
        RC1sensor < RC1Threshold &&
        RC2sensor < RC2Threshold) &&
       (TT == 0)) {
    turnSecondLeft();

    // Set LEDs to count 5
    pin1 = HIGH;
    pin2 = LOW;
    pin3 = HIGH;
    recover();  // this function stops and centers the robot over
                // the line sensors.
}

// IF 3 TURNS ARE DONE AND FRONT LINE SENSORS ALL GO OFF,
// TAKE A LEFT TURN (THIS IS THE 4th TURN, TURNING LEFT)
else if ((TurnsDone == 3) &&
        (LC1sensor < LC1Threshold &&
         LC2sensor < LC2Threshold &&
         RC1sensor < RC1Threshold &&
         RC2sensor < RC2Threshold) &&
        (TT == 0)) {
    turnFirstLeft();
    recover();

    // Set LEDs to count 4
    pin1 = HIGH;
    pin2 = LOW;
    pin3 = LOW;
}
```

```
    // RIGHT TURN (TURN NO. 3) FOR SWEEPING BACK ACROSS CAN FIELD
    // IF 2 TURNS ARE DONE AND ONE OF THE EXTREME RIGHT OR THE RIGHT
    // TURN SENSING SENSORS GO OFF, TAKE A LEFT TURN (THIS IS THE 3rd
    // TURN, TURNING RIGHT)
      else if (TurnsDone == 2 &&
              (RC4sensor > RC4Threshold ||
               RC5sensor > RC5Threshold ||
               RTsensor > RTThreshold) &&
              TT == 0){
        turnThirdRight();

        // Set LEDs to count 3
        pin1 = LOW;
        pin2 = HIGH;
        pin3 = HIGH;
      }

    // RIGHT TURN (TURN NO. 2) FOR SWEEPING FORWARD ACROSS CAN FIELD
    // IF 1 TURN IS DONE AND FRONT LINE SENSORS ALL GO OFF,
    // TAKE A RIGHT TURN (THIS IS THE 2nd TURN, TURNING RIGHT)
      else if  (TurnsDone == 1 &&
                (LC1sensor < LC1Threshold &&
                 LC2sensor < LC2Threshold &&
                 RC1sensor < RC1Threshold &&
                 RC2sensor < RC2Threshold)) {
        turnSecondRight();

        // Set LEDs to count 2
        pin1 = LOW;
        pin2 = HIGH;
        pin3 = LOW;
      }

    // IF NO TURN IS DONE AND FRONT LINE SENSORS ALL GO OFF,
    // TAKE A RIGHT TURN (THIS IS THE 1st TURN, TURNING RIGHT)
    else if  (TurnsDone == 0 &&
             (LC1sensor < LC1Threshold &&
              LC2sensor < LC2Threshold &&
              RC1sensor < RC1Threshold &&
              RC2sensor < RC2Threshold)) {
    turnFirstRight();

    // Set LEDs to count 1
    pin1 = LOW;
    pin2 = LOW;
    pin3 = HIGH;
    }

    // IF NONE OF THE ABOVE, CONTINUE FOLLOWING THE LINE
    else {lineFollow(followspeed);}
}
```

```
//FUNCTIONS DEFINED HERE:

// The line following function takes in a value of the motor speed and
// sends its value to the motor enable pin.
void lineFollow(int travelspeed)
{
    // Make sure the gripper is closed when the robot is following the
    // line.
    servoMain.write(closes);

    // If the laser trip wire is closed, stop immediately by reversing
    // the motors for a very short time (50 ms)
    senseval = digitalRead(sensorpin); //read in photoresistor value
    if (senseval == 0) {
         digitalWrite(LeftMotorPin1, LOW);   // Reverse
         digitalWrite(LeftMotorPin2, HIGH);
         digitalWrite(RightMotorPin1, LOW);   // Reverse
         digitalWrite(RightMotorPin2, HIGH);
         analogWrite(LeftEnablePin, 20); //Stop suddenly
         analogWrite(RightEnablePin, 20); //Stop suddenly
         delay(50);

         // after stopping the robot, lift the can by calling the
         // liftCan function
         liftCan();

         // once the can is in place, orient the motors to center it
         // over the line.
         recover();
    }

    // if no can is found, then read in line sensor values and drive the
    // robot based on the conditions listed below.
    LC1sensor = filteredRead(LeftControlPin1, LC1Threshold);
    RC1sensor = filteredRead(RightControlPin1, RC1Threshold);

    // if both center sensors are high, keep moving forward at the
    // specified speed (which is passed as argument to the lineFollow
    // function.
    if ((LC1sensor > LC1Threshold) && (RC1sensor > RC1Threshold)) {
        digitalWrite(LeftMotorPin1, HIGH);   // Forward
        digitalWrite(LeftMotorPin2, LOW);
        digitalWrite(RightMotorPin1, HIGH);  // Forward
        digitalWrite(RightMotorPin2, LOW);
        analogWrite(LeftEnablePin, travelspeed);
        analogWrite(RightEnablePin, travelspeed);
        d = d + 1; //every time the middle sensor goes high, count up
        if (TT == 1 && d > 30) { // once d hits 20, you can assume it's
                                 // consistently line following
        TT = 0;   // if the robot is consistently follwing the line,
                  // then let the next turning function know it is a
                  // valid turn (done by setting TT as 0.
        }
    }
```

```
// If the sensor to the right of center is on, move slower, but
// continue moving forward.
RC2sensor = filteredRead(RightControlPin2, RC2Threshold);
if ((RC2sensor > RC2Threshold)) {
    digitalWrite(LeftMotorPin1, HIGH);   // Forward
    digitalWrite(LeftMotorPin2, LOW);
    digitalWrite(RightMotorPin1, HIGH);  // Forward
    digitalWrite(RightMotorPin2, LOW);
    analogWrite(LeftEnablePin, 60);
    analogWrite(RightEnablePin, 60);
}

// If the sensor to the left of center is on, move slower, but
// continue moving forward.
LC2sensor = filteredRead(LeftControlPin2, LC2Threshold);
if ((LC2sensor > LC2Threshold)) {
    digitalWrite(LeftMotorPin1, HIGH);   // Forward
    digitalWrite(LeftMotorPin2, LOW);
    digitalWrite(RightMotorPin1, HIGH);  // Forward
    digitalWrite(RightMotorPin2, LOW);
    analogWrite(LeftEnablePin, 60); //Fast
    analogWrite(RightEnablePin, 60); //Fast
    //Serial.println("RC2");
}

// If the third sensor on the right is on, then correct by spinning
// the left motor forward and the right motor in reverse
RC3sensor = filteredRead(RightControlPin3, RC3Threshold);
if ((RC3sensor > RC3Threshold)) {
    digitalWrite(LeftMotorPin1, HIGH);    // Forward
    digitalWrite(LeftMotorPin2, LOW);
    digitalWrite(RightMotorPin1, LOW);    // Reverse
    digitalWrite(RightMotorPin2, HIGH);
    analogWrite(LeftEnablePin, 60);
    analogWrite(RightEnablePin, 60);
}

// If the third sensor on the left is on, then correct by spinning
// the right motor forward and the left motor in reverse
LC3sensor = filteredRead(LeftControlPin3, LC3Threshold);
if ((LC3sensor > LC3Threshold)) {
    digitalWrite(LeftMotorPin1, LOW);   // Reverse
    digitalWrite(LeftMotorPin2, HIGH);
    digitalWrite(RightMotorPin1, HIGH);  // Forward
    digitalWrite(RightMotorPin2, LOW);
    analogWrite(LeftEnablePin, 60); //Fast
    analogWrite(RightEnablePin, 60); //Fast
}

// If one or more of the extreme left, or one or more of the extreme
// right sensors go high, then really slow down the robot, but keep
// moving forward
LC5sensor = filteredRead(LeftControlPin5, LC5Threshold);
LC4sensor = filteredRead(LeftControlPin4, LC4Threshold);
RC4sensor = filteredRead(RightControlPin4, RC4Threshold);
```

```
    RC5sensor = filteredRead(RightControlPin5, RC5Threshold);
    if (((LC4sensor > LC4Threshold) &&
         (LC5sensor > LC5Threshold)) ||
        (LC5sensor > LC5Threshold)    ||
        ((RC4sensor > RC4Threshold) &&
         (RC5sensor > RC5Threshold))  ||
        (RC5sensor > RC5Threshold)) {
            digitalWrite(LeftMotorPin1, HIGH);   // Forward
            digitalWrite(LeftMotorPin2, LOW);
            digitalWrite(RightMotorPin1, HIGH);  // Forward
            digitalWrite(RightMotorPin2, LOW);
            analogWrite(LeftEnablePin, 40); //Fast
            analogWrite(RightEnablePin, 40); //Fast
        }

    // If all three sensors on the left and all three sensors on the
    // right (counting outwards from the center) go low, then call the
    // recover function to orient the robot till the middle sensors go
    // high.
    LC1sensor = filteredRead(LeftControlPin1, LC1Threshold);
    LC2sensor = filteredRead(LeftControlPin2, LC2Threshold);
    RC1sensor = filteredRead(RightControlPin1, RC1Threshold);
    RC2sensor = filteredRead(RightControlPin2, RC2Threshold);
     if ((LC1sensor < LC1Threshold &&
          LC2sensor < LC2Threshold &&
          LC3sensor < LC3Threshold &&
          RC1sensor < RC1Threshold &&
          RC2sensor < RC2Threshold &&
          RC3sensor < RC3Threshold)){
         recover();
     }
}

// FUNCTION FOR RIGHT TURN 1
void turnFirstRight(){
    // Stop suddenly by spinning the wheels in reverse for a very
    // short time (50 ms)
    digitalWrite(LeftMotorPin1, LOW);   // Reverse
    digitalWrite(LeftMotorPin2, HIGH);
    digitalWrite(RightMotorPin1, LOW);   // Reverse
    digitalWrite(RightMotorPin2, HIGH);
    analogWrite(LeftEnablePin, 20); //Stop suddenly
    analogWrite(RightEnablePin, 20); //Stop suddenly
    delay(50);

    // This function is called when all front sensors go low,
    // which happens when a corner is found.
    // Continue turning right until the two center sensors read
    // high again.
    while (LC1sensor < LC1Threshold &&
           RC1sensor < RC1Threshold) {
        // Make sure the gripper is closed when the robot is
        // following the line.
        servoMain.write(closes);
```

```
        // If the laser trip wire is closed, stop immediately by
        // reversing the motors for a very short time (50 ms)
        senseval = digitalRead(sensorpin);
        delay(10);
        if (senseval == 0) {
            digitalWrite(LeftMotorPin1, LOW);   // Reverse
            digitalWrite(LeftMotorPin2, HIGH);
            digitalWrite(RightMotorPin1, LOW);   // Reverse
            digitalWrite(RightMotorPin2, HIGH);
            analogWrite(LeftEnablePin, 20); //Stop suddenly
            analogWrite(RightEnablePin, 20); //Stop suddenly
            delay(50);
            liftCan();
            recover();
        }

        // Turn the robot right by spinning wheels in opp. directions
        digitalWrite(LeftMotorPin1, HIGH);   // Forward
        digitalWrite(LeftMotorPin2, LOW);
        digitalWrite(RightMotorPin1, LOW);   // Reverse
        digitalWrite(RightMotorPin2, HIGH);
        analogWrite(LeftEnablePin, turningspeed);
        analogWrite(RightEnablePin, turningspeed);

        // Read in sensors for the next iteration of the while loop
        LC2sensor = filteredRead(LeftControlPin2, LC2Threshold);
        LC1sensor = filteredRead(LeftControlPin1, LC1Threshold);
        RC2sensor = filteredRead(RightControlPin2, RC2Threshold);
        RC1sensor = filteredRead(RightControlPin1, RC1Threshold);

        d = 0; // every time the turn occurs, this is reset
        // this is to figure out if the switching between
        // turning and line following is complete.
    }

    // check if this function has been called after consistent line
    // following by checking if the value of TT has been set to 0. If
    // so, increment the number of turns by 1.
    if (TT == 0){
        count = count + 1;
        TT = 1;
        TurnsDone = count;
    }
}

// FUNCTION RIGHT TURN FOR TURN NUMBER 2
void turnSecondRight(){
    // Stop suddenly by spinning the wheels in reverse for a very
    // short time (50 ms)
    digitalWrite(LeftMotorPin1, LOW);   // Reverse
    digitalWrite(LeftMotorPin2, HIGH);
    digitalWrite(RightMotorPin1, LOW);   // Reverse
    digitalWrite(RightMotorPin2, HIGH);
    analogWrite(LeftEnablePin, 20); //Stop suddenly
    analogWrite(RightEnablePin, 20); //Stop suddenly
```

```
delay(50);

// This function is called when all front sensors go low,
// which happens when a corner is found.
// Continue turning right until the two center sensors read
// high again.
while (LC1sensor < LC1Threshold &&
       RC1sensor < RC1Threshold) {

    // Make sure the gripper is closed when the robot is
    // following the line.
    servoMain.write(closes);

    // If the laser trip wire is closed, stop immediately by
    // reversing the motors for a very short time (50 ms)
    senseval = digitalRead(sensorpin); //read in photoresistor value
    if (senseval == 0) {
        digitalWrite(LeftMotorPin1, LOW);   // Reverse
        digitalWrite(LeftMotorPin2, HIGH);
        digitalWrite(RightMotorPin1, LOW);   // Reverse
        digitalWrite(RightMotorPin2, HIGH);
        analogWrite(LeftEnablePin, 20); //Stop suddenly
        analogWrite(RightEnablePin, 20); //Stop suddenly
        delay(50);
        liftCan();
        recover();
    }

    // Turn the robot right by spinning wheels in opp. directions
    digitalWrite(LeftMotorPin1, HIGH);   // Forward
    digitalWrite(LeftMotorPin2, LOW);
    digitalWrite(RightMotorPin1, LOW);   // Reverse
    digitalWrite(RightMotorPin2, HIGH);
    analogWrite(LeftEnablePin, turningspeed); //Slow
    analogWrite(RightEnablePin, turningspeed); //Slow

    // Read in sensors for the next iteration of the while loop
    LC2sensor = filteredRead(LeftControlPin2, LC2Threshold);
    LC1sensor = filteredRead(LeftControlPin1, LC1Threshold);
    RC2sensor = filteredRead(RightControlPin2, RC2Threshold);
    RC1sensor = filteredRead(RightControlPin1, RC1Threshold);

    d = 0; // every time the turn occurs, this is reset
    // this is to figure out if the switching between
    // turning and line following is complete.
}

// check if this function has been called after consistent line
// following by checking if the value of TT has been set to 0. If
// so, increment the number of turns by 1.
if (TT == 0){
    count = count + 1;
    TT = 1;
    TurnsDone = count;
}
```

```
}

// FUNCTION FOR TURNING RIGHT (THIRD TURN)
void turnThirdRight() {
    // Stop suddenly by spinning the wheels in reverse for a very
    // short time (50 ms)
    digitalWrite(LeftMotorPin1, LOW);   // Reverse
    digitalWrite(LeftMotorPin2, HIGH);
    digitalWrite(RightMotorPin1, LOW);   // Reverse
    digitalWrite(RightMotorPin2, HIGH);
    analogWrite(LeftEnablePin, 20); //Stop suddenly
    analogWrite(RightEnablePin, 20); //Stop suddenly
    delay(50);

    // This particular turn is initiated when the right turn sensors
    // read high. But the front sensors are on at the start of a turn
    // and go off in the middle of a turn and then go on again at the
    // end of the turn. To have consistent condition for the while
    // statement that executes the turn, the "InTurn" variable is used.
    // This is set to 1 as soon as the front sensors go on. This coupled
    // with the rest of the condition in the while statement works well.
    InTurn = 0;

    // This variable is used to vary the speed in the middle of the
    // turn.
    speedReduction = 0;

    // The while condition mentioned above.
    while (InTurn == 0 ||
    (LC1sensor < LC1Threshold &&
    RC1sensor < RC1Threshold)) {
        // This variable increments up to thirty, which means the
        // original speed reduces by up to thirty.
        speedReduction++;
        if (speedReduction > 30){speedReduction = 30;}
        servoMain.write(closes);

        // If the laser trip wire is closed, stop immediately by
        // reversing the motors for a very short time (50 ms)
        senseval = digitalRead(sensorpin); //read in photoresistor value
        if (senseval == 0) {
            digitalWrite(LeftMotorPin1, LOW);   // Reverse
            digitalWrite(LeftMotorPin2, HIGH);
            digitalWrite(RightMotorPin1, LOW);   // Reverse
            digitalWrite(RightMotorPin2, HIGH);
            analogWrite(LeftEnablePin, 20); //Stop suddenly
            analogWrite(RightEnablePin, 20); //Stop suddenly
            delay(50);
            liftCan();
            recover();
        }

        // Turn the robot right by spinning wheels in opp. directions
        digitalWrite(LeftMotorPin1, HIGH);   // Forward
        digitalWrite(LeftMotorPin2, LOW);
```

```
        digitalWrite(RightMotorPin1, LOW);   // Reverse
        digitalWrite(RightMotorPin2, HIGH);
        analogWrite(LeftEnablePin, turningspeed); //Slow
        analogWrite(RightEnablePin, turningspeed); //Slow

        // Read in sensors for the next iteration of the while loop
        LC2sensor = filteredRead(LeftControlPin2, LC2Threshold);
        LC1sensor = filteredRead(LeftControlPin1, LC1Threshold);
        RC2sensor = filteredRead(RightControlPin2, RC2Threshold);
        RC1sensor = filteredRead(RightControlPin1, RC1Threshold);

        d = 0; // every time the turn occurs, this is reset
        // this is to figure out if the switching between
        // turning and line following is complete.

        // When the front sensors all go off, set the InTurn variable to
        // zero.
        if (LC1sensor < LC1Threshold &&
            LC2sensor < LC2Threshold &&
            RC1sensor < RC1Threshold &&
            RC2sensor < RC2Threshold) {
            InTurn = 1;
        }
    }

    // check if this function has been called after consistent line
    // following by checking if the value of TT has been set to 0. If
    // so, increment the number of turns by 1.
    if (TT == 0){
        count = count + 1;
        TT = 1;
        TurnsDone = count;
    }
}

// TAKE THE FOURTH TURN (TURNING LEFT)
void turnFirstLeft(){
    // Stop suddenly by spinning the wheels in reverse for a very
    // short time (50 ms)
    digitalWrite(LeftMotorPin1, LOW);   // Reverse
    digitalWrite(LeftMotorPin2, HIGH);
    digitalWrite(RightMotorPin1, LOW);   // Reverse
    digitalWrite(RightMotorPin2, HIGH);
    analogWrite(LeftEnablePin, 20); //Stop suddenly
    analogWrite(RightEnablePin, 20); //Stop suddenly
    delay(50);

    // This function is called when all front sensors go low,
    // which happens when a corner is found.
    // Continue turning right until the two center sensors read
    // high again.
    while (LC1sensor < LC1Threshold &&
           RC1sensor < RC1Threshold) {

        // Make sure the gripper is closed when the robot is
```

```
        // following the line.
        servoMain.write(closes);

        // If the laser trip wire is closed, stop immediately by
        // reversing the motors for a very short time (50 ms)
        senseval = digitalRead(sensorpin); //read in photoresistor value
        if (senseval == 0) {
            digitalWrite(LeftMotorPin1, LOW);   // Reverse
            digitalWrite(LeftMotorPin2, HIGH);
            digitalWrite(RightMotorPin1, LOW);   // Reverse
            digitalWrite(RightMotorPin2, HIGH);
            analogWrite(LeftEnablePin, 20); //Stop suddenly
            analogWrite(RightEnablePin, 20); //Stop suddenly
            delay(50);
            liftCan();
            recover();
        }

        // Turn the robot right by spinning wheels in opp. directions
        digitalWrite(LeftMotorPin1, LOW);   // Reverse
        digitalWrite(LeftMotorPin2, HIGH);
        digitalWrite(RightMotorPin1, HIGH);   // Forward
        digitalWrite(RightMotorPin2, LOW);
        analogWrite(LeftEnablePin, turningspeed-20); //Slow
        analogWrite(RightEnablePin, turningspeed-20); //Slow

        // Read in sensors for the next iteration of the while loop
        LC2sensor = filteredRead(LeftControlPin2, LC2Threshold);
        LC1sensor = filteredRead(LeftControlPin1, LC1Threshold);
        RC2sensor = filteredRead(RightControlPin2, RC2Threshold);
        RC1sensor = filteredRead(RightControlPin1, RC1Threshold);

        d = 0; // every time the turn occurs, this is reset
        // this is to figure out if the switching between
        // turning and line following is complete.
    }

    // check if this function has been called after consistent line
    // following by checking if the value of TT has been set to 0. If
    // so, increment the number of turns by 1.
    if (TT == 0){
        count = count + 1;
        TT = 1;
        TurnsDone = count;
    }
}

void turnSecondLeft(){
    // Stop suddenly by spinning the wheels in reverse for a very
    // short time (50 ms)
    digitalWrite(LeftMotorPin1, LOW);   // Reverse
    digitalWrite(LeftMotorPin2, HIGH);
    digitalWrite(RightMotorPin1, LOW);   // Reverse
    digitalWrite(RightMotorPin2, HIGH);
    analogWrite(LeftEnablePin, 20); //Stop suddenly
```

```
    analogWrite(RightEnablePin, 20); //Stop suddenly
    delay(50);

    //Turn LEFT
    while (LC1sensor < LC1Threshold &&
           RC1sensor < RC1Threshold) {
        servoMain.write(closes);

        // Turn the robot left by spinning wheels in opp. directions
        digitalWrite(LeftMotorPin1, LOW);   // Reverse
        digitalWrite(LeftMotorPin2, HIGH);
        digitalWrite(RightMotorPin1, HIGH);   // Forward
        digitalWrite(RightMotorPin2, LOW);
        analogWrite(LeftEnablePin, turningspeed); //Slow
        analogWrite(RightEnablePin, turningspeed); //Slow

        // Read in sensors for the next iteration of the while loop
        LC2sensor = filteredRead(LeftControlPin2, LC2Threshold);
        LC1sensor = filteredRead(LeftControlPin1, LC1Threshold);
        RC2sensor = filteredRead(RightControlPin2, RC2Threshold);
        RC1sensor = filteredRead(RightControlPin1, RC1Threshold);

        d = 0; // every time the turn occurs, this is reset
        // this is to figure out if the switching between
        // turning and line following is complete.
    }

    // check if this function has been called after consistent line
    // following by checking if the value of TT has been set to 0. If
    // so, increment the number of turns by 1.
    if (TT == 0){
        count = count + 1;
        TT = 1;
        TurnsDone = count;
    }
}
void liftCan() {
    digitalWrite(LeftMotorPin1, LOW);   // Stop
    digitalWrite(LeftMotorPin2, LOW);
    digitalWrite(RightMotorPin1, LOW);  // Stop
    digitalWrite(RightMotorPin2, LOW);
    analogWrite(LeftEnablePin, 0); //Stop
    analogWrite(RightEnablePin, 0); //Stop

    i2c_start_wait(dev+I2C_WRITE);
    i2c_write(0x07);

    // read temperature sensor
    i2c_rep_start(dev+I2C_READ);
    data_low = i2c_readAck(); //Read 1 byte and then send ack
    data_high = i2c_readAck(); //Read 1 byte and then send ack
    pec = i2c_readNak();
    i2c_stop();
```

```
// a little bit of the servo code here to give the temperature
    // sensor time to register.
    delay(1000);
    servoMain.write(opens);
    delay(300);
    servoWinch.write(low);
    delay(1300);
    //This converts high and low bytes together and processes
    //temperature, MSB is a error bit and is ignored for temps double

    double tempFactor = 0.02;
    // 0.02 degrees per LSB (measurement resolution
    //of the MLX90614)
    double tempData = 0x0000; // zero out the data
    int frac; // data past the decimal point

    // This masks off the error bit of the high byte, then moves it left
    // 8 bits and adds the low byte.
    tempData = (double)(((data_high & 0x007F) << 8) + data_low);
    tempData = (tempData * tempFactor)-0.01;
    float celsius = tempData - 273.15;
    float fahrenheit = (celsius*1.8) + 32;

    if (fahrenheit < 50 && JJ == 0)  {
        can_temp = cold;
        JJ == 1; // make sure after one can is set as cold, no other
                 // cans are considered for its location
    }
    else {
        can_temp = warm;
    }

    //Define can parameters
    if (can_number==1) // first hot can
    {
        placement_angle = 0;
        travel_time = 2000;
        travel_time_empty = 750;
    }
    if (can_number==2) // second hot can
    {
        placement_angle = 201;
        travel_time = 1250;
        travel_time_empty = 625;
    }
    if (can_number==3) // third hot can
    {
        placement_angle = 402;
        travel_time = 1000;
        travel_time_empty = 500;
    }
```

```
if (can_number==4) // fourth hot can
    {
        placement_angle = 603;
        travel_time = 750;
        travel_time_empty = 300;
    }
    if (can_temp==cold) // cold can
    {
        placement_angle = 1023;
        travel_time = 500;
        travel_time_empty = 250;
        cold_can_done = 1;
    }

    servoMain.write(closes);   // Close gripper jaws
    delay(300);
    servoWinch.write(highwithcan); // boom raises to high position with
                                   // can
    delay(1200);
    Dynamixel.moveSpeed(1,placement_angle,200); // boom serbo rotates to
                                                // place the can
    delay(travel_time + 200);
    servoWinch.write(mid); // boom lowers set can on robot
    delay(1700);
    servoMain.write(opens); // Open gripper jaws
    delay(300);
    // go back to the home position
    servoWinch.write(high);
    delay(600);
    Dynamixel.moveSpeed(1,pickup_angle,600); // empty boom moves over to
                                             // pickup point
    delay(600);
    if (can_temp == warm) {
        can_number = can_number + 1;
    }
    if (can_number == 4 && cold_can_done == 1) {
          all_cans_done = 1;
          can_number = 5;
    }
}

void dumpCans(){
    digitalWrite(LeftMotorPin1, LOW);   // Reverse
    digitalWrite(LeftMotorPin2, HIGH);
    digitalWrite(RightMotorPin1, LOW);   // Reverse
    digitalWrite(RightMotorPin2, HIGH);
    analogWrite(LeftEnablePin, 20); //Stop suddenly
    analogWrite(RightEnablePin, 20); //Stop suddenly
    delay(50);
    Rsensor = filteredRead(RightSensorPin, RThreshold);
    while (Rsensor < RThreshold){
        digitalWrite(LeftMotorPin1, LOW);   // Reverse
        digitalWrite(LeftMotorPin2, HIGH);
        digitalWrite(RightMotorPin1, HIGH);   // Forward
        digitalWrite(RightMotorPin2, LOW);
```

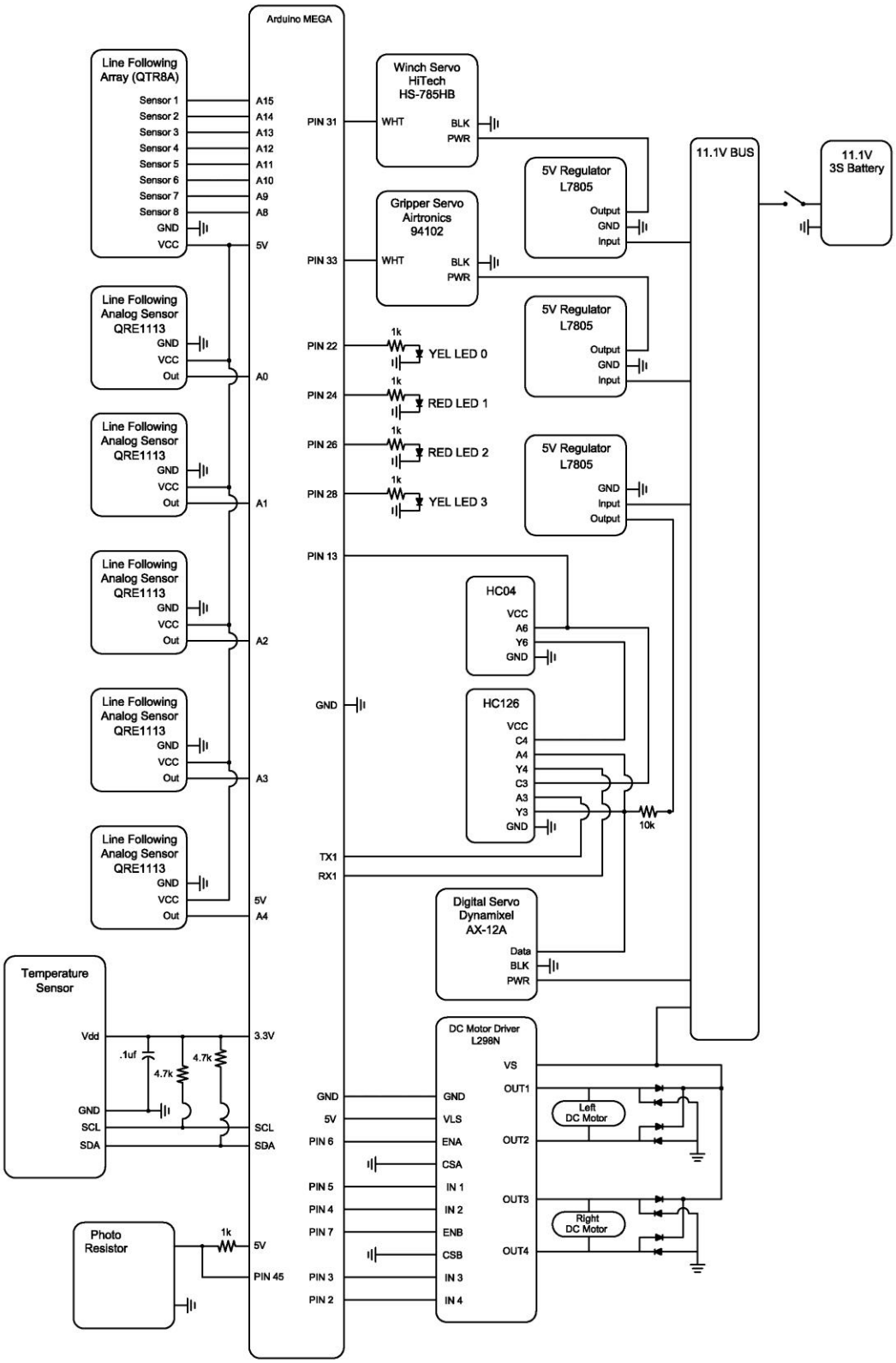
```
    analogWrite(LeftEnablePin, 50);
    analogWrite(RightEnablePin, 50);
    Rsensor = filteredRead(RightSensorPin, RThreshold);
}
digitalWrite(LeftMotorPin1, LOW);   // Reverse
digitalWrite(LeftMotorPin2, HIGH);
digitalWrite(RightMotorPin1, LOW);   // Reverse
digitalWrite(RightMotorPin2, HIGH);
analogWrite(LeftEnablePin, 50); //Stop suddenly
analogWrite(RightEnablePin, 50); //Stop suddenly
delay(50);
digitalWrite(LeftMotorPin1, LOW);   // Reverse
digitalWrite(LeftMotorPin2, LOW);
digitalWrite(RightMotorPin1, LOW);   // Reverse
digitalWrite(RightMotorPin2, LOW);
analogWrite(LeftEnablePin, 0); //Stop suddenly
analogWrite(RightEnablePin, 0); //Stop suddenly
delay(1000);
//Define can parameters
cans_left = can_number;
placement_angle = 818;
while (cans_left > 0){
    if (cans_left==5) // fourth hot can
    {
      pickup_angle = 603;
      travel_time = 750;
      travel_time_empty = 300;
    }
    if (cans_left==4) // third hot can
    {
      pickup_angle = 402;
      travel_time = 1000;
      travel_time_empty = 500;
    }
    if (cans_left==3) // second hot can
    {
      pickup_angle = 201;
      travel_time = 1250;
      travel_time_empty = 625;
    }
    if (cans_left==2) // first hot can
    {
      pickup_angle = 0;
      travel_time = 2000;
      travel_time_empty = 750;
    }
    if (cans_left==1 && cold_can_done==1) // cold can goes last
    {
      pickup_angle = 1023;
      travel_time = 500;
      travel_time_empty = 250;
      //cold_can_done = 1;
    }
    cans_left = cans_left - 1;
    servoWinch.write(highwithcan);
```

```
        delay(800);
        Dynamixel.moveSpeed(1,pickup_angle,400); // empty boom moves
                                                 // over to pickup point
        servoMain.write(opens);
        delay(1000);
        servoWinch.write(mid);
        delay(1300);
        servoMain.write(closes);   // Turn Servo Left to 0 degrees
        delay(300);
        servoWinch.write(highwithcan); // boom raises to high position
                                       // with can
        delay(1200);
        Dynamixel.moveSpeed(1,placement_angle,200); // moves to place
                                                    // the can
        delay(travel_time + 500);
        servoWinch.write(low); //boom lowers set can on robot
        delay(1700);
        servoMain.write(opens); // hand is opened
        delay(300);
        // go back to the home position
        servoWinch.write(highwithcan);
        delay(600);
    }
}

// FUNCTION FOR DEBOUNCING LINE SENSOR READINGS. THIS FUNCTION CHECKS IF
// THE READING STAYS OVER THE SPECIFIED THRESHOLD VALUE OVER A PERIOD OF
// TIME (milliseconds), AND ONLY THEN PASSES THE STABILIZED READING
// BACK.
int filteredRead(int pinID, int pinThresh){
    stable = 0; // start assuming reading is not stable
    lastReading = 2;  // set a random value for lastReading to begin
    while (stable == 0){
        actualReading = analogRead(pinID);  // read in pin value
        if (actualReading > pinThresh){
            reading = 1;}   // if read value is higher than specified
            // threshold, set the variable to 1
        else {
            reading = 0;}   // If the value is lower, set it to 0
        if (reading != lastReading){
            lastDebounceTime = millis(); // if the set reading is
            // consistently 1, it means that the sensor is consistently
            // reading above the threshold. At this point return the
            // actual sensor reading.
        }
        if ((millis() - lastDebounceTime) > debounceDelay) {
            stable = 1; // set the reading as stable
        }
        return actualReading;
    }
    }

// FUNCTION FOR RE-CENTERING THE ROBOT OVER THE LINE FOLLOWERS
// (USUALLY AT THE END OF PICKING UP A CAN)
// IF BOTH FRONT SENSORS ARE OFF THE LINE, THE FUNCTION FINDS OUT WHICH
```

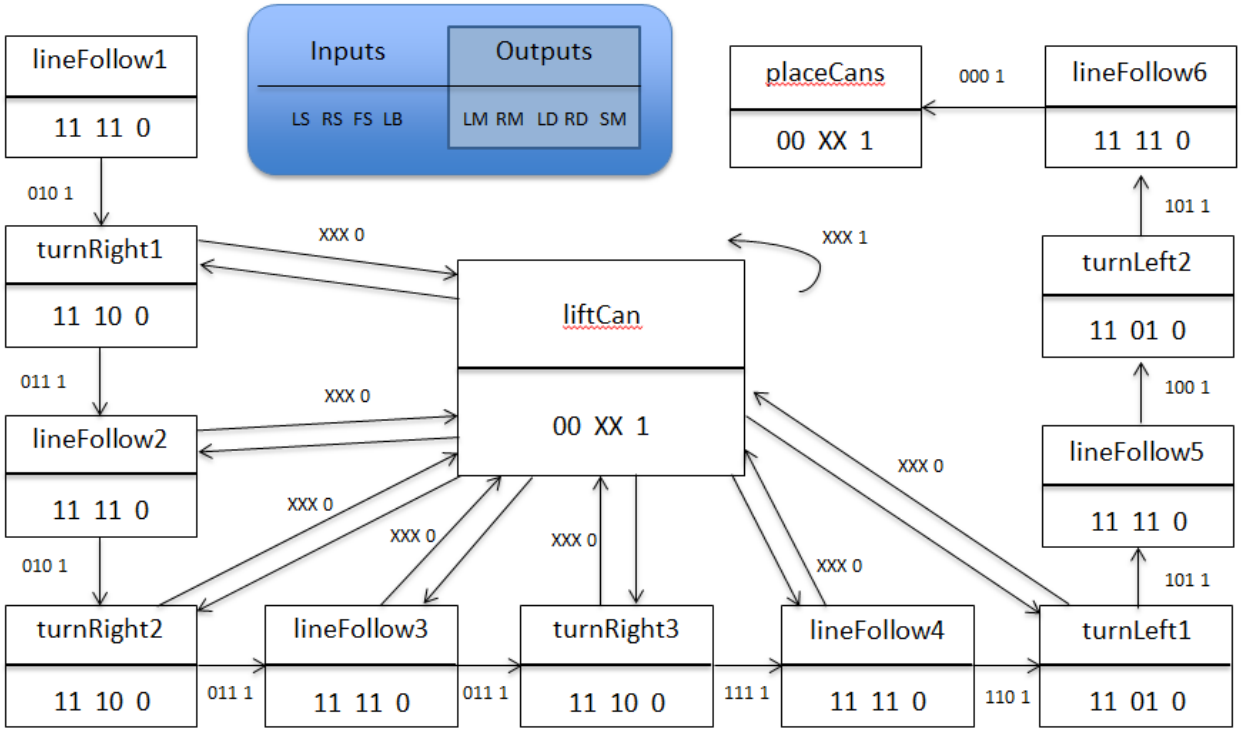
```
// SENSOR IS OVER THE LINE, AND APPLIES MOTOR CORRECTIONS APPROPRIATELY.
void recover(){
    while (!(LC1sensor > LC1Threshold && RC1sensor > RC1Threshold)){
        LC5sensor = filteredRead(LeftControlPin5, LC5Threshold);
        LC4sensor = filteredRead(LeftControlPin4, LC4Threshold);
        LC3sensor = filteredRead(LeftControlPin3, LC3Threshold);
        LC2sensor = filteredRead(LeftControlPin2, LC2Threshold);
        LC1sensor = filteredRead(LeftControlPin1, LC1Threshold);
        RC1sensor = filteredRead(RightControlPin1, RC1Threshold);
        RC2sensor = filteredRead(RightControlPin2, RC2Threshold);
        RC3sensor = filteredRead(RightControlPin3, RC3Threshold);
        RC4sensor = filteredRead(RightControlPin4, RC4Threshold);
        RC5sensor = filteredRead(RightControlPin5, RC5Threshold);
        if ((RC2sensor > RC2Threshold)) {
            digitalWrite(LeftMotorPin1, HIGH);   // Forward
            digitalWrite(LeftMotorPin2, LOW);
            digitalWrite(RightMotorPin1, LOW);  // Reverse
            digitalWrite(RightMotorPin2, HIGH);
            analogWrite(LeftEnablePin, 50); //Fast
            analogWrite(RightEnablePin, 50); //Fast
        }
        if ((LC2sensor > LC2Threshold)) {
            digitalWrite(LeftMotorPin1, LOW);   // Reverse
            digitalWrite(LeftMotorPin2, HIGH);
            digitalWrite(RightMotorPin1, HIGH);  // Forward
            digitalWrite(RightMotorPin2, LOW);
            analogWrite(LeftEnablePin, 50); //Slow
            analogWrite(RightEnablePin, 50); //Slow
        }
        if ((RC3sensor > RC3Threshold)) {
            digitalWrite(LeftMotorPin1, HIGH);   // Forward
            digitalWrite(LeftMotorPin2, LOW);
            digitalWrite(RightMotorPin1, LOW);  // Reverse
            digitalWrite(RightMotorPin2, HIGH);
            analogWrite(LeftEnablePin, 50); //Fast
            analogWrite(RightEnablePin, 50); //Fast
        }
        if ((LC3sensor > LC3Threshold)) {
            digitalWrite(LeftMotorPin1, LOW);   // Reverse
            digitalWrite(LeftMotorPin2, HIGH);
            digitalWrite(RightMotorPin1, HIGH);  // Forward
            digitalWrite(RightMotorPin2, LOW);
            analogWrite(LeftEnablePin, 50); //Slow
            analogWrite(RightEnablePin, 50); //Slow
        }
        if ((RC4sensor > RC4Threshold)) {
            digitalWrite(LeftMotorPin1, HIGH);   // Forward
            digitalWrite(LeftMotorPin2, LOW);
            digitalWrite(RightMotorPin1, LOW);  // Reverse
            digitalWrite(RightMotorPin2, HIGH);
            analogWrite(LeftEnablePin, 50); //Fast
            analogWrite(RightEnablePin, 50); //Fast
        }
```

```
if ((LC4sensor > LC4Threshold)) {
            digitalWrite(LeftMotorPin1, LOW);   // Reverse
            digitalWrite(LeftMotorPin2, HIGH);
            digitalWrite(RightMotorPin1, HIGH);  // Forward
            digitalWrite(RightMotorPin2, LOW);
            analogWrite(LeftEnablePin, 50); //Slow
            analogWrite(RightEnablePin, 50); //Slow
        }
        if ((RC5sensor > RC5Threshold)) {
            digitalWrite(LeftMotorPin1, HIGH);   // Forward
            digitalWrite(LeftMotorPin2, LOW);
            digitalWrite(RightMotorPin1, LOW);  // Reverse
            digitalWrite(RightMotorPin2, HIGH);
            analogWrite(LeftEnablePin, 50); //Fast
            analogWrite(RightEnablePin, 50); //Fast
        }
        if ((LC5sensor > LC5Threshold)) {
            digitalWrite(LeftMotorPin1, LOW);   // Reverse
            digitalWrite(LeftMotorPin2, HIGH);
            digitalWrite(RightMotorPin1, HIGH);  // Forward
            digitalWrite(RightMotorPin2, LOW);
            analogWrite(LeftEnablePin, 50); //Slow
            analogWrite(RightEnablePin, 50); //Slow
        }
    }
}
```

Appendix 2 – Circuit diagrams



Appendix 3 – State diagrams



Appendix 4 – Expenditure

Description	Unit Price	QTY	Total
Makeblock Configurable 2WD Robot Kit – Blue	69.95	1	$69.95
Droids 1000Pads Mini Basic Breadboard	3.45	3	$10.35
Infrared Thermometer - MLX90614	19.95	1	$19.95
QRE1113 Line Sensor Breakout - Analog	2.95	7	$20.65
Laser	7.95	1	$7.95
Photo resistor	1.50	1	$1.50
Cage Assembly for 1" Shaft Diameter, 1-9/16" OD Steel Thrust Needle-Roller Bearing	3.09	2	$6.18
Aluminum	19.34	1	$19.34
.032" Thick Washer for 1" Shaft Diameter Steel Thrust Needle-Roller Bearing	1.15	4	$4.60
Arduino MEGA 2560	50.00	1	$50.00
Airtronics 94102 Servo	12.99	1	$12.99
Gripper Servo	10.00	1	$10.00
Winch Servo	49.99	1	$49.99
Dynamixel Servo	44.99	1	$44.99
QTR8A-8 line sensor array	14.95	1	$14.95
L298N	4.50	1	$4.50
Switches	0.50	2	$1.00
Plastic Casters	2.50	2	$5.00
Lipos Batteries	7.00	1	$7.00
HC04	0.40	1	$0.40
HC126	0.80	1	$0.80
L7805--5V Linear Voltage Regulator	0.95	2	$1.90
Resistors, Capacitors, and Diodes	5.00	1	$5.00
Wires, Connectors, Pins	20.00	1	$20.00
Machine Shop Parts	10.00	1	$10.00
Rubber Pad	1.00	1	$1.00
Total			**$399.99**