ME 588 Mechatronics Spring 2024

**Final Project Report**

**Autonomous Robotic Pollinator**

**Team Po11eneers**

A black and yellow device with wires

Description automatically generated

|  |  |
| --- | --- |
| Team Members: | |
| Idan Adar | Noah Davis |
| Ethan DeVries | Daniel Kim |
| Oliver Shindell | Nikhil Tiwari |

April 29, 2024

# Autonomous Robotic Pollinators

## Abstract:

## This report presents the conception, fabrication, and deployment of an Autonomous Robotic Pollinator, developed within the framework of the ME588 Mechatronics Final Project guidelines. The driving mechanism of the autonomous device relies on encoders and ultrasonic sensors, while the sorting and delivery system is propelled by a gravity-driven mechanism controlled by servos and sensors. The initial sections of the report delve into the team's ideation process and decision-making regarding various sorting mechanisms and motion planning sensors. Subsequently, the design of the sorting and delivery mechanism is elaborated upon, alongside the exploration of different strategies for navigation and delivery. Finally, the report delves into the outcomes and analysis of the project. Despite encountering challenges in consistent field navigation, the sorting mechanism achieved notable success, efficiently sorting balls with a high degree of accuracy.

## Introduction:

## The project aimed to develop an autonomous robotic system tasked with color sorting balls and transporting them to designated locations. The robot was expected to operate independently throughout a 2-minute match duration, excluding time allocated for loading pollen at the home base. All mechanisms and attachments of the robot were required to fit within a confined space measuring 12 inches in each dimension. Specifically, the robot was designed to sort and deliver balls with a diameter of 1.65 inches within the playing field, as outlined in Figure 1 and Figure 2.

A diagram of a farm

Description automatically generated A diagram of a delivery zone

Description automatically generated

Figure 1: Dimensions of Playing Field Figure 2: Dimensions of Plants

The robot utilizes gravity to facilitate the transportation of balls through its body. Through the integration of five servos, the robot can redirect, or block balls as needed to ensure they reach their designated locations. The servos are controlled based on inputs from two sensors: a color sensor and an ultrasonic sensor. The color sensor identifies the color of each ball and adjusts the sorting flap to direct it to the appropriate lane. The lane assignments are determined at the start of the game via inputs from push buttons and an LCD display. Meanwhile, the ultrasonic sensor detects the position of the plant and activates the gate servo to open, allowing the balls to fall into place. For propulsion, the robot is equipped with two 12-volt motors mounted on the front wheels, complemented by two caster wheels at the rear. Encoders are employed on the motors to ensure straight driving and accurate turning, while an ultrasonic sensor detects obstacles at the front and sides. The DC motors driving the wheels are controlled using L298 H-bridges, with power supplied by an 11.1V 2200mAh battery. The entire system is governed by an Arduino Mega 2560, which orchestrates the robot's design, operational strategy, and control. The calibration of sensors and actuators is meticulously detailed in the subsequent sections.

A grey robot with wheels

Description automatically generatedA grey machine with wheels

Description automatically generated

Figure 3: The Robot

## Methodology:

### Sorting Mechanism:

A grey and green object with a black handle

Description automatically generated with medium confidenceA grey metal object with black and white details

Description automatically generated with medium confidence

Figure 4: Sorting Mechanism Section and Top view

The sorting mechanism depicted in Figure 4 operated through the synchronized action of two MG160 micro-servos, namely the Gate servo and Sorter servo, in conjunction with the TCS3200 color sensor. Its purpose was to segregate differently colored balls into three distinct lanes labeled Plant 1, Plant 2, and Plant 3, corresponding to the designated colors for each plant. Initially, the color assignment for each plant was determined via user inputs facilitated by push buttons, ensuring flexibility in assigning colors to plants dynamically. This organizational approach based on plants rather than colors streamlined the delivery process. The mechanism functioned on gravity, with the servos serving to redirect balls to their respective lanes efficiently.

To illustrate the sorting process, a ball would enter the sorting mechanism and be intercepted by the Gate servo. The Gate servo's closed position (0°) held the ball beneath the color sensor for accurate sensing and delayed its movement, allowing the Sorter servo to adjust its angle accordingly. The Sorter servo received specific angle inputs to guide the ball into the appropriate lane based on the color sensed by the color sensor, altering its flap direction accordingly. Initially planned to operate with angle inputs of 45° (Plant 1), 90° (Plant 2), and 135° (Plant 3), calibration and troubleshooting revealed optimal angles of 35°, 85°, and 135° due to servo library discrepancies. Upon completion of the Sorter servo's motion, the Gate servo opened to 90°, enabling the ball to roll into the designated lane. Subsequently, the system reset, allowing for the sorting of a new ball. This process continued until a lane reached its capacity, determined by a predefined ball count for each color.

### Driving Mechanism:

A grey metal object with wheels

Description automatically generated

Figure 5: Driving Mechanism Bottom View

The chassis illustrated in Figure 5 demonstrates that movement is provided by two encoder motors located in the front, complemented by two caster wheels affixed at the rear for stability and aiding in turning. With the two driven motors, the robot possesses the capability to maneuver in any direction; however, in practice, it primarily executes driving straight, stopping, and turning ninety degrees to the left. The motors are equipped with two ultrasonic sensors, which provide feedback enabling the robot to determine when to halt for the plants and when to execute a left turn. The side ultrasonic sensor prompts the robot to stop when there is a change in the wall, indicating the presence of a plant. For this detection to trigger, the distance must be greater than 30 cm but less than 90 cm for a specified duration. The upper distance limit and time requirement serve to filter out noise and gaps in the walls. Meanwhile, the front ultrasonic sensor aids in identifying when the robot is 39 cm from the front wall. This distance allows the robot to turn with adequate space, ensuring it is not too close to the wall to hinder ball delivery to the next plant.

The motors employed in this robot are geared DC motors with encoders, controlled using an L-298 motor controller. This controller facilitates stepping a PWM signal to a higher voltage, direction reversal, and braking. The wiring configuration for the motors and motor controller is depicted in the circuit diagram. Utilizing the encoders, which are connected to interrupt pins of the Arduino, the motor speed is determined by counting the number of encoder pulses within a known time frame. Employing Euler approximations, values for error, integral error, and derivative error are calculated. These values are utilized to create a digitized PID controller, which regulates the speed of each motor to maintain it at the desired value. Similarly, for turning maneuvers, position control is applied with a comparable structure, ensuring consistent execution of ninety-degree turns.

### Delivery Mechanism:

The delivery mechanism illustrated in Figure 6 aimed for simplicity, employing three servos as gates positioned at the end of each lane to retain pre-sorted balls. During robot movement, these delivery servos remained closed with a 180° input, while an L-shaped flange ensured ball retention in the storage lanes until the robot halted. The storage was designed to accommodate eight balls with a 10-degree incline, facilitating ball release through gravity once the servos for the designated plant opened. The robot identified plant locations via side ultrasonic sensor readings, stopping when the distance to the wall exceeded 25cm. Plant identification was further aided by a plant count to open the correct servo for ball release. To dispense the balls, the delivery servos were instructed to open to 90°, allowing the stored balls to descend under gravity.

However, testing revealed a persistent issue: when more than four balls were stored, jams occurred in the curved section of the storage. Several potential causes were identified, including residual support material from printing, particularly challenging to remove in inaccessible areas. Despite efforts to alleviate the issue through sanding, jams persisted due to the curvature's inaccessibility. Another contributing factor was the 2-inch hole size, providing excess room for 1.65-inch balls, allowing them to stack and jam. To address this challenge, motor coding introduced a jerk maneuver as the robot left each plant station, facilitating the release of stuck balls just in time. Additionally, the delay for delivery servo closure was extended to accommodate this maneuver.

A grey machine with multiple cylinders

Description automatically generated with medium confidenceA grey machine with a curved tube

Description automatically generated with medium confidence

Figure 6: Delivery and Storage ISO and Section View

### Inputs and Outputs:

The Autonomous Robotic Pollinator interacts with its environment through a combination of sensory inputs and controlled outputs. The core input for the sorting mechanism comes from the color sensor, which identifies the color of each ball and transmits this information to the Arduino Mega. This data is vital for directing the balls along their designated paths. Additionally, ultrasonic sensors play a crucial role in navigation. Sensors positioned at the front and side of the robot continuously detect the distance from the robot to the wall, enabling the control system to make informed decisions about the movement and prevent collisions. The ultrasonic sensor on the side of the robot specifically targets the plant locations. When the robot reaches its designated destination, this sensor triggers the robot to stop its movement and the gate servo to release the sorted balls into the appropriate delivery zone. Finally, push buttons provide user input at the beginning of the game. By pressing the first two black buttons, the operator assigns specific colors to designated lanes, ensuring the robot sorts and delivers the balls efficiently. The third green button is used to confirm the assigned colors and move onto the sorting stage.

The robot translates these information and sensory data into a series of controlled outputs. Two 12V DC motors, governed by the Arduino Mega through L298 H-bridges, determine the robot’s movement throughout the playing field. The control system also manages five servos. The gate servo opens and closes, regulating the flow of balls into the sorter and its subsequent release. The sorter servo, equipped with a flap, adjusts its position based on the color sensor input. Depending on the ball’s color, the servo will tilt the flap to a specific angle (35°, 85°, or 135°), directing the ball down the designated lane. Finally, the LCD display shows the current state of the robot, presenting lane assignments based on user’s push button selections and ball counts during the sorting stage. It also displays whether the robot is sorting or driving, and whether it has reached a plant and is delivering the colored balls. This interplay between sensory inputs and controlled outputs allows the Autonomous Robotic Pollinator to sort colored balls, navigate its environment, and deliver colored balls to designated locations.

### Sensors:

For the robot's sensing capabilities, two types of sensors were employed: ultrasonic sensors for distance measurements and a color sensor for sorting input pollen balls. The wiring diagrams for both sensor types are detailed in Appendix A.

Two ultrasonic sensors were utilized, with one serving to determine the distance between the robot and the wall ahead, while the other was positioned on the side of the robot to detect when the robot reached a plant (noted by a sudden increase in the side ultrasonic reading). Both ultrasonic sensors were powered by 5 volts and grounded. A PWM signal was sent to their trigger pins, and the Arduino Mega received the output signal from the ultrasonic echo pins. The input signal to the ultrasonic sensors was set to high for ten microseconds and then low for two microseconds. The pulseIn() function in Arduino was employed to measure the duration of the echo signal's high state, allowing for distance calculation using a calibration factor (approximately 0.034/2 to convert to centimeters). Additionally, the ultrasonic sensors were equipped with built-in high-pass filters in their circuitry. To further filter the input signal and address signal fluctuations occurring between gaps in the field wall, a combination of median filtering and setting an upper bound distance threshold was employed.

The color sensor used was a TCS3200, which analyzes color using red, blue, and green filtered photodiodes and built-in LEDs. The sensor was powered by 5 volts from the battery, while the LEDs were powered by the same battery through resistors to reduce the voltage to 1.2 volts. This voltage reduction was necessary to prevent excessive light emission from the LEDs directly into the sensor. Achieved using a 3.3 kOhm resistor and 1 kOhm resistor, the setup emulated the layout of a potentiometer. The S1 and S0 pins on the sensor were utilized to select the output frequency scaling, set to 100%. The code activated each filtered photodiode by configuring combinations of the S2 and S3 pins on the sensor to high or low. The OUT pin delivered an output frequency proportional to the current flowing through the photodiodes, based on the color filter used and the amount of detected color. This frequency was translated into an integer value, with a larger value indicating less of the color detected. Due to the potential impact of ambient light on the sensor's readings, consistent readings required the balls to be positioned close to the LEDs. Additionally, calibration of the sensor's readings for each ball was necessary. Once calibrated, each ball was assigned a range for each set of color-filtered photodiodes, allowing the code to identify it accurately.

### Power Distribution:

The robot was powered by a 2200mAh 11.1V 3-cell LiPo battery, chosen for its ample capacity, ensuring prolonged operation without the need for frequent recharging during testing phases. Power distribution was facilitated through a pegboard conveniently housed within the angled wedge of the robot, providing accessible connection points. The pegboard featured screw-down terminals, offering secure connections without necessitating soldering for each wire. The battery was connected to the pegboard, supplying power to a 12V power rail, which in turn powered essential 12-volt components such as the motor controller, Arduino Mega, and a 12V to 5V Buck Converter. The output from the Buck Converter powered a 5V voltage rail, supplying power to the remaining devices including sensors, push buttons, LCD, and servos. To ensure stable voltage and eliminate the possibility of ground loops, each device received power and ground connections directly from the pegboard. Additionally, power distribution was prioritized from the most power-hungry devices to the least, ensuring all components received sufficient power for optimal performance. The implementation of the pegboard served to segregate power distribution from the Arduino, thereby safeguarding against voltage surges that could potentially damage the Arduino. This segregation ensured that the Arduino's power capacity was exclusively reserved for signaling to the components, optimizing its functionality.

### Motion Planning:

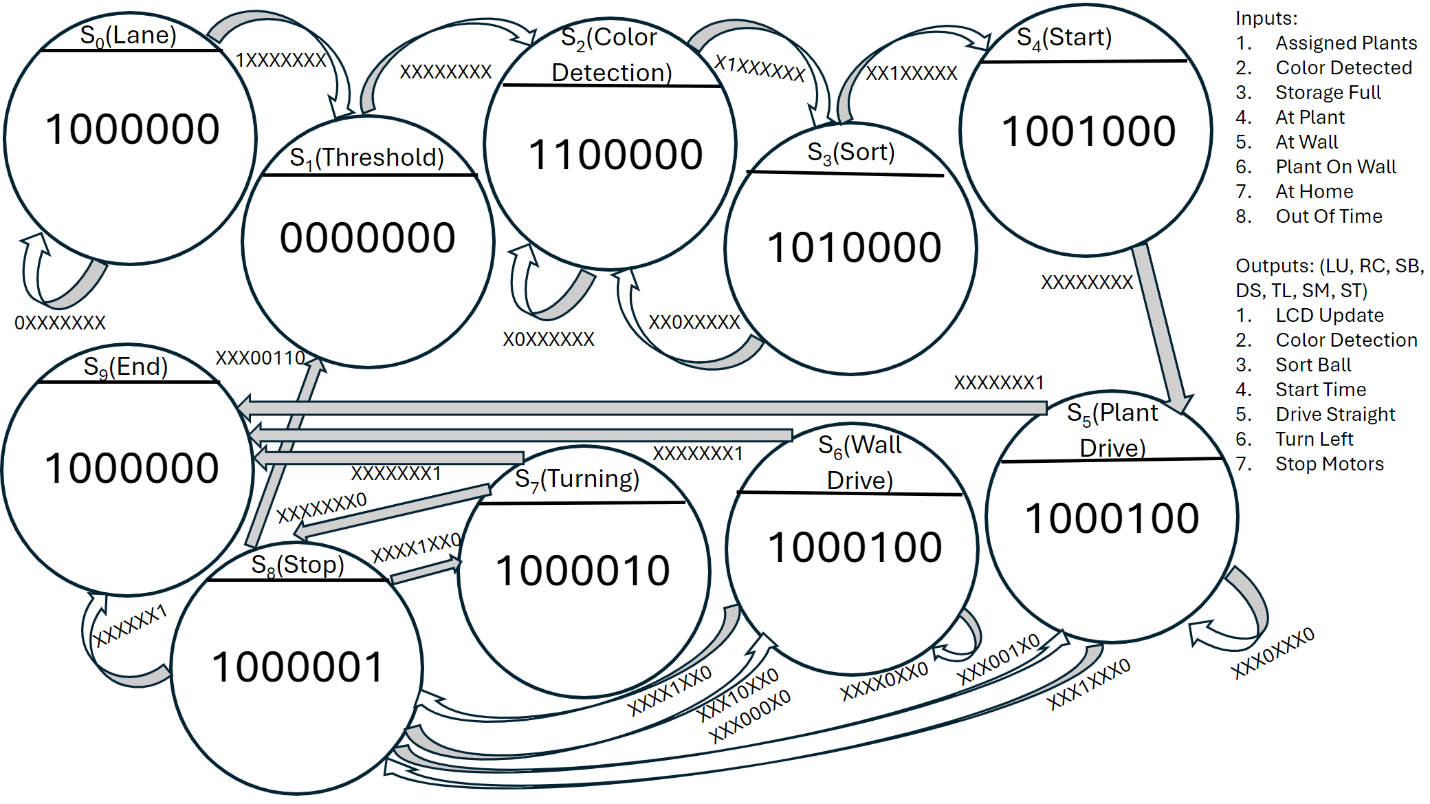


Figure 9: Overall FSM

The overall Finite State Machine (FSM) depicted in Figure 9 governs the autonomous operation of the robot. It initiates in the Lane state, where the user assigns plant colors by pressing push buttons. Upon assignment, pressing the start button activates the autonomous phase. The robot transitions to the Threshold state, facilitating the color sensor's reading when no ball is present, serving as a reference for subsequent ball detection. Upon completing its tasks, the robot exits the Threshold state without any input change.

Next, the robot enters the Color\_Detection state, which detects the presence of a ball in the input and determines its color using the color sensor. Subsequently, in the Sort state, the robot directs the ball to the corresponding plant lane based on its color, utilizing the Sort servo. The gate servo is then opened to allow the ball to roll down into the proper storage lane. After sorting, the gate closes, and the robot returns to the Color\_Detection state before reentering the Sort state. This cycle continues until the ball count for any color reaches the predefined maximum (four balls).

Upon indicating full capacity, the robot transitions to the Start state, initiating a timer to ensure a two-minute operation duration. From Start, it proceeds to the Plant\_Drive state, where it moves forward using encoder motors, guided by the side ultrasonic sensor to detect plant presence. The side ultrasonic sensor prompts the robot to stop when there is a change in the wall, indicating the presence of a plant. For this detection to trigger, the distance must be greater than 30 cm but less than 90 cm for a specified duration. The upper distance limit and time requirement serve to filter out noise and gaps in the walls. Upon reaching a plant, the robot enters the Stop state, opening the storage plant servo to release balls into the plant. The plant count ensures proper navigation among the plants. Following a delay, the robot enters the Wall\_Drive state to maintain straight motion. Meanwhile, the front ultrasonic sensor aids in identifying when the robot is 39 cm from the front wall. This distance allows the robot to turn with adequate space, ensuring it is not too close to the wall to hinder ball delivery to the next plant. When the front ultrasonic indicates that the walls are 39 cm away the robot once again enters the Stop state, then assesses that it is at the wall and enters the Turning state.

In the Turning state, the right motor executes a 90-degree turn to align with the next wall, before returning to Plant\_Drive, Stop, Wall\_Drive, Stop, and Turning. This cycle repeats until the plant count reaches four, signifying the robot is on a wall with no plant, so it is time to return home. To do so it interrupts the normal cycle and goes straight to Wall\_Drive to find the home wall. Once it finds the wall it returns to Stop and then Turning to get to its original position. Upon reaching the original position, the process restarts from the Threshold state, with the timer paused until leaving home again. The robot exits this loop upon the timer reaching the two-minute field play duration, transitioning to the End state. Throughout the operation, the LCD display provides essential information on plant and ball colors, current state, and home status.

## Results:

Numerous tests were conducted to evaluate the performance of the navigation system and pollen delivery mechanism. Results from these tests indicated inconsistencies in both driving behavior and pollen delivery efficiency. While the navigation system successfully detected plants upon reaching them, it relied solely on encoder measurements to maintain a straight trajectory. Consequently, deviations in track conditions or starting points often resulted in collisions with walls or deviations from the intended path. The sensing of plants was generally consistent, and the ability to make precise 90 degree turns worked well.

Although plant detection was generally reliable, challenges arose in the pollen delivery process. Balls frequently became lodged inside the storage area, hindering their proper release at designated plants. This issue was attributed to uneven surfaces within the storage area, preventing the robot from achieving maximum pollen capacity and consequently reducing delivery efficiency.

During initial testing, all sensing operations were performed within the main loop of the Arduino program. However, it was discovered that more consistent results could be achieved by utilizing interrupt pins and functions. For each motor, one of its encoder channels was moved to an interrupt pin, while the other was left as a basic digital read. A subfunction was set to trigger on the rising edge of the interrupt channel, during which the reading from the second channel was taken to determine the direction of the motor. Similarly, ultrasonic sensors were set up in a similar fashion, with echo pins assigned to interrupt channels. This approach enabled precise readings to be captured while the main code continued executing, contributing to improved overall performance and reliability.

## Discussion:

The primary limitation affecting driving performance was the absence of trajectory correction. While the robot occasionally completed multiple laps around the track, it frequently collided with walls or deviated from the intended path due to reliance on initial conditions and consistent track conditions, which were not always present. One potential solution, utilizing existing hardware, involved measuring the rate of change of the side ultrasonic reading and adjusting the robot's trajectory away from the wall if it approached too closely. However, due to higher priority concerns, this approach was not pursued.

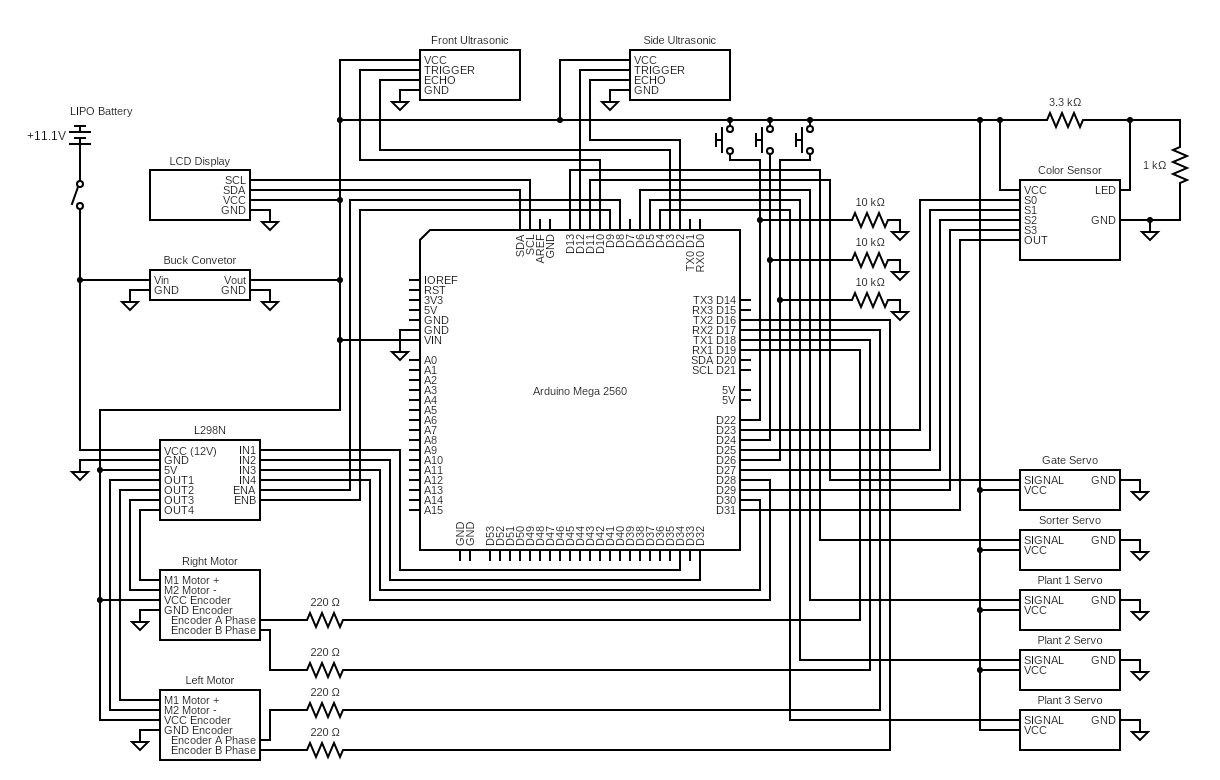
Another significant challenge involved anomalies and erroneous readings from the ultrasonic sensors, which could misinterpret wall geometry or register zero readings upon startup. Initially, a median filter was implemented to address this issue; however, this solution introduced delays in detecting actual plants due to the need for a large filter size to filter out unwanted readings. Instead, a dual-threshold approach was adopted, with minimum and maximum thresholds set for ultrasonic readings. For instance, the side ultrasonic sensor would detect a plant if the reading exceeded 35 cm but ignore readings over 90 cm. This adjustment allowed the robot to consistently identify both plants and walls.

To mitigate the problem of balls becoming lodged inside the robot, a jitter function was introduced. This function briefly set both motors to maximum speed before abruptly braking them, inducing a jittering motion that dislodged stuck balls. However, excessive jittering risked compromising the robot's position on the field, necessitating a balance between jitter intensity and positional stability. Despite these challenges, the jitter function notably improved the consistency of pollen delivery.

## Conclusion:

In conclusion, while the robot showcased in this report demonstrated well-designed systems and commendable functionality, its performance was hindered by a lack of sufficient sensing data to consistently execute its required tasks. Currently, the robot heavily relies on a consistent field and precise setup procedures to ensure reliable operation, which limits its practical usability. To address this limitation, integrating additional sensing capabilities, such as an IMU sensor or enhanced distance sensing, would significantly enhance the robot's reliability by closing the navigational control loop. This would enable the robot to adapt to varying field conditions and navigate more effectively. Furthermore, redesigning the pollen storage system to accommodate a larger capacity and minimize delivery errors is recommended. By implementing these improvements, the robot's overall performance and usability can be substantially enhanced, paving the way for more successful and reliable autonomous operation in various environments.

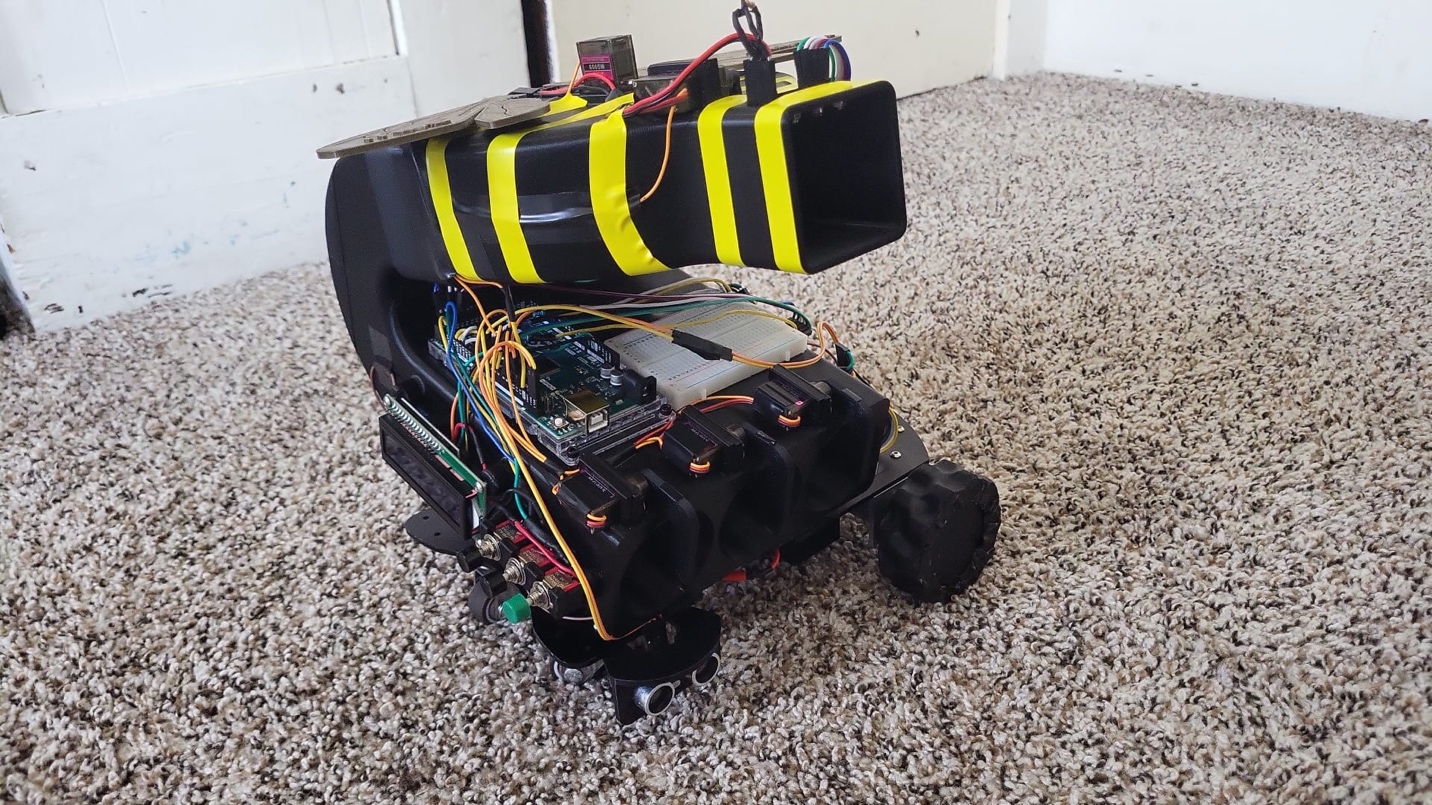
## Appendix A: Circuit Diagram



## Appendix B: Bill of Materials

|  |  |  |  |
| --- | --- | --- | --- |
| **Received From Past Year** | | | |
| **ITEM** | **QTY** | **PRICE** | **TOTAL PRICE** |
| 12V Motors | 2 | 20 | $ 40.00 |
| Plate | 1 | 10 | $ 10.00 |
| L298N Motor Driver | 1 | 3 | $ 3.00 |
| Color Sensor | 1 | 10 | $ 10.00 |
| Ultrasonic | 2 | 5 | $ 10.00 |
| **Total** |  |  | **$ 63.00** |
| **School Purchase** | | | |
| LIPO Battery | 1 | 18 | $ 18.00 |
| Buck Converter | 1 | 8 | $ 8.00 |
| 3M Screw | 1 | 25 | $ 25.00 |
| Push Buttons | 3 | 2 | $ 6.00 |
| **Total** |  |  | **$ 69.00** |
| **Personal Purchase** | | | |
| MG90S | 5 | 3 | $ 15.00 |
| LCD Display | 1 | 5.5 | $ 5.50 |
| Screw Terminals | 1 | 14 | $ 14.00 |
| Bread Board | 1 | 3 | $ 3.00 |
| Deans T Connector | 1 | 1 | $ 1.00 |
| Wire | 1 | 15 | $ 15.00 |
| Filament | 2 | 20 | $ 40.00 |
| Arduino Mega | 1 | 40 | $ 40.00 |
| Caster Wheels | 2 | 5 | $ 10.00 |
| **Total** |  |  | **$ 143.50** |
| **Robot Total** |  |  | **$ 275.50** |
| **Manufactured Parts** | | | |
| **ITEM** | **QTY** |  | **Method** |
| Main Body | 1 |  | 3D Printed |
| Power Housing | 1 |  | 3D Printed |
| Power Board | 1 |  | Solder |
| Plant Gates | 3 |  | 3D Printed |
| Sorting Flap | 1 |  | 3D Printed |
| Front Ultra Mount | 1 |  | 3D Printed |
| Side Ultra Mount | 1 |  | 3D Printed |
| Caster Spacers | 2 |  | 3D Printed |
| Battery Housing | 1 |  | 3D Printed |
| Motor Mounts | 2 |  | Machined |

## Appendix C: Manufactured Robot



## Appendix D: Full Code

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include <Servo.h>

#include <TimerOne.h>

//Push Button Pins

const int buttonPin1 = 22; // Pin 4 for the button 1

const int buttonPin2 = 24; // Pin 3 for the button 2

const int buttonPin3 = 26; // Pin 2 for the button 3

// Color Sorting Pins

#define s0 23 //Module pins wiring

#define s1 25

#define s2 27

#define s3 29

#define out 31

//States

const int LANE = 200;

const int THRESHOLD = 201;

const int COLOR\_DETECTION = 202;

const int SORT = 203;

const int START = 204;

const int PLANT\_DRIVE = 205;

const int WALL\_DRIVE = 206;

const int TURNING = 207;

const int STOP = 208;

const int END = 209;

int state = 200;

//Inputs

int clickCount1 = 0; // Variable to store the click count

int clickCount2 = 0;

int clickCount3 = 0;

int LastclickCount1 = 5;

int LastclickCount2 = 5;

int LastclickCount3 = 5;

int lastButtonState1 = 0; // Variable to store the last state of the button

int lastButtonState2 = 0;

int lastButtonState3 = 0;

int plant1 = 100;

int plant2 = 100;

int plant3 = 100;

int UNKNOWN = 100;

int YELLOW = 101;

int BLUE = 102;

int GREEN = 103;

int RED = 104;

String lane1 = "";

String lane2 = "";

String lane3 = "";

//Sorting

int Red\_0=0, Blue\_0=0, Green\_0=0;

int Red\_=0, Blue\_=0, Green\_=0; //RGB values

int TYellow=0, TRed=0, TBlue=0, TGreen=0; //Total Balls of each color sorted

int Ball = UNKNOWN;

int FULL = 4;

//Servos

Servo ServoSort;

Servo ServoGate;

Servo ServoP3;

Servo ServoP2;

Servo ServoP1;

const byte SorterServo = 13;

const byte Lane1angle = 35;

const byte Lane2angle = 85;

const byte Lane3angle = 135;

const byte GateServo = 11;

const byte OpenGate = 100;

const byte CloseGate = 0;

const byte Plant3Servo = 4;

const byte Plant2Servo = 5;

const byte Plant1Servo = 6;

const byte OpenPlant = 90;

const byte ClosePlant = 180;

//Timer

long int TimeRem = 120000;

long int TimeCon = 0;

long int TimeStart = 0;

//Driving

int PlantCount = 0;

// Encoder setup

const int LCHA\_PIN = 18;

const int LCHB\_PIN = 17;

const int RCHA\_PIN = 19;

const int RCHB\_PIN = 16;

volatile int Lenc\_count = 0; volatile int Lenc\_change = 0;

int Ltemp\_count = 0;

volatile int Renc\_count = 0; volatile int Renc\_change = 0;

int Rtemp\_count = 0;

long int speed\_time = 0;

float Lspeed = 0;

float Rspeed = 0;

float desired\_speed = 200;

float sampling\_delay = 10;

long int pos\_time = 0; float turnPos = 420; float Rpos = 0; bool turnSwitch = 0; long int turnSettle = 6000;

// Ultrasonic setup

#define TIMER\_US 100 // 50 uS timer duration

#define TICK\_COUNTS 4000 // 200 mS worth of timer ticks

const int RtrigPin = 12;

const int RechoPin = 2;

const int FtrigPin = 10;

const int FechoPin = 3;

volatile long echo\_start = 0; // Records start of echo pulse

volatile long echo\_end = 0;

volatile long echo\_duration = 0;

volatile int trigger\_time\_count = 0;

volatile long range\_flasher\_counter = 0;

long int ultraSpace = 69; // variable to control distance change reading

// Motor control setup

int leftMotor = 8;

int rightMotor = 9;

int In1 = 34;

int In2 = 32;

int In3 = 30;

int In4 = 28;

float Lerror = 0; float Lerror\_prior = 0; float Lintegral\_prior = 0; float Lintegral = 0; float Lderivative = 0; float Ldrive = 0;

float Rerror = 0; float Rerror\_prior = 0; float Rintegral\_prior = 0; float Rintegral = 0; float Rderivative = 0; float Rdrive = 0; float Rturn = 0;

int actual\_time = 0;

float LKP = .1; float LKI = 0.03; float LKD = .5; float RKP = .1; float RKI = 0.03; float RKD = .5;

float Rscaling\_factor = 7; float Lscaling\_factor = 7;

//SubFSM setup

int prevstate = STOP;

//route Setup

int wall = 1;

int lap = 0;

#if defined(ARDUINO) && ARDUINO >= 100

#define printByte(args) write(args);

#else

#define printByte(args) print(args,BYTE);

#endif

LiquidCrystal\_I2C lcd(0x27,20,4);

void setup() {

pinMode(buttonPin1, INPUT); // Set button pin as input

pinMode(buttonPin2, INPUT); // Set button pin as input

pinMode(buttonPin3, INPUT); // Set button pin as input

lcd.init(); // initialize the lcd

lcd.backlight();

Serial.begin(9600); // Initialize serial communication

pinMode(s0,OUTPUT); //pin modes

pinMode(s1,OUTPUT);

pinMode(s2,OUTPUT);

pinMode(s3,OUTPUT);

pinMode(out,INPUT);

digitalWrite(s0,HIGH); //Putting S0/S1 on HIGH/HIGH levels means the output frequency scalling is at 100% (recommended)

digitalWrite(s1,HIGH); //LOW/LOW is off HIGH/LOW is 20% and LOW/HIGH is 2%

//Servo

ServoSort.attach(SorterServo);

ServoGate.attach(GateServo);

ServoP3.attach(Plant3Servo);

ServoP2.attach(Plant2Servo);

ServoP1.attach(Plant1Servo);

ServoSort.write(Lane2angle); // Set default position

ServoGate.write(CloseGate); // Set default position

ServoP3.write(ClosePlant); // Set default position

ServoP2.write(ClosePlant); // Set default position

ServoP1.write(ClosePlant); // Set default position

pinMode(LCHA\_PIN, INPUT);

pinMode(LCHB\_PIN, INPUT);

pinMode(RCHA\_PIN, INPUT);

pinMode(RCHB\_PIN, INPUT);

attachInterrupt(digitalPinToInterrupt(LCHA\_PIN), Lencoder, RISING);

attachInterrupt(digitalPinToInterrupt(RCHA\_PIN), Rencoder, RISING);

pinMode(leftMotor, OUTPUT);

pinMode(rightMotor, OUTPUT);

pinMode(In1, OUTPUT);

pinMode(In2, OUTPUT);

pinMode(In3, OUTPUT);

pinMode(In4, OUTPUT);

pinMode(RtrigPin, OUTPUT); // Sets trigPin as output

pinMode(RechoPin, INPUT); // Sets echoPin as input

pinMode(FtrigPin, OUTPUT); // Sets trigPin as output

pinMode(FechoPin, INPUT); // Sets echoPin as input

Timer1.initialize(TIMER\_US); // Initialise timer 1

Timer1.attachInterrupt(timerIsr); // Attach interrupt to the timer service routine

attachInterrupt(digitalPinToInterrupt(2), Recho\_interrupt, CHANGE); // Attach interrupt to the sensor echo input

attachInterrupt(digitalPinToInterrupt(3), Fecho\_interrupt, CHANGE);

}

void loop() {

// Read the current state of the button

int buttonState1 = digitalRead(buttonPin1);

int buttonState2 = digitalRead(buttonPin2);

int buttonState3 = digitalRead(buttonPin3);

switch (state) {

case LANE:

// Check for rising edge (button going from LOW to HIGH)

if (buttonState1 == HIGH && lastButtonState1 == LOW) {

clickCount1++; // Increment the click count

}

if (buttonState2 == HIGH && lastButtonState2 == LOW) {

clickCount2++; // Increment the click count

}

if (buttonState3 == HIGH && lastButtonState3 == LOW) {

if (plant1 == UNKNOWN || plant2 == UNKNOWN || (plant1==plant2)){

clickCount3 =0;

}

else{

clickCount3++; // Increment the click count

}

}

// Update the last button state

lastButtonState1 = buttonState1;

lastButtonState2 = buttonState2;

lastButtonState3 = buttonState3;

if (clickCount3 == 0){

if(clickCount1 != LastclickCount1){

lcd.setCursor(0,0);

lcd.print("Plant 1 ");

if (clickCount1 == 3) {

plant1 = RED;

lane1 ="Red ";

lcd.print(lane1);

}

else if (clickCount1 == 2) {

plant1 = GREEN;

lane1 ="Green ";

lcd.print(lane1);

}

else if (clickCount1 == 1) {

plant1 = BLUE;

lane1 ="Blue ";

lcd.print(lane1);

}

else {

plant1 = UNKNOWN;

lane1 ="Unknown ";

clickCount1 = 0; //reset click count1

lcd.println(lane1);

}

LastclickCount1 = clickCount1;

}

if(clickCount2 != LastclickCount2 ){

lcd.setCursor(0,1);

lcd.print("Plant 2 ");

if (clickCount2 == 3) {

plant2 = RED;

lane2 ="Red ";

lcd.print(lane2);

}

else if (clickCount2 == 2) {

plant2 = GREEN;

lane2 ="Green ";

lcd.print(lane2);

}

else if (clickCount2 == 1) {

plant2 = BLUE;

lane2 ="Blue ";

lcd.print(lane2);

}

else {

plant2 = UNKNOWN;

lane2 ="Unknown ";

clickCount2 = 0; //reset click count1

lcd.println(lane2);

}

LastclickCount2 = clickCount2;

}

}

if(clickCount3 != LastclickCount3 && clickCount3 != 0 && (plant1 != UNKNOWN || plant2 != UNKNOWN) && (plant1 != plant2) ){

if (lane1 != lane2 && clickCount3 == 1){

lcd.clear();

lcd.print("P1:");

if (plant1 == BLUE){

lcd.print("B ");

}

else if (plant1 == GREEN){

lcd.print("G ");

}

else if (plant1 == RED){

lcd.print("R ");

}

lcd.print("P2:");

if (plant2 == BLUE){

lcd.print("B ");

}

else if (plant2 == GREEN){

lcd.print("G ");

}

else if (plant2 == RED){

lcd.print("R ");

}

if (plant1 == UNKNOWN || plant2 == UNKNOWN){

//Welcome Screen

lcd.clear();

}

else if (plant1 != BLUE && plant2 != BLUE){

plant3 = BLUE;

}

else if (plant1 != GREEN && plant2 != GREEN){

plant3 = GREEN;

}

else if (plant1 != RED && plant2 != RED){

plant3 = RED;

}

lcd.print("P3:");

if (plant3 == BLUE){

lcd.print("B ");

}

else if (plant3 == GREEN){

lcd.print("G ");

}

else if (plant3 == RED){

lcd.print("R ");

}

}

else if (clickCount3 == 2){

clickCount3 = 0;

state = THRESHOLD;

lcd.clear();

lcd.print("SORTING");

break;

}

LastclickCount3 = clickCount3;

}

delay(100);

break;

case THRESHOLD: {

GetColors(); //Execute the GetColors function to get the value of each RGB color

//Depending of the RGB values given by the sensor we can define the color and displays it on the monitor

Red\_0=Red\_;

Blue\_0=Blue\_;

Green\_0=Green\_;

state = COLOR\_DETECTION;

}break;

case COLOR\_DETECTION: {

ServoGate.write(CloseGate);

delay(500);

GetColors();

if (Red\_>35 && Blue\_>40 && Green\_>40){ //If the Red, Blue, and Green values within threshhold range, there is no ball

Ball = UNKNOWN;

delay(500); //2s delay

}

else if (Red\_<=Blue\_ && Red\_<=Green\_ && Red\_<20 && Green\_>15 && Blue\_>10){ //if Red value is the lowest one and smaller thant 23 it's likely Red

delay(500);

if (Red\_ <= Blue\_ && Red\_ <= Green\_ && Red\_<20 && Green\_ > 15 && Blue\_ >10){

Ball = RED;

if (Ball == plant1){

ServoSort.write(Lane1angle);

}

else if (Ball == plant2){

ServoSort.write(Lane2angle);

}

else if (Ball == plant3){

ServoSort.write(Lane3angle);

} //2s delay

state = SORT;

}

}

else if (Blue\_<Green\_ && Blue\_<Red\_ && Blue\_<15 && Red\_>15){ //Same thing for Blue

delay(500);

if (Blue\_<Green\_ && Blue\_<Red\_ && Blue\_<15 && Red\_>15){

Ball = BLUE;

if (Ball == plant1){

ServoSort.write(Lane1angle);

}

else if (Ball == plant2){

ServoSort.write(Lane2angle);

}

else if (Ball == plant3){

ServoSort.write(Lane3angle);

}

state = SORT;

}

}

else if (Green\_<=Red\_ && Red\_>5 && Blue\_>= 5 && Red\_<25 && Green\_<20){ //Green it was a little tricky, you can do it using the same method as above (the lowest), but here I used a reflective object

delay(500);

if (Green\_<=Red\_ && Red\_>5 && Blue\_>= 5 && Red\_<25 && Green\_<20){

Ball = GREEN;

if (Ball == plant1){

ServoSort.write(Lane1angle);

}

else if (Ball == plant2){

ServoSort.write(Lane2angle);

}

else if (Ball == plant3){

ServoSort.write(Lane3angle);

} //2s delay

state = SORT;

}

}

}break;

case SORT: {

if (Ball == UNKNOWN){

state = COLOR\_DETECTION;

}

else if (Ball == RED){

TRed++;

ServoGate.write(OpenGate);

Ball = UNKNOWN;

state = COLOR\_DETECTION;

}

else if (Ball == BLUE){

TBlue++;

ServoGate.write(OpenGate);

Ball = UNKNOWN;

state = COLOR\_DETECTION;

}

else if (Ball == GREEN){

TGreen++;

ServoGate.write(OpenGate);

Ball = UNKNOWN;

state = COLOR\_DETECTION;

}

if (TRed == FULL || TBlue == FULL || TGreen == FULL){

state = START;

}

else{

delay(500); //2s delay

Ball = UNKNOWN;

state = COLOR\_DETECTION;

}

}break;

case START:

TimeStart = millis();

state = PLANT\_DRIVE;

lcd.clear();

lcd.print("DRIVING");

break;

case PLANT\_DRIVE:

if(millis()-speed\_time > sampling\_delay){

actual\_time = millis()-speed\_time;

Lspeed = (Lenc\_count-Ltemp\_count)\*1000/(actual\_time\*Lscaling\_factor);

Rspeed = (Renc\_count-Rtemp\_count)\*1000/(actual\_time\*Rscaling\_factor);

Ltemp\_count = Lenc\_count;

Rtemp\_count = Renc\_count;

speed\_time = millis();

Lerror = (1.1\*desired\_speed)-Lspeed;

Lintegral = Lintegral\_prior + Lerror\*actual\_time;

Lderivative = (Lerror-Lerror\_prior)/actual\_time;

Ldrive = LKI\*Lintegral;

Lerror\_prior = Lerror;

Lintegral\_prior = Lintegral;

Rerror = (1\*desired\_speed)-Rspeed;

Rintegral = Rintegral\_prior + Rerror\*actual\_time;

Rderivative = (Rerror-Rerror\_prior)/actual\_time;

Rdrive = RKI\*Rintegral;

Rerror\_prior = Rerror;

Rintegral\_prior = Rintegral;

if(Ldrive > 255){

Ldrive = 255;

}

if(Ldrive < 0){

Ldrive = 0;

}

if(Rdrive > 255){

Rdrive = 255;

}

if(Rdrive < 0){

Rdrive = 0;

}

}

TimeCon = TimeRem - millis() + TimeStart;

if(TimeCon <= 0){

state = END;

lcd.clear();

lcd.print("COMPLETED");

break;

}

if((echo\_duration/58) > 30 & echo\_duration/58 < 90 & ultraSpace == 69){

ultraSpace = millis();

}

else if(millis()-ultraSpace > 50 & ultraSpace != 69){

ultraSpace = 69;

if((echo\_duration/58) > 30 & echo\_duration/58 < 90){

state = STOP;

prevstate = PLANT\_DRIVE;

speed\_time = millis();

}

}

break;

case WALL\_DRIVE:

TimeCon = TimeRem - millis() + TimeStart;

if(TimeCon <= 0){

state = END;

lcd.clear();

lcd.print("COMPLETED");

break;

}

if(millis()-speed\_time > sampling\_delay){

actual\_time = millis()-speed\_time;

Lspeed = (Lenc\_count-Ltemp\_count)\*1000/(actual\_time\*Lscaling\_factor);

Rspeed = (Renc\_count-Rtemp\_count)\*1000/(actual\_time\*Rscaling\_factor);

Ltemp\_count = Lenc\_count;

Rtemp\_count = Renc\_count;

speed\_time = millis();

Lerror = (1.1\*desired\_speed)-Lspeed;

Lintegral = Lintegral\_prior + Lerror\*actual\_time;

Lderivative = (Lerror-Lerror\_prior)/actual\_time;

Ldrive = LKP\*Lerror + LKI\*Lintegral + LKD\*Lderivative;

Lerror\_prior = Lerror;

Lintegral\_prior = Lintegral;

Rerror = (1\*desired\_speed)-Rspeed;

Rintegral = Rintegral\_prior + Rerror\*actual\_time;

Rderivative = (Rerror-Rerror\_prior)/actual\_time;

Rdrive = RKP\*Rerror + RKI\*Rintegral + RKD\*Rderivative;

Rerror\_prior = Rerror;

Rintegral\_prior = Rintegral;

if(Ldrive > 255){

Ldrive = 255;

}

if(Ldrive < 0){

Ldrive = 0;

}

if(Rdrive > 255){

Rdrive = 255;

}

if(Rdrive < 0){

Rdrive = 0;

}

}

if(((echo\_duration/58) < 37) & ((echo\_duration/58) > 2)){

prevstate = WALL\_DRIVE;

state = STOP;

speed\_time = millis();

}

break;

case TURNING: {

TimeCon = TimeRem - millis() + TimeStart;

if(TimeCon <= 0){

state = END;

lcd.clear();

lcd.print("COMPLETED");

break;

}

if(millis()-pos\_time > sampling\_delay){

Rpos = Renc\_count/Rscaling\_factor;

Rtemp\_count = Renc\_count;

pos\_time = millis();

Rerror = (turnPos)-Rpos;

Rderivative = (Rerror-Rerror\_prior)/sampling\_delay;

Rturn = 10\*Rerror;

if(Rpos > (turnPos-10)){

Rintegral = Rintegral\_prior + Rerror\*sampling\_delay;

Rturn = 15\*Rerror + .01\*Rintegral;

}

Rerror\_prior = Rerror;

Rintegral\_prior = Rintegral;

}

if(Rturn < 0){

//Rturn = abs(Rturn);

turnSwitch = 1;

}

else{

turnSwitch = 0;

}

if(Rturn > 255){

Rturn = 255;

}

if(Rpos > turnPos){

if(turnSettle == 6000){

turnSettle = millis();

}

}

if(millis()-turnSettle > 1000 & turnSettle != 6000){

state = STOP;

prevstate = TURNING;

speed\_time = millis();

turnSettle = 6000;

}

//state = PLANT\_DRIVE;

}break;

case STOP: {

TimeCon = TimeRem - millis() + TimeStart;

if(TimeCon <= 0){

state = END;

lcd.clear();

lcd.print("COMPLETED");

break;

}

if(millis()-speed\_time > 1000){

Lerror = 0; Lerror\_prior = 0; Lintegral\_prior = 0; Lintegral = 0; Lderivative = 0; Ldrive = 0;

Rerror = 0; Rerror\_prior = 0; Rintegral\_prior = 0; Rintegral = 0; Rderivative = 0; Rdrive = 0;

Lenc\_count = 0; Lenc\_change = 0; Ltemp\_count = 0; Renc\_count = 0; Renc\_change = 0; Rtemp\_count = 0; Lspeed = 0; Rspeed = 0; Rpos = 0;

if(prevstate == PLANT\_DRIVE){

if (wall == 1){

//insert servo code here

ServoP1.write(OpenPlant);

delay(2000);

ServoP1.write(ClosePlant);

}

else if(wall == 2){

//insert servo code here

ServoP2.write(OpenPlant);

delay(2000);

ServoP2.write(ClosePlant);

}

else if(wall == 3){

//insert servo code here

ServoP3.write(OpenPlant);

delay(2000);

ServoP3.write(ClosePlant);

}

state = WALL\_DRIVE;

}

if(prevstate == WALL\_DRIVE){

wall++;

state = TURNING;

if(wall == 3){

turnPos = 410;

}

else{

turnPos = 420;

}

}

if(prevstate == TURNING | prevstate == STOP){

if(wall == 4){

state = WALL\_DRIVE;

}

else if(wall == 1 | wall == 2 | wall == 3){

state = PLANT\_DRIVE;

}

else if(wall == 5){

wall = 1;

lap++;

state = THRESHOLD;

}

state = PLANT\_DRIVE;

}

prevstate = STOP;

echo\_duration = 100\*58;

}

break;

}

case END: {

}break;

}

if(state == WALL\_DRIVE | state == PLANT\_DRIVE){

digitalWrite(In1, LOW);

digitalWrite(In2, HIGH);

digitalWrite(In3, LOW);

digitalWrite(In4, HIGH);

analogWrite(leftMotor, Ldrive);

analogWrite(rightMotor, Rdrive);

}

else if(state == TURNING){

digitalWrite(In1, HIGH);

digitalWrite(In2, HIGH);

if(turnSwitch == 1){

digitalWrite(In3, HIGH);

digitalWrite(In4, LOW);

}

else{

digitalWrite(In3, LOW);

digitalWrite(In4, HIGH);

}

analogWrite(leftMotor, 0);

analogWrite(rightMotor, Rturn);

}

else{

digitalWrite(In1, HIGH);

digitalWrite(In2, HIGH);

digitalWrite(In3, HIGH);

digitalWrite(In4, HIGH);

analogWrite(leftMotor, 255);

analogWrite(rightMotor, 255);

}

}

void Rencoder() {

Renc\_change = readEncoder(RCHB\_PIN);

if (Renc\_change != 0) {

Renc\_count = Renc\_count - Renc\_change;

}

}

void Lencoder() {

Lenc\_change = readEncoder(LCHB\_PIN);

if (Lenc\_change != 0) {

Lenc\_count = Lenc\_count + Lenc\_change;

}

}

int result = 0;

int chA\_last = 0;

int chB\_last = 0;

int chA\_new = 0;

int chB\_new = 0;

int readEncoder(int chB) {

result = 0;

chB\_new = digitalRead(chB);

if ((chB\_new == HIGH)) { // CW turn

result = 1;

}

if ((chB\_new == LOW)) { // CW turn

result = -1;

}

return result;

}

void timerIsr() {

static volatile int ultrastate = 0; // State machine variable

if (!(--trigger\_time\_count)) // Count to 200mS

{ // Time out - Initiate trigger pulse

trigger\_time\_count = TICK\_COUNTS; // Reload

ultrastate = 1; // Changing to state 1 initiates a pulse

}

switch(ultrastate) // State machine handles delivery of trigger pulse

{

case 0: // Normal state does nothing

break;

case 1:

if(state == PLANT\_DRIVE){

digitalWrite(RtrigPin, HIGH);

}

else if(state == WALL\_DRIVE){

digitalWrite(FtrigPin, HIGH);

}

else{

digitalWrite(RtrigPin, LOW);

digitalWrite(FtrigPin, LOW);

}

ultrastate = 2; // and set state to 2

break;

case 2: // Complete the pulse

default:

if(state == PLANT\_DRIVE){

digitalWrite(RtrigPin, LOW);

}

else if(state == WALL\_DRIVE){

digitalWrite(FtrigPin, LOW);

}

else{

digitalWrite(RtrigPin, LOW);

digitalWrite(FtrigPin, LOW);

}

ultrastate = 0; // and return state to normal 0

break;

} // Flash the onboard LED distance indicator

}

void Recho\_interrupt(){

switch(digitalRead(RechoPin)) // Test to see if the signal is high or low

{

case HIGH: // High so must be the start of the echo pulse

echo\_end = 0; // Clear the end time

echo\_start = micros(); // Save the start time

break;

case LOW: // Low so must be the end of hte echo pulse

echo\_end = micros();

if(state == PLANT\_DRIVE | state == WALL\_DRIVE){

echo\_duration = echo\_end - echo\_start;

}

break;

}

}

void Fecho\_interrupt(){

switch(digitalRead(FechoPin)) // Test to see if the signal is high or low

{

case HIGH: // High so must be the start of the echo pulse

echo\_end = 0; // Clear the end time

echo\_start = micros(); // Save the start time

break;

case LOW: // Low so must be the end of hte echo pulse

echo\_end = micros(); // Save the end time

if(state == PLANT\_DRIVE | state == WALL\_DRIVE){

echo\_duration = echo\_end - echo\_start;

}

break;

}

}

void GetColors(){

digitalWrite(s2, LOW); //S2/S3 levels define which set of photodiodes we are using LOW/LOW is for RED LOW/HIGH is for Blue and HIGH/HIGH is for green

digitalWrite(s3, LOW);

Red\_ = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH); //here we wait until "out" go LOW, we start measuring the duration and stops when "out" is HIGH again, if you have trouble with this expression check the bottom of the code

delay(20);

digitalWrite(s3, HIGH); //Here we select the other color (set of photodiodes) and measure the other colors value using the same techinque

Blue\_ = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);

delay(20);

digitalWrite(s2, HIGH);

Green\_ = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);

delay(20);

}