ILR #1: Sensor and Motors Lab

Harry Golash – Team A

October 14, 2016

Teammates: Amit Agarwal, Yihao Qian, Menghan Zhang, Zihao Zhang

1. Individual Progress:

1.1 Tasks:

Our team had selected an Arduino Uno R3 as our microcontroller for this lab. Since we are a team of five, we had to use 4 sensors of our choice connected to the 3 motors provided (a servo, a stepper, and a DC brushed motor with an attached encoder).

Since I believe that a potentiometer does not qualify as a sensor, I decided to use an additional fifth sensor, namely the ultrasonic ping sensor, as part of our lab. I later decided to have this sensor output to a buzzer with a beeping frequency inversely proportional to the distance sensed. My team was more than happy with my decision to do both these tasks in addition to doing the GUI and combining their code.

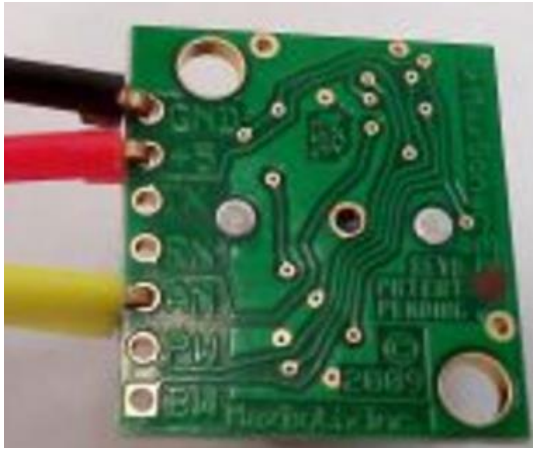
The tasks undertaken by me were:

1. Interface an ultrasonic distance sensor with Arduino to detect distances in inches
2. Interface a buzzer with Arduino to beep at increased frequencies at closer distances
3. Combine teammates' code and circuits (Amit did most of this)
4. Create a GUI in C# to interface with the Arduino and the user

1.2 Implementation:

1.2.1 Ultrasonic distance sensor:

I connected the ultrasonic ping sensor (MaxBotix EZ1) to the Arduino by connecting the ground (GND) and power (VCC) pins to the GND and the +5V pins on the Arduino, respectively. The AN pin of the sensor was connected to the analog pin A4 on the Arduino (see right image below) and voltages were read directly. Distance values in inches were calculated by calibration with a measuring tape, and by using an averaging method to reduce noise in the reading.



Pictures taken from manufacturer's website: http://www.maxbotix.com/Ultrasonic_Sensors/MB1010.htm

1.2.2 Buzzer output:

I connected one of the two-pin buzzers available in the lab to the Arduino by connecting one of the buzzer pins to the GND pin and the other to digital pin 13 on the Arduino. All connections to the Arduino were made using a solderless breadboard. The Tone(pin, int) function was used at a note I found sufficiently annoying to make the buzzer beep. Beeping was done at an increased frequency the closer an object got to the sensor. At distances below 15 inches, the beeping was a continuous tone. At distances over 40 inches, the buzzer was silenced. These parameters were chosen for demo purposes to simulate the proximity sensors on a car.

1.2.3 Combine code and circuits:

Amit did most of the work combining the circuits and the code, and I helped with this later on. I had to configure the code and the circuit to behave with my GUI. As mentioned previously, a solderless breadboard was used to combine the circuitry for all the sensors and motors of this lab with the inputs and outputs of an Arduino Uno microcontroller.

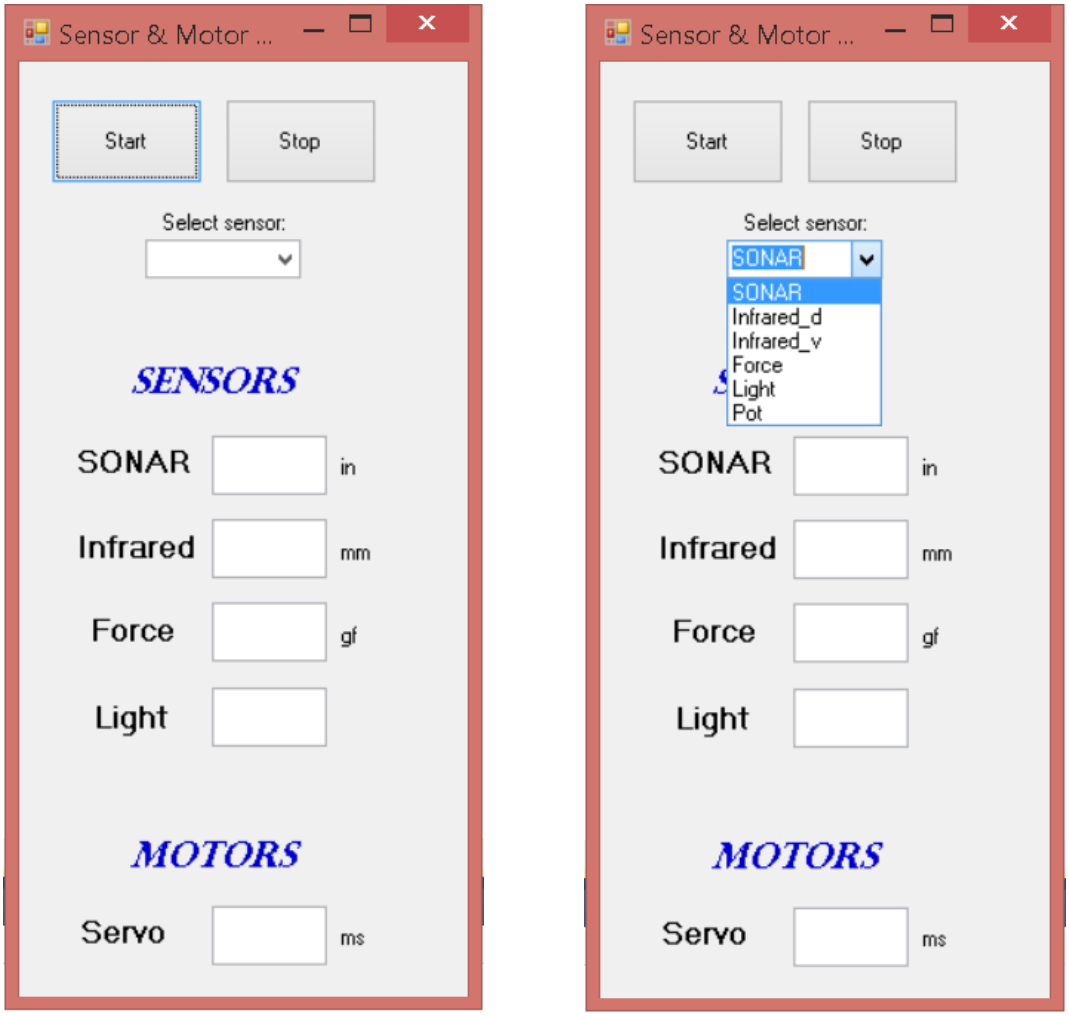
1.2.4 Graphical User Interface (GUI):

One member of each team was required to make a GUI to interface with the Arduino board via serial commands. As none of my teammates had any experience making GUIs or working with serial commands, I offered to do this part.

I chose to program my GUI in C# using Visual Studio 2010. I came up with the following simple GUI that could change the system state and read the five sensors and the servo motor position (see image on next page).

Each system state is tied to a serial integer byte that is received on the Arduino via the serial communication from the C# code and GUI. The dropdown menu on the GUI allows the user to change the state of system by selecting the appropriate choice. Each choice in the dropdown menu sends a serial character that identifies the state to the Arduino as shown in the table below:

Serial character	**Integer byte value**	**State**	
		Sensor	**Output**
f	102	Force sensor	Servo motor
i	105	Infrared rangefinder	DC motor position
j	106	Infrared rangefinder	DC motor velocity
l	108	Light intensity sensor	Stepper motor
p	112	Potentiometer	Servo motor
r	114	Ultrasonic rangefinder	Buzzer



All the code has been uploaded to Blackboard and is also present in the appendices of this report.

The Start button reads the dropdown menu selection made by the user and passes the appropriate character to the Arduino via serial communication. It then reads the serial output from the Arduino and outputs it to the appropriate sensor display box continuously (as long as the Arduino keeps sending data) until the Stop button is clicked. Clicking the Stop button pauses the output display and holds the last displayed value. This is useful if the user wants to capture values at a given configuration or time period. Upon clicking the Start button, the readings are displayed again for the last used sensor. Stop pauses the display for all sensors, so the user can hold the reading from one or more sensors while the GUI is actively displaying the output of a different sensor selected from the dropdown menu.

The GUI has a refresh rate of 10 Hz, which we found to be suitable for our purposes. It allowed us to reduce the noise from our sensors by averaging the readings while still displaying readings to the user in basically real-time. The GUI performed as it was supposed to. Given more time, a more sophisticated GUI could be developed using the same principles used in this one.

2. Challenges:

2.1 Interfacing between GUI and Arduino:

One of the issues I initially had was reading the sensor data sent by the Arduino via the serial port. Clicking buttons and having them pass a character to the board via serial was fairly easy; however, getting feedback from the Arduino in a loop following that proved difficult. I later found out that this was caused due to a cross-threading error that I had failed to recognize.

In order to display the data to the user, the write and read operations as well as the opening and closing of the serial port must take place in the same thread. To make this work, I used the multi-threading library and used two threads to make the GUI software. One thread monitored the user input - the Start and Stop buttons and the dropdown menu. The other parallel thread passed commands to the Arduino and displayed values in the appropriate boxes. The two threads passed values between each other by use of a delegate. This allowed both threads to be aware of the system state and display data accordingly.

2.2 Scheduling and experience issues:

My teammates all worked hard on this project and everyone completed the parts they were supposed to do before the deadline. However, the final circuitry was completed one hour before the deadline: that is, the circuit was handed to me at just after 2 pm on Thursday (10/13/16). This meant that I had about one hour to combine and edit the code and then create a GUI solution for this lab. Fortunately, I was able to make a functional (albeit fairly simple) GUI for the presentation. In the future, we will work on better planning and task selection and division of labor.

2.3 Ultrasonic rangefinder and buzzer system:

Calibrating this system was so wonderfully annoying – all that beeping drove people mad.

3. Teamwork:

With the exception of the challenge faced above, our team worked harmoniously to complete this lab. We divided the tasks among ourselves and helped each other with various difficulties whenever a team member requested help. Each person chose a sensor and a motor they felt like they would like to work with; since there were 3 motors, Yihao and Menghan both used the servo motor with their sensors, and used a physical button to switch between sensor modes.

Amit chose the tasks of working on the DC motor coupled with IR rangefinder sensor. He also volunteered to combine the circuits and the code. Zihao worked on the stepper motor coupled with a light intensity sensor. Yihao used a force sensor and Menghan used a potentiometer. They shared the servo motor. Since there were no motors left I used a buzzer as the output of my sensor selection. I chose to use an ultrasonic rangefinder because a potentiometer is really not a sensor.

4. Future Plans:

Given the challenges we faced with our planning for this lab, we are going to work on allocating tasks more appropriately and putting in extra hours where and if necessary. As team manager, I will work on creating a more accommodating and reliable work plan for the team and I will urge my teammates to play to their strengths.

We have received the sensors that we need for our project from our sponsor, Delphi, and we are currently working on a mounting system for the sensors. Amit and I will design, fabricate, and test the mounting rack for the cameras and radar on one of our cars. Meanwhile, Zihao, Menghan, and Yihao will work on benchmarking and acquiring synchronous data from our sensors.

Once we have a reliable mounting method and a method for acquiring synchronized data from all our sensors, Amit and I will work on finding optimum locations for the sensors on the car; we will also work on stereo vision. Zihao, Menghan, and Yihao will do research into methods for object detection. By the end of this semester, we aim to have our sensors on a car and have stereo vision working with basic object detection capabilities. This will allow us to test and improve our code and computing resources in the next semester.

Appendix A: Arduino code for ultrasonic sensor and buzzer

```
#define rangePin A0
#define buzPin 11

unsigned int note = 520;
int beep = 9999;
int count = 0;

bool Force = false;
bool Infrared = false;
bool Light = false;
bool Pot = false;
bool SONAR = false;

void setup() {
 Serial.begin(9600); // Start serial
 pinMode(buzPin, OUTPUT); // Pin for the buzzer
 pinMode(rangePin, INPUT); // Pin for ultrasonic rangefinder
 noTone(buzPin); // Shut up the buzzer initially

 Serial.println("Let's go!"); // Intro message
 tone(buzPin, (note - 100));
 delay(100);
 tone(buzPin, note);
 delay(100);
 tone(buzPin, (note + 100));
 delay(100);
 tone(buzPin, (note + 200));
 delay(100);
 noTone(buzPin);
}

void loop() {

 if (Serial.available() > 0) {
   int incomingByte = Serial.read();
   // f=102 (force sensor_on))

   // i=105 (infrared position)
   // j=106 (infrared velocity)


   // l=108 (light sensor_on)

   // p=112 (potentiometer_on)

   // r=114 (reversing-SONAR sensor_on)

   switch (incomingByte) {
     case 102:
       StopSensors();
       Force = true;
       break;
     case 105:
       StopSensors();
       Infrared = true;
```

```
      break;
    case 108:
      StopSensors();
      Light = true;
      break;
    case 112:
      StopSensors();
      Pot = true;
      break;
    case 114:
      StopSensors();
      SONAR = true;
      break;
    default:
      StopSensors();
      break;
  }
 }

 if (SONAR == true)
   ReverseSonar();
 else
   noTone(buzPin);
}

void StopSensors() {
 Force = false;
 Infrared = false;
 Light = false;
 Pot = false;
 SONAR = false;
}

void ReverseSonar() {
 float sum = 0;
 for (int i = 0; i < 10; i++) {
   float reading = analogRead(rangePin);
   sum += reading * (5 / 1023.0);
   delay(10);
 }

 float distance = sum * 10 + 4.0; // Distance in inches
 Serial.println(distance);

 if (distance <= 15) {
   beep = 0;
 }
 else if (distance <= 15) {
   beep = 2;
 }
 else if (distance <= 20) {
   beep = 5;
 }
 else if (distance <= 25) {
   beep = 7;
 }
 else if (distance <= 30) {
   beep = 10;
 }
```

```
 else if (distance <= 35) {
   beep = 12;
 else if (distance <= 40) {
   beep = 15;
 }
 else if (distance > 50) {
   beep = 999;
 }

 if (count >= beep) {
   tone(buzPin, note);
   count = 0;
 }
 else
   noTone(buzPin);
 if (count > 60)
   count = 0;

 count += 1;
}
```

Appendix B: C# code for the GUI

1. Control.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO.Ports;

namespace SensorMotorLab
{
   public partial class Controller : Form
   {
       #region Variables
       // State variables to turn on/off sensors and associated motors
       private bool Force = false;
       private bool Infrared = false;
       private bool Light = false;
       private bool Pot = false;
       private bool Reversing = false;

       private bool Start = false;
       private bool Stop = false;

       private int check = 0;

       private int State = 0;
       #endregion

       #region Delegates
       // Delegates for threads
       public delegate void Update_Display(string str);
       #endregion

       #region Threads
       private Thread Read_Thread; // Thread to read and display serial
       #endregion

       public Controller()
       {
           InitializeComponent();
       }

       private void Controller_Load(object sender, EventArgs e)
       {
       }

       public void update_gui(string str)
```

```
{
    try
    {
        switch (State)
        {
            case 0:
                this.SONARbox.Text = str.ToString();
                break;
            case 1:
            case 2:
                this.IRbox.Text = str.ToString();
                break;
            case 3:
                this.Forcebox.Text = str.ToString();
                break;
            case 4:
                this.Lightbox.Text = str.ToString();
                break;
            case 5:
                this.Servobox.Text = str.ToString();
                break;
            default:
                break;
        }
    }

    catch
    {
        MessageBox.Show("Error updating the GUI");
        SONARbox.Text = "***";
        return;
    }

    this.Refresh();
}

private void Arduino_Read()
{
    SerialPort ArduinoPort = new SerialPort("COM4");
    ArduinoPort.BaudRate = 9600;

    if (!ArduinoPort.IsOpen) // If ArduinoPort is closed
        ArduinoPort.Open(); // Try to open the port

    while (true)
    {
        if (check == 1)
        {
            switch (State)
            {
                case 0:
                    ArduinoPort.Write("r");
                    break;
                case 1:
                    ArduinoPort.Write("i");
                    break;
                case 2:
                    ArduinoPort.Write("j");
                    break;
                case 3:
```

```
                    ArduinoPort.Write("f");
                    break;
                case 4:
                    ArduinoPort.Write("l");
                    break;
                case 5:
                    ArduinoPort.Write("p");
                    break;
                default:
                    ArduinoPort.Write("S");
                    break;
            }

            Update_Display update_del = new Update_Display(update_gui);

            Invoke(update_del, ArduinoPort.ReadLine());

            Thread.Sleep(100);
        }

        else if (check == 2)
        {
            ArduinoPort.Write("S");
        }

        else if (check == -1)
        {
            break;
        }
    }
}

private void StartBtn_Click(object sender, EventArgs e)
{
    if (check == 0)
    {
        check = 1;

        Read_Thread = new Thread(new ThreadStart(Arduino_Read));

        Read_Thread.Start(); // Start the serial read thread
    }
    try
    {
        if (Start == false)
        {
            Start = true;
            Stop = false;
            check = 1;
            StartBtn.BackColor = Color.LightGreen;
            StopBtn.BackColor = default(Color);
        }
    }
    catch
    {
        MessageBox.Show("There was an error in Start Button!");
    }
}
```

```
        private void StopBtn_Click(object sender, EventArgs e)
        {
            try
            {
                if (Stop == false && check == 1)
                {
                    Stop = true;
                    Start = false;
                    check = 2;
                    StopBtn.BackColor = Color.LightGreen;
                    StartBtn.BackColor = default(Color);
                }
            }
            catch
            {
                MessageBox.Show("There was an error in Stop Button!");
            }
        }


        private void StateBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            switch (StateBox.SelectedIndex)
            {
                case 0:
                    State = 0;
                    break;
                case 1:
                    State = 1;
                    break;
                case 2:
                    State = 2;
                    break;
                case 3:
                    State = 3;
                    break;
                case 4:
                    State = 4;
                    break;
                case 5:
                    State = 5;
                    break;
                default:
                    break;
            }
        }
    }
}
```

Appendix B: C# code for the GUI (contd.)

2. Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace SensorMotorLab
{
   static class Program
   {
       /// <summary>
       /// The main entry point for the application.
       /// </summary>
       [STAThread]
       static void Main()
       {
           Application.EnableVisualStyles();
           Application.SetCompatibleTextRenderingDefault(false);

           Controller temp_control_form = new Controller();
           temp_control_form.FormClosed += new
FormClosedEventHandler(temp_control_form_OnClosed);

           Application.Run(new Controller());
       }

       public static void temp_control_form_OnClosed(Object sender, EventArgs e)
       {
           Application.Exit();
       }
   }
}
```

Appendix C: Final combined Arduino code

```
#include <Encoder.h>

#include <Servo.h>
#include <Stepper.h>
#define step_pin 6 // Definepin 3 as the steps pin
#define dir_pin 4 // Define pin 4 as the direction pin
#define buzPin 13
#define rangePin A4
#define light_pin A2 // Define A2 as the light sensor

Servo myservo;


// harry's code:
unsigned int note = 520;
int beep = 9999;
int count = 0;

bool Force = false;
bool Infrared_d = false;
bool Infrared_v = false;
bool Light = false;
bool Pot = false;
bool SONAR = false;

// create servo object to control a servo
const int numReadings = 10;
int potpin = A0;  // analog pin used to connect the potentiometer
int val;    // variable to read the value from the analog pin
int readings[numReadings];      // the readings from the analog input
int readIndex = 0;              // the index of the current reading
int total = 0;                  // the running total
int average = 0;

int potpinf = 1;
int valf;
int i;
const int numReadingsf = 10;
int readingsf[numReadingsf];      // the readings from the analog input
int readIndexf = 0;              // the index of the current reading
int totalf = 0;                  // the running total
int averagef = 0;

//Stepper
const int stepsPerRevolution = 200;
int lastStep = 0;
int currStep = 0;
Stepper myStepper(stepsPerRevolution, step_pin, dir_pin);

//Position PID
int enable_pin = 11;
int pin_l1 = 7;  //pin l1
int pin_l2 = 8;  //pin l2
int pin_e1 = 2;
```

```
int pin_e2 = 3;
const int irSense = A3;          // Connect sensor to analog pin A3
int dist = 0;
int sample = 0;
double temp = 0;
double error = 0;
double error_last = 0, error_d = 0, error_i = 0;
//PID vars
double pid_in = 0;    //encoder value
double pid_out = 0;   //PWM from 0-255
double pid_set = 0;   //encoder value from IR through mapping
double pid_p = 0.8;
double pid_i = 0.2;
double pid_d = 0.9;
double new_pos_val;
//Encoder vars
Encoder enc_pos(pin_e1, pin_e2);
long pos_val  = -999;
int loop_check = 0;

//Velocity PID
unsigned int encoder0Pos = 0;
unsigned int tmp_Pos = 1;
double vtemp;
double vpid_set = 0;
double ang_vel;
long time_int = 0;
long init_pos = 0;
double vpid_in = 0, vpid_out = 0;
unsigned int vcount = 0;
double verror = 0;
double verror_last = 0, verror_d = 0, verror_i = 0;
double vpid_p = 4.5;
double vpid_i = 0.8;
double vpid_d = 1.5;


//debouncing
int buttonPin = 5;    // the number of the pushbutton pin


// Variables will change:

int buttonState;             // the current reading from the input pin
int lastButtonState = LOW;   // the previous reading from the input pin

// the following variables are unsigned long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
unsigned long lastDebounceTime = 0;  // the last time the output pin was toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if the output flickers
int sysstate = 0;



void setup() {

 Serial.begin(9600);
```

```
 //Stepper
 pinMode(dir_pin, OUTPUT);
 pinMode(step_pin, OUTPUT);
 pinMode(light_pin, INPUT);
 myStepper.setSpeed(200);

 //harry's code
 pinMode(buzPin, OUTPUT); // Pin for the buzzer
 pinMode(rangePin, INPUT); // Pin for ultrasonic rangefinder
 noTone(buzPin); // Shut up the buzzer initially

 myservo.attach(9);  // attaches the servo on pin 9 to the servo object
 pinMode(potpinf, INPUT);
 pinMode(buttonPin, INPUT);
 myservo.attach(9);

 for (int thisReadingf = 0; thisReadingf < numReadingsf; thisReadingf++) {
   readingsf[thisReadingf] = 0;
 }

 for (int thisReading = 0; thisReading < numReadings; thisReading++) {
   readings[thisReading] = 0;
 }
 ang_vel = 0;
 init_pos = 0;

 Serial.println("Let's go!"); // Intro message
 tone(buzPin, (note - 100));
 delay(100);
 tone(buzPin, note);
 delay(100);
 tone(buzPin, (note + 100));
 delay(100);
 tone(buzPin, (note + 200));
 delay(100);
 noTone(buzPin);
}


void loop() {
 int reading = digitalRead(buttonPin);

 // check to see if you just pressed the button
 // (i.e. the input went from LOW to HIGH),  and you've waited
 // long enough since the last press to ignore any noise:

 // If the switch changed, due to noise or pressing:
 if (reading != lastButtonState) {
   // reset the debouncing timer
   lastDebounceTime = millis();
 }

 if ((millis() - lastDebounceTime) > debounceDelay) {
   // whatever the reading is at, it's been there for longer
   // than the debounce delay, so take it as the actual current state:

   // if the button state has changed:
   if (reading != buttonState) {
```

```
      buttonState = reading;

      // only toggle the LED if the new button state is HIGH
      if (buttonState == HIGH) {
        sysstate = (sysstate + 1) % 2;
      }
    }
  }
  lastButtonState = reading;

  if (Serial.available() > 0) {
    int incomingByte = Serial.read();
    // f=102 (force sensor_on))

    // i=105 (infrared position)
    // j=106 (infrared velocity)


    // l=108 (light sensor_on)

    // p=112 (potentiometer_on)

    // r=114 (reversing-SONAR sensor_on)
    switch (incomingByte) {
      case 102:
        StopSensors();
        Force = true;
        break;
      case 105:
        StopSensors();
        Infrared_d = true;
        break;
      case 106:
        StopSensors();
        Infrared_v = true;
        break;
      case 108:
        StopSensors();
        Light = true;
        break;
      case 112:
        StopSensors();
        Pot = true;
        break;
      case 114:
        StopSensors();
        SONAR = true;
        break;
      default:
        StopSensors();
        break;
    }

  }
  delay(2);

  if (SONAR == true)
    ReverseSonar();
```

```
 else
   noTone(buzPin);

 if (Force == true)
   force();

 if (Pot == true)
   portential();

 if (Light == true)
   stepp();

 if (Infrared_d == true)
   pos();

 if (Infrared_v == true)
   vel();
}


void portential()
{

 total = total - readings[readIndex];
 val = analogRead(potpin);            // reads the value of the potentiometer (value between 0
and 1023)

 readings[readIndex] = map(val, 0, 1023, 500, 2400);     // scale it to use it with the servo
(value between 0 and 180)
 // add the reading to the total:
 total = total + readings[readIndex];
 // advance to the next position in the array:
 readIndex = readIndex + 1;
 if (readIndex >= numReadings) {
   // ...wrap around to the beginning:
   readIndex = 0;
   Serial.println(average);
   myservo.writeMicroseconds(average);
 }

 // calculate the average:
 average = total / numReadings;
 // send it to the computer as ASCII digits

 delay(10);                           // waits for the servo to get there
}


void force()
{
 valf = analogRead(potpinf);
 Serial.println(map(valf, 0, 120, 0, 180));
 valf = map(valf, 0, 120, 500, 2400);
 totalf = totalf - readingsf[readIndexf];
 readingsf[readIndexf] = valf;
 totalf = totalf + readingsf[readIndexf];
 readIndexf = readIndexf + 1;
```

```
 if (readIndexf >= numReadingsf) {
   readIndexf = 0;
 }
 averagef = totalf / numReadingsf;
 myservo.writeMicroseconds(averagef);

 delay(2);
}

void stepp() {

 int brightness = analogRead(light_pin); // 0-1023
 Serial.println(brightness);

 brightness = analogRead(light_pin);
 currStep = map(brightness, 0, 1023, 0, 2 * stepsPerRevolution);

 myStepper.step(currStep - lastStep);
 lastStep = currStep;
 delay(100);
}


void pos() {

 //From IR to Encoder units through mapping for pid setpoint

 if (true) {
   temp = map(irRead(), 100, 500, 0, 360);

   if (temp > 179)
     temp = temp - 360;

   delay(10);
   loop_check = 1;
   enc_pos.write(0);
   while (loop_check) {
     pid_set = temp;


    // Serial.print("setpoint (from IR) is ");
    // Serial.println(pid_set);

     new_pos_val = ((int(enc_pos.read())) % 700) * float(360) / 700;


    // Serial.print("Position from encoder = ");
     //Serial.println(new_pos_val);
     //PID stuff

     pid_in = new_pos_val;
     error = pid_set - pid_in;
     error_d = error - error_last;
     error_last = error;
    // Serial.print("Error = ");
    // Serial.println(pid_set - pid_in);
```

```
     if ((error > 5) || (error < -5) && pid_set != 0) {


       error_i = error_i + error;
       pid_out = pid_p * error + pid_i * error_i +  pid_d * error_d;


       if (pid_out < 0) {
         //   low on 2 and high on 1
         digitalWrite(pin_l1, HIGH);
         digitalWrite(pin_l2, LOW);
       } else {
         //   low on 1 and high on 2
         digitalWrite(pin_l1, LOW);
         digitalWrite(pin_l2, HIGH);
       }

       if (pid_out > 255)
         pid_out = 255;
       else if (pid_out < -255)
         pid_out = -255;


       analogWrite(enable_pin, abs(pid_out));
     //  Serial.print("Val from PID (PWM) = ");
     //  Serial.println(pid_out);
     }
     else
     {
       analogWrite(enable_pin, 0);
       digitalWrite(pin_l1, HIGH);
       digitalWrite(pin_l2, HIGH);
       // pid_set = 0;
       error_i = 0;
       error_d = 0;
       error = 0;
       loop_check = 0;
       enc_pos.write(0);
       //        Serial.println("0000000000000000000000000000000000000000");
     //  Serial.println("0000000000000000000000000000000000000000000000000000000000000000000
0000000000000");
       //        Serial.println("0000000000000000000000000000000000000000");
       delay(1000);
     }
   }
 }
}

void vel() {

 //Check each second for change in position
 if (vcount > 100)
 {
   analogWrite(enable_pin, 100);
   vtemp = map(irRead(), 100, 500, 300, 450);
   //    vtemp = Serial.parseInt();
   //    vtemp = vtemp;
   //   verror_i = 0;
   //   verror_d = 0;
```

```
   //    time_int = millis();
   delay(10);
   vcount = 0;
   //Serial.println("Triggered Triggered Triggered Triggered Triggered Triggered Triggered
Triggered Triggered Triggered ");
   analogWrite(enable_pin, map(vtemp, 300, 450, 200, 255));
   //delay(2000);
 }
 else
 {
   vcount++;
   vpid_set = vtemp;
   encoder0Pos =  360 * double(enc_pos.read()) / 700.0;
   tmp_Pos = encoder0Pos;
   //Serial.print("pt = ");
   ang_vel = -1 * float(1000 * (tmp_Pos - init_pos)) / ((millis() - time_int));

   if (ang_vel > 500) {
     ang_vel = 500;
   }
   if (ang_vel < -500) {
     ang_vel = -500;
   }
  // Serial.println(ang_vel);

   time_int = millis();
   init_pos = tmp_Pos;

   //    if((ang_vel <0.6) && (ang_vel >-0.6))
   vpid_in = ang_vel;
   verror = vpid_set - vpid_in;

   if (verror > 50)
     verror_d = verror - verror_last;
   verror_last = verror;
  // Serial.print("vpid_set = ");
  // Serial.println(vpid_set);

  // Serial.print("verror = ");
   //Serial.println(vpid_set - vpid_in);
  // Serial.print("vpid_in = ");
 //  Serial.println(vpid_in);


   //   if(vcount  ==1000) {
   //     verror_i = 5;     //By obversation for 30RPM
   //     vcount = 0;
   //   }
   if (abs(verror) > 50) {
     verror_i = verror_i + verror;
     vpid_out = vpid_p * verror + vpid_i * verror_i +  vpid_d * verror_d;

    // Serial.print("verror_i = ");
//      Serial.println(verror_i);
//
//      Serial.print("verror_d = ");
//      Serial.println(verror_d);
//
//      Serial.print("PID out = ");
```

```
//      Serial.println(vpid_out);
//
//      Serial.print("Enc h = ");
//      Serial.println(360 * double(enc_pos.read()) / 700.0);

     if (vpid_out > 0) {
       //   low on 2 and high on 1
       digitalWrite(pin_l1, HIGH);
       digitalWrite(pin_l2, LOW);
     } else {
       //   low on 1 and high on 2
       digitalWrite(pin_l1, LOW);
       digitalWrite(pin_l2, HIGH);
     }

     if (vpid_out > 255)
       vpid_out = 255;
     else if (vpid_out < -255)
       vpid_out = -255;

     if (vpid_set > 0 && vpid_out < 0) {
       vpid_out = 0;
     }
     if (vpid_set < 0 && vpid_out > 0) {
       vpid_out = 0;
     }

     analogWrite(enable_pin, abs(vpid_out));
//      Serial.print("Val from PID (PWM) = ");
//      Serial.println(vpid_out);
//      //      delay(50);
   }
 }
}

// Take multiple readings, and average them out to reduce false readings
int irRead() {
 int averaging = 0;             //  Holds value to average readings
 // Get a sampling of 10 readings from sensor
 for (int i = 0; i < 10; i++) {
   dist = analogRead(irSense);
   averaging = averaging + dist;
   delay(55);      // Wait 55 ms between each read
   // According to datasheet time between each read
   //  is -38ms +/- 10ms. Waiting 55 ms assures each
   //  read is from a different sample
 }
 dist = averaging / 10;      // Average out readings
 if (dist > 500) {
   dist = 500;
 }
 if (dist< 100) {
   dist = 100;
 }
 //Serial.println("Triggered Triggered Triggered Triggered Triggered Triggered Triggered
Triggered Triggered Triggered ");
 Serial.println(dist);
 return (dist);
}
```

```
void StopSensors() {
 Force = false;
 Infrared_d = false;
 Infrared_v = false;
 Light = false;
 Pot = false;
 SONAR = false;
}

void ReverseSonar() {
 float sum = 0;
 for (int i = 0; i < 10; i++) {
   float reading = analogRead(rangePin);
   sum += reading * (5 / 1023.0);
   delay(10);
 }

 float distance = sum * 10 + 4.0; // Distance in inches
 Serial.println(distance);

 if (distance <= 15) {
   beep = 0;
 }
 else if (distance <= 15) {
   beep = 2;
 }
 else if (distance <= 20) {
   beep = 5;
 }
 else if (distance <= 25) {
   beep = 7;
 }
 else if (distance <= 30) {
   beep = 10;
 }
 else if (distance <= 35) {
   beep = 12;
 }
 else if (distance <= 40) {
   beep = 15;
 }
 else if (distance > 50) {
   beep = 999;
 }

 if (count >= beep) {
   tone(buzPin, note);
   count = 0;
 }
 else
   noTone(buzPin);
 if (count > 60)
   count = 0;

 count += 1;
}
```