ECE160 - 02 Engineering Practice

Zany Zombie Zamboree

Final Project Report

Fall 2017

Team 02: RoboCops
Scott Cai
Piotr Patryk Galas

November 11, 2017

Instructor: Dr. J. J. Song

Contents

Executive Summary

The purpose of this report is to describe in detail the implementation, hardships, and thought processes when constructing the Zombie Gathering Robot in ECE 160. The ultimate goal of this project was to construct and encode a robot that was capable of picking up different colored zombies and dropping them into different various sized holes and bases. This robot consisted of various sensors (obstacle sensors in this case), a wireless system, servos, and an arduino and its shield. In order to bring all of these things together the team was split up into two subsystems, hardware and software. These subsystems had to work together in order for the robot to operate properly. The hardware team came up with ideas on which sensors to utilize and where to attached them, the programming team tested each sensor in order to find their accuracy and stability on the platform. During the testing of the robot the programming team made sure that the robot was fully wireless using a PS2 controller and another arduino through RF24. Once the hardware team finalized the placement of all sensors, the programming team made the robot completely autonomous. The autonomous part of the project lasted for 30 seconds, and the robot was able to determine the difference between plants and zombies. In the end the team made a fully autonomous robot that was capable of being controller over a wireless system. The team was able to accomplish all the requirements for the final project and also had time to practice setting up and driving the robot. The results were successful but due to the misconnection of the controller and the robot, the team failed to compete in the final competition. In conclusion, the team recommends that one way to improve this project would be to add color sensors to the robot and also add more accurate sensors which could fix many errors in the autonomous part of the code.

Introduction

The purpose of this report is to summarize in detail the implementation, and thought processes when building the Zombie Gathering Robot in ECE 160. The project was to construct and build a wireless and autonomous robot that was able to pick up different colored zombies and put them in various places to score points. The robot was programmed to utilize object avoidance sensors to find walls and zombies on the game board. Using sensor feedback the robot would decided what it has to do and where it has to move too. Using the mini-projects and other lectures from the beginning of the quarter the group was able to code a simple robot that was able to move on its own. The group utilized the knowledge learned from mini-projects like breadboarding, arrays, soldering, wireless transmission, loops, machine states, and if statements to create the physical part of the robot and the decision matrix that the robot used to determine what it should do next.

Requirements

The zombie gathering robot's main function was to pick up zombies and place them into either various sized holes, or the robots homebase. The robot itself was supposed to be autonomous and wirelessly operated using either a RF24 module or a PS2 controller.

Requirements for Autonomous:

- The robot has to be autonomous for no more than 30 seconds
- The robot can switch right into remote control mode flawlessly after 30 seconds
- Must utilized sensor feedback to decide what it should do next
- Deliver zombies and plants to different holes and storage areas
- The robot can decide between plants and zombies using the sensors
- Use the grippers to pick up the zombies and move them

Requirements for Remote Control:

- Must be completely wireless
- Can pick up various plants and zombies, and deliver them to certain holes and storage areas
- Can be driven accurately and without a problem
- Move forward, backward, turn left and right
- Use the grippers to pick up the zombies and move them

The team was also required to split into three subsystems, hardware, programming, and Secretary. The group member in charge of hardware had to mount all the sensors and determine which sensors would be the best to use for the autonomous portion of the project. The group member in charge of programming had to code the remote control and autonomous part of the robot. The group member that was the secretary had to write the team memorandums, meetings, and minutes, and also kept the team on track by creating a list of accomplished goals and goals that the team wanted to accomplish in the next week. In order for each group member to experience each subsystem, the group rotated positions every meeting so each group member could learn how to program, create circuits, and write memorandums and other important papers.

Literature Review

The team utilized several different literary sources in order to further our knowledge on how certain sensors and function works. The team used the Arduino Forums in order to get assistance with coding certain aspects of the robot like, the timer, the servos, the mapping of the controls, and the sensors (Arduino Forums). The team used the forums in order to see how the functions like map work and how they can be implemented in the project to make the way our robot move more perscies. The team also used websites the provide test code to make sure each sensor was detecting objects correctly. The team did this for the obstacle avoidance sensor. The website provided the team with code that could be used to test the sensors and also provided the team with different ideas that we could use on our own robot (Henry's Bench). The team also used Instructables in order to find examples to how people approached similar projects. Instructables was mainly used to troubleshoot the PS2 controller due to there being many disconnection and connection problems. The instructables page titled “PS2 Wire Controller and Arduino (Control LEDs)” was used to see how PS2 controller should be plugged into the arduino. The group experienced trouble after connecting the PS2 controller and realized that there was a problem with the way we connected the pins. Some of the pins were not in the correct order (TranThinh, and Instructables). The last source the team used was a data sheet for the IR sensor. The group knew that the IR sensors was capable of detecting different colors. Black absorbs IR colors while other colors reflect different analog values. This data sheet allowed for us to determine the accuracy of this sensors and if it could used to detect the difference between plants and zombies, or between a white line and black floor (Vishay Semiconductors). The group was successful at this but the use of obstacle avoidance sensor in these ways was not very accurate. As seen by the literature used, the group's main focus was to understand each sensors better and use test programs to see if the circuit was built correctly. Ultimately these sources assisted this team greatly in making a successful robot.

Methods

The method used to complete this final project was to utilize all possible resources that each member had in order to make the robot operational. We worked as a team in order to complete each phase in the project. We didn't move on until everyone was on the same page, and we didn't let each other fall behind. This project had two main subsystems, one was hardware, and the other was programming. Shunhao was in charge of the hardware team, where he focused on which sensors were the best for this project, what additional sensors could be used, and where to mount each sensor on the robot. Piotr was in charge of programming team, where he focused on make the hardware work correctly through software. The main focus of the programming team was to make sure the robot was able to move autonomously and with a controller. In order for all team members to get experience with each subsystem, every meeting each team member would go to a different subsystem and every meeting we changed who was documenting the meeting. This can be seen in table 1.

Table 1: Team Member Responsibilities

	Week 7	Week 8	Week 9
Hardware	Shunhao mounted all needed sensors onto the robot and Piotr assisted with mounting the sensors on the robot. Shunhao also mounted and wired the PS2 controller as to make the robot wireless	Shunhao adjusted all the sensors so they do not interfere with each other. He also changed the placing of each sensor to account for the plants and zombies	Shunhao fixed up the mounting of the sensors, and made sure each sensor worked without outside interference
Software	Shunhao write the code of the robot, while Piotr wrote the pseudocode for the robot Throughout the week we have worked on converting the pseudocode to operational code	During the meeting Piotr was coding and testing the robot to see how the autonomous portion of the robot worked	Piotr perfected the autonomous part of the robot, made sure the autonomous part only lasts for 30 seconds, made sure that the controller successfully connects to the robot at the start of the driving phase, practiced driving the robot and setting it up.

Documentation	Piotr wrote the team memo, minutes, and meeting for this week in order to keep track of what the team has done.	Piotr wrote the team memo, minutes, and meeting for this week in order to keep track of what the team has done. Shunhao read about other sensors that could help the group make the robot autonomous	Piotr wrote the team memo, minutes, and meeting for this week. Shunhao began to write out the final report and presentation

I. Subsystems

The first subsystem for this zombie gathering robot was hardware. The hardware subsystem was in charge of mounting all the sensors, maintaining the robot, and deciding which sensors are truly needed for this competition. As it can be seen in figures X-x to X-x there is a complete robot with the sensors that the hardware team attached. The second subsystem was the program or software. The software and programming subsystem was in charge of making sure the robot operated wirelessly, making sure the sensors operated as planned, and made sure that the robot was able to drive autonomously and manually.

The team also made decisions on what sensors were the best to use to make the robot autonomous. The sensor that were available to us were obstacle avoidance, IR, line detection, and sonar. At first, the group used the sonar sensor to detect whether there was an object in front of the robot. This worked until the code was changed and the sonar no longer read accurate values. At this moment of time we made a decision matrix on all the sensors. The criteria was object detection accuracy, robots speed, interferences, ease of use, and versatility. This can be seen in table 2. Based upon the ratings and the strengths and weaknesses of each sensor the team decided to use the obstacle avoidance sensor.

Table 2 Sensor Decision Matrix

	Obstacle Avoidance Sensor	**IR Sensor**	**Line Detection Sensor**	**Sonar Sensor**
Accuracy	9	2	1	4
Speed	9	4	8	4
Interference	3	1	8	7
Ease of Use	8	1	1	5

Versatility	10	1	1	3
Total	**39**	**9**	**19**	**23**

* This matrix is on the scale of one to ten. Ten being the best and one being the worst

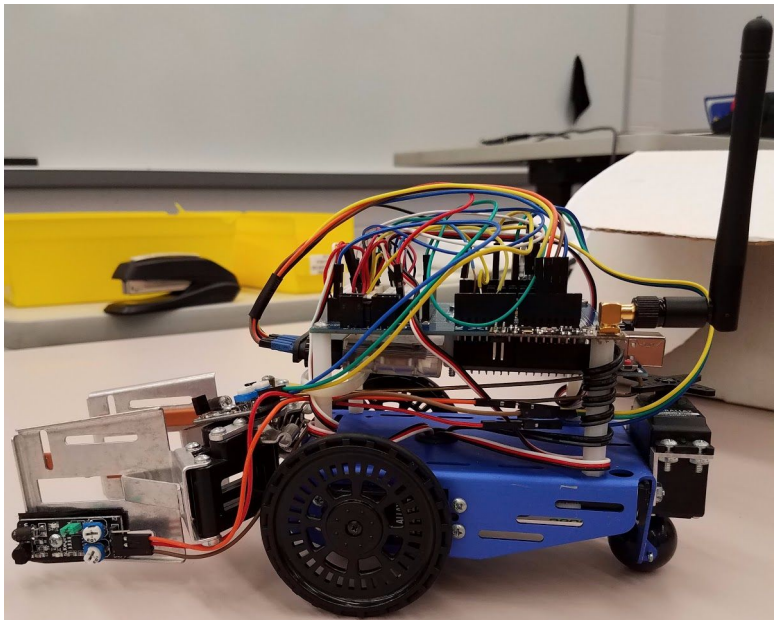

Figure 1: Side view of completed robot.

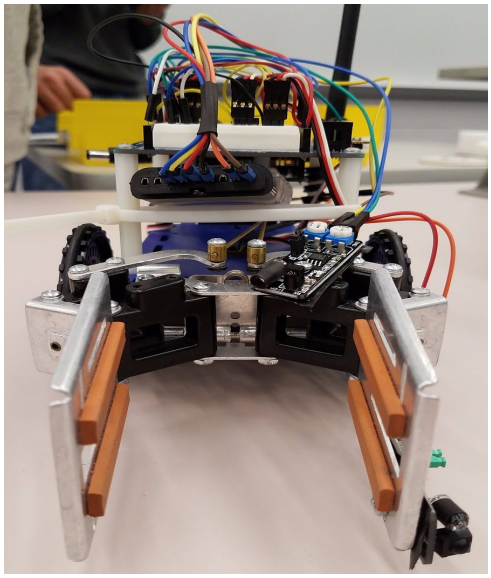

Figure 2: Front view of completed robot.

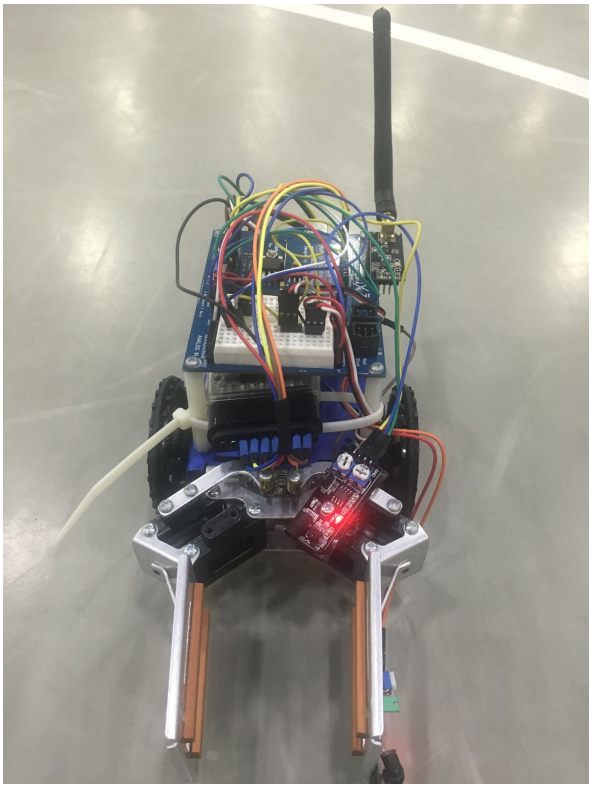

Figure 3: Robot ready for the competition

II. Troubleshooting

The team would usually troubleshoot the robot together when there was an error. When the robot failed to operate each member of the team would usually stop what they were doing and would come to assist the programmer or tester. The person testing the robot would explain what the problem is with the robot and then each member would see if they know why it could be occurring. If none of the members come to a verbal conclusion, each member would look over the code to find specific mistakes. If there was a mistake beyond our knowledge we would try to find a solution on the internet and if the problem was beyond that, the team would take the robot to their professor and ask what was wrong. Before speaking to our professor, we ran a few tests to see if our code was running properly. We would have the code print different lines to see if each parameter of the code was used. If this failed we checked over the wiring of our circuit, in the end after all our resources have been exhausted we would go to our professor and ask for assistance. In this project, the group only encountered two major problems that caused trouble when troubleshooting. The arduino that came with robot had a lot of misfires where it would run the code incorrectly or would miss a whole piece of code like it was never typed out even though all parameters for that block were true. Likewise, our wireless system did not work as well as other groups. The PS2 did not always connect and always seemed to not be recognized by the system or it would just disconnect in the middle of the competition.

III. Changes

In order to make any changes in the hardware or the software, all team members had to plead their changes and all members had to agree to it before implementing the change. Each change

that was implemented had to have a decision matrix drawn out to show why it was better than the current setup, and how the team would go about redesigning the robot and implementing the change. Every change that was made to the robot was written down in the team memorandums and in the team meeting minutes in order to keep track the changes in case we need to revert back to the old design. Also, every time there was major change involved in the code, the team would make a new folder for the code. This would make sure that we wouldn't lose the old code incase we needed to go back to it.

Bill of Materials

Quantity	Price ($)	Material	Function
2	-	Arduino Uno	Used to control the robot. This was the brain of the whole system. It was also used to receive and transmit data from the PS2 Controller and RF24 Modules
1	-	Parallax Shield Bot	This was the robot itself. It came with a shield with more I/O ports for the arduino, three servos, and a grabber
3	-	Obstacle Avoidance Sensors	These are IR sensors that can detect if there is an object in front of them. In this case they were used to detect zombies, plants, and walls
2	$10	RF24 Modules	These modules were used as a backup transmitter and receiver. These modules were able to transmit over a longer range than the standard module.
1	-	Line Detection Sensor	This is a IR sensor that can tell the difference between a black surface and a white surface. This is commonly used for to make a robot follow a line. Due to the way the game board was painted this sensor did not operate correctly
1	-	PS2 Wireless Controller and Receiver	The PS2 controller and receiver were used to control the robot wirelessly during the remote control part of the competition
10	-	AA Batteries	These double A batteries powered the whole robot. The robot came with 10 batteries and only 5 were used at a time to power the robot. This means that there was about 6 volts powering the whole system

Results

There were many requirements that had to be satisfied for the completion of the project. For the autonomous part of the project, the robot had to be in autonomous mode for only 30 seconds and once the 30 seconds ended the robot had to flawlessly go into remote control mode. The group experiences some troubles here. The group had to utilize an active timer that kept track of the time. The group used a while loop to keep track how many times a function was executed. This method worked but this method did not account for the delays in the function that was running making the autonomous mode unpredictable. Also when the robot exited autonomous mode the controller would not always connect to the robot. This means that the group would not be able to move the robot after the autonomous mode ended. In order to prevent these problems from occurring the group made the autonomous only run for 30 seconds by using extended delays that added up to 30 seconds. Fixing this also fixed the disconnection problem between the PS2 controller and the receiver. For the remote control portion of the competition the team successful satisfied the requirement for smooth transitions between autonomous mode and remote control mode, while also managing to move the robot accurately after the autonomous phase. This was demonstrated in Phase III of the project and in the competition. Once the autonomous mode ended the driver of our group was able to control the robot and pick up zombies with little to no delay once. Other then these main problems that were solved by changing the code, all of the requirements stated in the requirments part of the paper were satisfied by the group. The only other problem that the group encountered was a problem with the arduino. The arduino would sometimes have a misfire where it only runs part of the code and acts as if the rest of the code was commented out. This was a hardware problem with the arduino due to the amount of times it was used. Even if the code ran correctly the arduino would send a pulse to servos making them move or twitch when the robot should of been standing still. Although we had to overcome many of problems the group did complete all the requirements that were asked for and were able to compete at the final competition.

Conclusions

In conclusion, the team was able to satisfy all the requirements for the project. The robot was able to utilize the obstacle sensors to pick up zombies, find walls, and move the zombies to their correct spots. The arduinos were not the best to use for this projects due to their limitations, but the sensors that were used have poor reading. Although the equipment could be better, it is very well capable of making a fully autonomous robot. The results that were gather from the project show that with the given equipment and time it is possible to make a fully autonomous robot. Although the sensors were not the best and the designs of the robots were none customizable, students were capable of making different robots with different capabilities, strengths, and weaknesses. This projects captures the ability of all students and the ways this competition always changes because of creativity, limitations, and resources.

Schematics

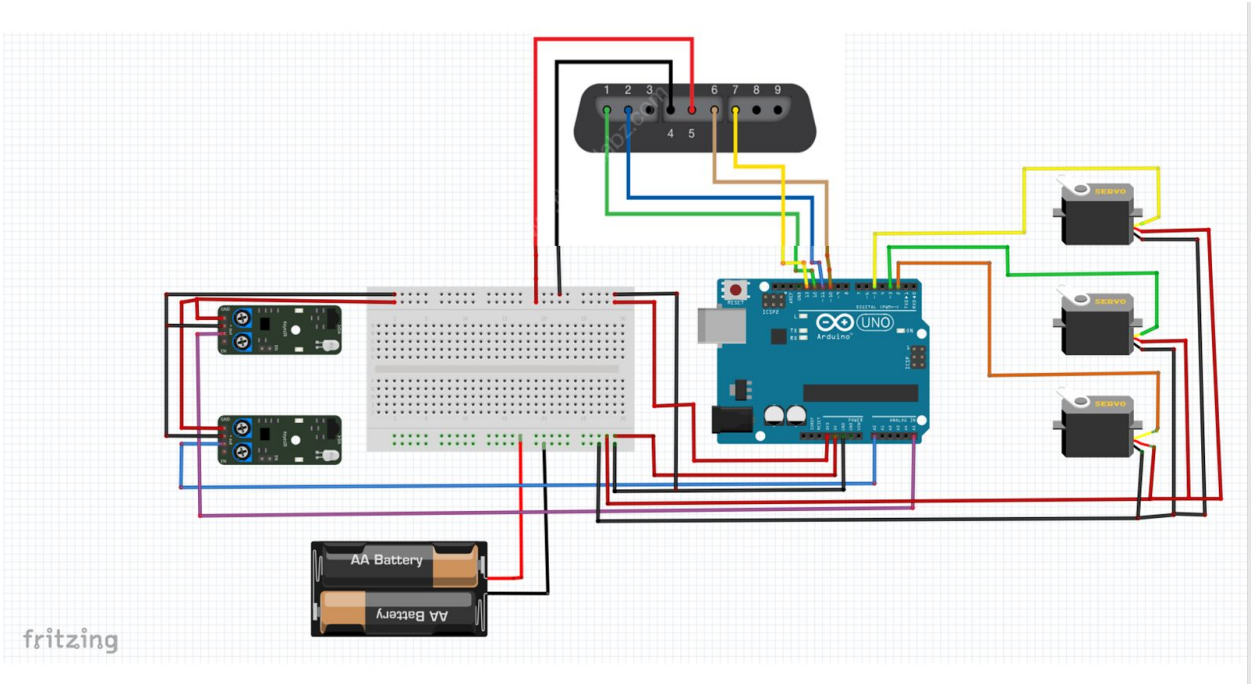

Figure 4: Schematics with fritzing

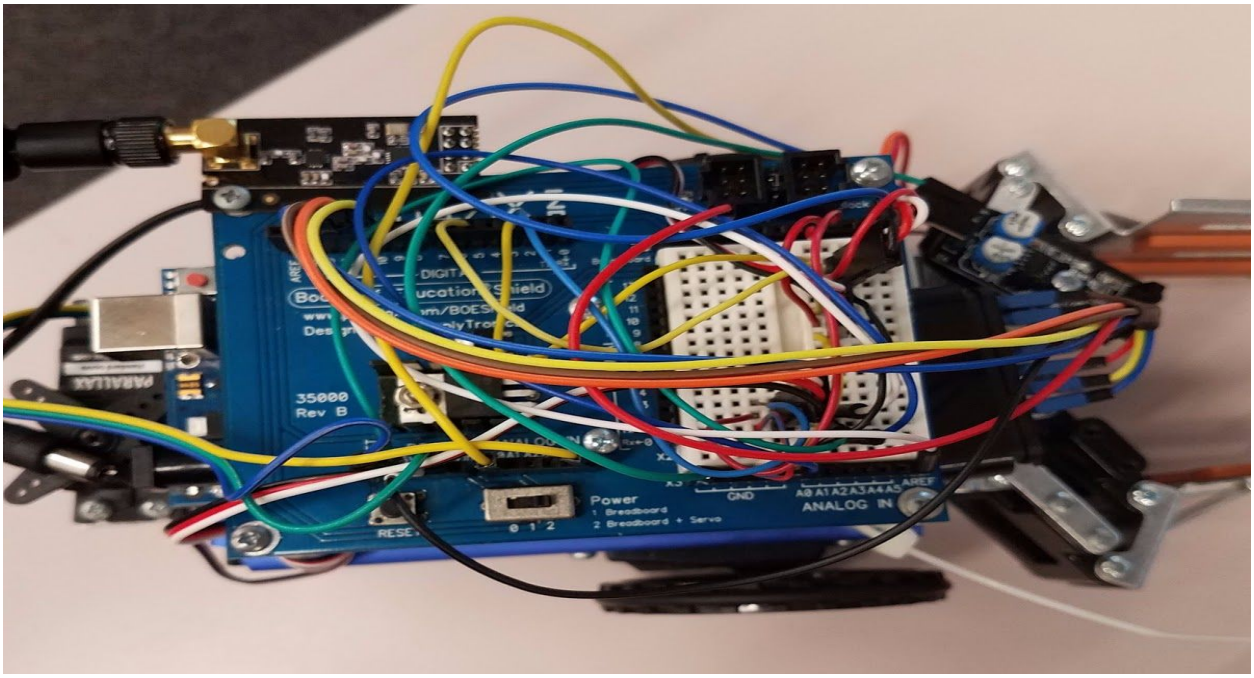

Figure 5: Actual wiring

Pin	Components
2	Rear/Grabber Servo
3	Right Servo
5	Left Servo
10	Attention Pin for PS2 Controller
11	Command Pin for PS2 Controller
12	Data Pin for PS2 Controller
13	Clock Pin for PS2 Controller
A0	Gripper Obstacle Avoidance Sensor
A5	Wall Obstacle Avoidance Sensor

Table 3 Arduino pins connected to the according components

Recommendations

Based upon the results from the competition and testing, the team came up with several recommendations to improve this project. First off, including and using better sensors would make the robots more accurate and have students gain experience with more expensive and luxurious hardware that is commonly used in the industry. Likewise, allowing students to create their own gripper design or robot design would allow for students to be more innovative and creative. This would allow students to not just have fun with code and sensors, but would add a hint of personality and functionality to their robot. The group also recommends to change the layout of the board every few years. This means either adding more walls onto the board or increasing the total size of each board. This would give students a chance to create different strategies and would actually tempt students to go for the plants as so ruin the other team's score. Lastly, the group recommends to increase the variety sensors each group is able to have. Although there are endless combinations and strategies, many of them are very similar but changing these would make it more fun and would change the strategies that students use every year.

References

"Arduino Forum - Index." Arduino Forum - Index, forum.arduino.cc/.

"Arduino IR Obstacle Sensor: Tutorial and Manual." Henry's Bench, 6 Feb. 2017, henrysbench.capnfatz.com/henrys-bench/arduino-sensors-and-input/arduino-ir-obstacle-sensor-tutorial-and-manual/.

TranThinh, and Instructables. "PS2 Wire Controller and Arduino (Control LEDs). "Instructables.com, Instructables, 21 Sept. 2017, www.instructables.com/id/PS2-Wire-Controller-and-Arduino-control-LEDs/.

Vishay Semiconductors. TSSP40.., iopscience.iop.org/article/10.1088/1757-899X/149/1/012141/pdf.

Appendices

A. Flowchart

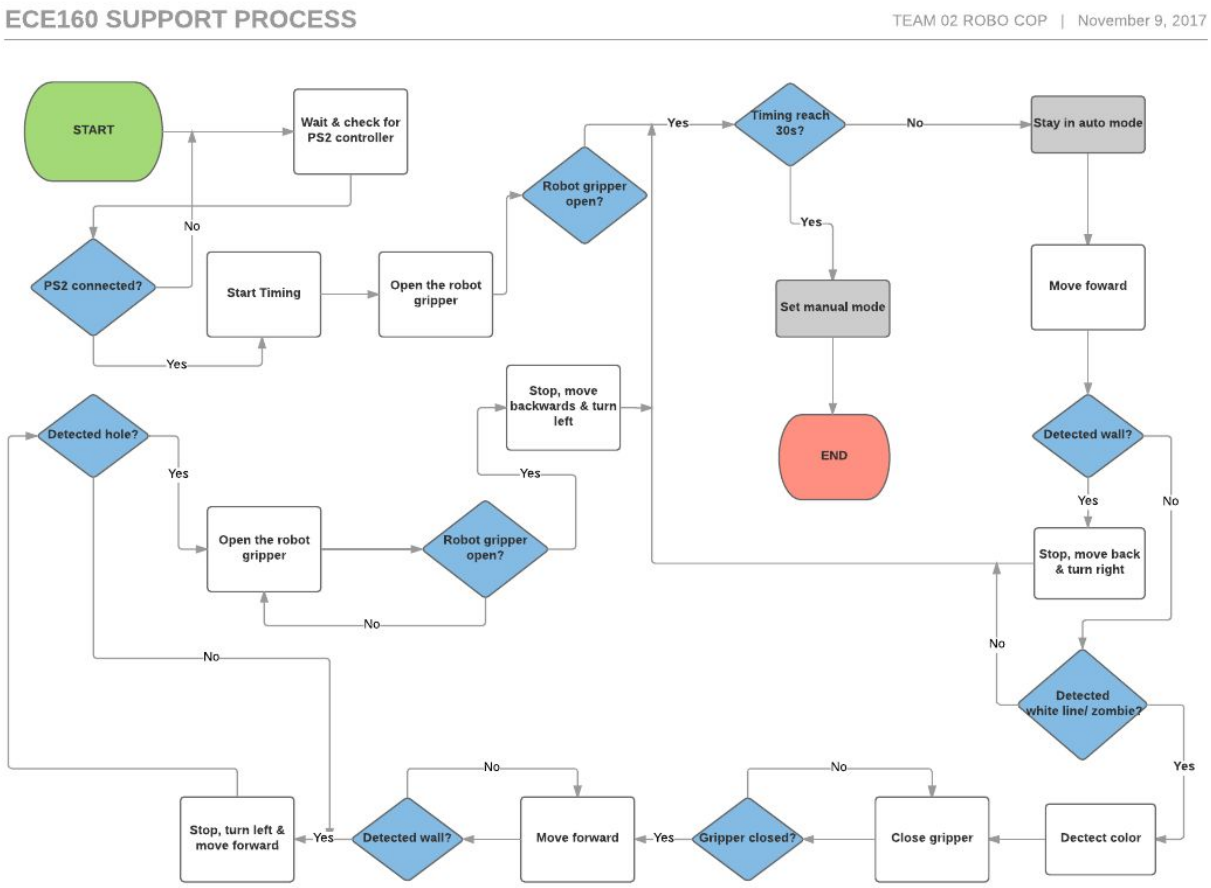

Figure 6: Automatic mode flow chart

B. Project Code

```
#include <Servo.h>       // Include servo library
#include <Timer.h>
#include <PS2X_lib.h>  //for v1.6

Servo servoLeft;         // Declare left servo signal
Servo servoRight;        // Declare right servo signal
Servo servoGrabber;              // Declare rear servo signal
PS2X ps2x;               // create PS2 Controller Class
long starttime;           // Used for timer start
long endtime;             // Used to determine end time
int timedYes = 0;
boolean isGrabberOpen; //determines whether grabber is open
boolean isObjectInside = false; // determines whether an object is in the grabber
boolean isAutonomous; // determines whether the robot is in autonomous mode
int error = 0; // error code for ps2 controller
byte type = 0;//data type for ps2 controller
byte vibrate = 0;//whether vibration motor are on
int count = 0; // counter

int rowCount = 0; // counts rows for autonomous mode
int forwardObsSensor = A5; // front wall detection obsticle sensor connected to pin A5
int grabberSensor = A0; // grabber detection obsticle sensor connected to pin A5

#define pressures   true
#define rumble          false
#define PS2_DAT        12  //ps2 data pin 12
#define PS2_CMD        11  //ps2 command pin 11
#define PS2_SEL        10  //ps2 attention pin 10
#define PS2_CLK        13  //ps2 clock pin 13

void setup()
{
  Serial.begin(9600);

error = ps2x.config_gamepad(PS2_CLK, PS2_CMD, PS2_SEL, PS2_DAT, pressures, rumble);

 if(error == 0)
          Serial.print("Found Controller, configured successful ");

 else if(error == 1)
```

```
  Serial.println("No controller found, check wiring, see readme.txt to enable debug. visit www.billporter.info for troubleshooting tips");

 else if(error == 2)
  Serial.println("Controller found but not accepting commands. see readme.txt to enable debug. Visit www.billporter.info for troubleshooting tips");

else if(error == 3)
        Serial.println("Controller refusing to enter Pressures mode, may not support it. ");


 type = ps2x.readType();
 switch(type)
 {
        case 0:
        Serial.print("Unknown Controller type found ");
        break;
        case 1:
        Serial.print("DualShock Controller found ");
        break;
        case 2:
        Serial.print("GuitarHero Controller found ");
        break;
 case 3:
        Serial.print("Wireless Sony DualShock Controller found ");
        break;
  }
  isAutonomous = false; //robot starts in non autonmous mode
  pinMode(forwardObsSensor, INPUT);//sets pin A5 as input
  pinMode(grabberSensor,INPUT);//sets pin to A0 as inputs
  servoLeft.attach(5); // Attach left signal to P5
  servoRight.attach(3);// Attach right signal to P3
  servoGrabber.attach(2);// Attach left signal to P2

        delay(4000);
        starttime = millis();//sets starttime at 0 and as milliseconds
        endtime = starttime;
        while ((endtime - starttime) <= 1000 * 30) // do this loop for up to 30 sec
        {
        Autonomous();
        endtime = millis();
        }
}
```

```
void loop()
{
  if(error == 1) //skip loop if no controller found
          return;

  if(type == 1)//PS2 Controller
  {
      ps2x.read_gamepad();          //read controller

          int xAxis = ps2x.Analog(PSS_LY); // reads left joystick
          int yAxis = ps2x.Analog(PSS_LX); // reads left joystick
          int zAxis = ps2x.Analog(PSS_RY); // reads right joystick
      Serial.print("x=");
          Serial.print(xAxis);
      Serial.print("          y=");
          Serial.println(yAxis);
            if(xAxis > 220) // move forward
            {
               int xRAxis = map(xAxis,128,255,1500,1700); // maps values for variable speeds
               int xLAxis = map(xAxis,128,255,1500,1300);
               forward(xRAxis, xLAxis);
            }
            if(xAxis < 80) // move backwards
            {
              int xRAxis = map(xAxis,128,0,1500,1300);// maps values for variable speeds
              int xLAxis = map(xAxis,128,0,1500,1700);
              reverse(xRAxis, xLAxis);
            }
           if(yAxis > 200) // turn left
            {
              int yRAxis = map(yAxis,128,255,1500,1700);// maps values for variable speeds
              int yLAxis = map(yAxis,128,255,1500,1700);
              turnRight(yRAxis, yLAxis);
            }
          if(yAxis < 100) //turn right
            {
              int yRAxis = map(yAxis,128,0,1500,1300);// maps values for variable speeds
              int yLAxis = map(yAxis,128,0,1500,1300);
              turnLeft(yRAxis, yLAxis);
            }
          if((xAxis < 200) && (xAxis > 100) && (yAxis < 200) && (yAxis > 100)) // stops robot
              stopServo();
```

```
            if(zAxis > 200) // closed grabber
              grabberClose();
            if(zAxis < 100)// opens grabbers
              grabberOpen();

           ps2x.read_gamepad(); // reads ps2 controller


  }
}


void forward() // move forward full speed
{
  servoLeft.write(1700);
  servoRight.write(1300);
}
void forward(int r, int l) // move forward variable speed
{
  servoLeft.write(l);
  servoRight.write(r);
}

void reverse() // reverse backwards full speed
{
  servoLeft.write(1300);
  servoRight.write(1700);
}
void reverse(int r, int l) // reverse backwards variable speed
{
  servoLeft.write(l);
  servoRight.write(r);
}

void turnRight() // turn right full speed
{
  servoLeft.write(1700);
  servoRight.write(1700);
}
void turnRight(int r, int l) // turn right variable speed
{
  servoLeft.write(l);
```

```
  servoRight.write(r);
}
void turnLeft() // turn left full speed
{
  servoLeft.write(1300);
  servoRight.write(1300);
}
void turnLeft(int r, int l) //turn left variable speed
{
  servoLeft.write(l);
  servoRight.write(r);
}
void grabberOpen() // opens grabber
{
  servoGrabber.write(90);
  isGrabberOpen = true;
}
void grabberClose() //close grabber
{
  servoGrabber.write(0);
  servoGrabber.write(180);
  isGrabberOpen = false;
}

void stopServo() // stop all servos
{
  servoLeft.write(1500);
  servoRight.write(1500);
}

void objectInside()
{
  int forwardObsValue = analogRead(forwardObsSensor); // reads wall sensor
  while((forwardObsValue > 300)&& (isObjectInside)) // while wall sensor detects no wall and while
there is an object in the grabber run this code
  {
   forward();
   forwardObsValue = analogRead(forwardObsSensor); // reads wall sensor
   if((forwardObsValue <= 50)) // if there is a wall
          {
          if(count == 1)// && (rowCount < 2)) // if this is the second wall and first row use this code
          {
            stopServo();
```

```
            delay(1000);
            grabberOpen();
            delay(1000);
            count = 0;
            rowCount ++;
            if(rowCount >= 2) // if this is the second row run this code
            {
              reverse();
              delay(3600);
              turnLeft();
              delay(720);
              count = 0;
              isObjectInside =  false;
            }
            else // if only first wall is detected do this
            {
              reverse();
              delay(5000);
              turnLeft();
              delay(720);
              isObjectInside = false;
            }
          }
          else // if nothing detected by sensor do this
          {
            stopServo();
            delay(1000);
            turnLeft();
            delay(520);
            count++;
            forwardObsValue = analogRead(forwardObsSensor); // read front wall sensor
          }
         }
  }
}

void Autonomous()
{
        int forwardObsValue = analogRead(forwardObsSensor);// reads wall sensors
        int grabberSensorValue = analogRead(grabberSensor);// reads grabber sensor
      ps2x.read_gamepad();          //read controller
```

```
      if((forwardObsValue > 300)) // move forward if everything is clear
        {
       forward();
       forwardObsValue = analogRead(forwardObsSensor);
       grabberSensorValue = analogRead(grabberSensor);
       //Serial.println(lineSensorValue);
        }

      if((grabberSensorValue <=50)) //if something is in the grabber
           {
        forward();
        delay(100);
        grabberSensorValue = analogRead(grabberSensor);
        if(grabberSensor<=50) // if there is still an object in grabber then close grabber
        {
          stopServo();
          delay(1000);
          grabberClose(); // close grabber
          delay(1000);
          turnRight(); // turn around
          delay(1100);
          stopServo();
          isObjectInside = true; // there is an object there
          objectInside(); // runs code on what to do with the object
        }
       }

        forwardObsValue = analogRead(forwardObsSensor); //reads wall sensor
        ps2x.read_gamepad();          //read controller


}
```

Sent from Mail for Windows 10

```
#include <Servo.h>       // Include servo library
#include <Timer.h>
#include <PS2X_lib.h>  //for v1.6

Servo servoLeft;         // Declare left servo signal
Servo servoRight;        // Declare right servo signal
```

```
Servo servoGrabber;             // Declare rear servo signal
PS2X ps2x;              // create PS2 Controller Class
long starttime;           // Used for timer start
long endtime;             // Used to determine end time
int timedYes = 0;
boolean isGrabberOpen; //determines whether grabber is open
boolean isObjectInside = false; // determines whether an object is in the grabber
boolean isAutonomous; // determines whether the robot is in autonomous mode
int error = 0; // error code for ps2 controller
byte type = 0;//data type for ps2 controller
byte vibrate = 0;//whether vibration motor are on
int count = 0; // counter

int rowCount = 0; // counts rows for autonomous mode
int forwardObsSensor = A5; // front wall detection obsticle sensor connected to pin A5
int grabberSensor = A0; // grabber detection obsticle sensor connected to pin A5

#define pressures   true
#define rumble          false
#define PS2_DAT        12  //ps2 data pin 12
#define PS2_CMD        11  //ps2 command pin 11
#define PS2_SEL        10  //ps2 16
#define PS2_CLK        13  //ps2 clock pin 13

void setup()
{
  Serial.begin(9600);

error = ps2x.config_gamepad(PS2_CLK, PS2_CMD, PS2_SEL, PS2_DAT, pressures, rumble);

 if(error == 0)
          Serial.print("Found Controller, configured successful ");

 else if(error == 1)
   Serial.println("No controller found, check wiring, see readme.txt to enable debug. visit
www.billporter.info for troubleshooting tips");

 else if(error == 2)
   Serial.println("Controller found but not accepting commands. see readme.txt to enable debug.
Visit www.billporter.info for troubleshooting tips");

else if(error == 3)
          Serial.println("Controller refusing to enter Pressures mode, may not support it. ");
```

```
 type = ps2x.readType();
 switch(type)
 {
        case 0:
        Serial.print("Unknown Controller type found ");
        break;
        case 1:
        Serial.print("DualShock Controller found ");
        break;
        case 2:
        Serial.print("GuitarHero Controller found ");
        break;
 case 3:
        Serial.print("Wireless Sony DualShock Controller found ");
        break;
  }
  isAutonomous = false; //robot starts in non autonmous mode
  pinMode(forwardObsSensor, INPUT);//sets pin A5 as input
  pinMode(grabberSensor,INPUT);//sets pin to A0 as inputs
  servoLeft.attach(5); // Attach left signal to P5
  servoRight.attach(3);// Attach right signal to P3
  servoGrabber.attach(2);// Attach left signal to P2

        delay(4000);
        starttime = millis();//sets starttime at 0 and as milliseconds
        endtime = starttime;
        while ((endtime - starttime) <= 1000 * 30) // do this loop for up to 30 sec
        {
        Autonomous();
        endtime = millis();
        }
}

void loop()
{
 if(error == 1) //skip loop if no controller found
        return;

 if(type == 1)//PS2 Controller
 {
     ps2x.read_gamepad();          //read controller
```

```
    int xAxis = ps2x.Analog(PSS_LY); // reads left joystick
    int yAxis = ps2x.Analog(PSS_LX); // reads left joystick
    int zAxis = ps2x.Analog(PSS_RY); // reads right joystick
Serial.print("x=");
    Serial.print(xAxis);
Serial.print("          y=");
    Serial.println(yAxis);
      if(xAxis > 220) // move forward
      {
         int xRAxis = map(xAxis,128,255,1500,1700); // maps values for variable speeds
         int xLAxis = map(xAxis,128,255,1500,1300);
         forward(xRAxis, xLAxis);
      }
      if(xAxis < 80) // move backwards
      {
        int xRAxis = map(xAxis,128,0,1500,1300);// maps values for variable speeds
        int xLAxis = map(xAxis,128,0,1500,1700);
        reverse(xRAxis, xLAxis);
      }
     if(yAxis > 200) // turn left
      {
        int yRAxis = map(yAxis,128,255,1500,1700);// maps values for variable speeds
        int yLAxis = map(yAxis,128,255,1500,1700);
        turnRight(yRAxis, yLAxis);
      }
    if(yAxis < 100) //turn right
      {
        int yRAxis = map(yAxis,128,0,1500,1300);// maps values for variable speeds
        int yLAxis = map(yAxis,128,0,1500,1300);
        turnLeft(yRAxis, yLAxis);
      }
    if((xAxis < 200) && (xAxis > 100) && (yAxis < 200) && (yAxis > 100)) // stops robot
        stopServo();

        if(zAxis > 200) // closed grabber
          grabberClose();
        if(zAxis < 100)// opens grabbers
          grabberOpen();

       ps2x.read_gamepad(); // reads ps2 controller
```

```
  }
}


void forward() // move forward full speed
{
  servoLeft.write(1700);
  servoRight.write(1300);
}
void forward(int r, int l) // move forward variable speed
{
  servoLeft.write(l);
  servoRight.write(r);
}

void reverse() // reverse backwards full speed
{
  servoLeft.write(1300);
  servoRight.write(1700);
}
void reverse(int r, int l) // reverse backwards variable speed
{
  servoLeft.write(l);
  servoRight.write(r);
}

void turnRight() // turn right full speed
{
  servoLeft.write(1700);
  servoRight.write(1700);
}
void turnRight(int r, int l) // turn right variable speed
{
  servoLeft.write(l);
  servoRight.write(r);
}
void turnLeft() // turn left full speed
{
  servoLeft.write(1300);
  servoRight.write(1300);
}
void turnLeft(int r, int l) //turn left variable speed
{
```

```
  servoLeft.write(l);
  servoRight.write(r);
}
void grabberOpen() // opens grabber
{
  servoGrabber.write(90);
  isGrabberOpen = true;
}
void grabberClose() //close grabber
{
  servoGrabber.write(0);
  servoGrabber.write(180);
  isGrabberOpen = false;
}

void stopServo() // stop all servos
{
  servoLeft.write(1500);
  servoRight.write(1500);
}

void objectInside()
{
  int forwardObsValue = analogRead(forwardObsSensor); // reads wall sensor
  while((forwardObsValue > 300)&& (isObjectInside)) // while wall sensor detects no wall and while
there is an object in the grabber run this code
  {
   forward();
   forwardObsValue = analogRead(forwardObsSensor); // reads wall sensor
   if((forwardObsValue <= 50)) // if there is a wall
          {
          if(count == 1)// && (rowCount < 2)) // if this is the second wall and first row use this code
          {
            stopServo();
            delay(1000);
            grabberOpen();
            delay(1000);
            count = 0;
            rowCount ++;
            if(rowCount >= 2) // if this is the second row run this code
            {
              reverse();
              delay(3600);
```

```
            turnLeft();
            delay(720);
            count = 0;
            isObjectInside =  false;
           }
           else // if only first wall is detected do this
           {
            reverse();
            delay(5000);
            turnLeft();
            delay(720);
            isObjectInside = false;
           }
          }
          else // if nothing detected by sensor do this
          {
           stopServo();
           delay(1000);
           turnLeft();
           delay(520);
           count++;
           forwardObsValue = analogRead(forwardObsSensor); // read front wall sensor
          }
         }
  }
}

void Autonomous()
{
        int forwardObsValue = analogRead(forwardObsSensor);// reads wall sensors
        int grabberSensorValue = analogRead(grabberSensor);// reads grabber sensor
      ps2x.read_gamepad();          //read controller


      if((forwardObsValue > 300)) // move forward if everything is clear
        {
       forward();
       forwardObsValue = analogRead(forwardObsSensor);
       grabberSensorValue = analogRead(grabberSensor);
       //Serial.println(lineSensorValue);
        }

      if((grabberSensorValue <=50)) //if something is in the grabber
```

```
        {
      forward();
      delay(100);
      grabberSensorValue = analogRead(grabberSensor);
      if(grabberSensor<=50) // if there is still an object in grabber then close grabber
      {
        stopServo();
        delay(1000);
        grabberClose(); // close grabber
        delay(1000);
        turnRight(); // turn around
        delay(1100);
        stopServo();
        isObjectInside = true; // there is an object there
        objectInside(); // runs code on what to do with the object
      }
    }

      forwardObsValue = analogRead(forwardObsSensor); //reads wall sensor
      ps2x.read_gamepad();          //read controller


}
```