

Final Report: Robot-Pet, Human Following Robot

1.1 Aimun Jawed, Physics*

A.Kemal Demir (assistant) Department of Physics,
Bilkent University, Bilkent, 06800, Ankara, Turkey

(Dated: December 27, 2020)

Ultrasonic waves are sound waves of frequency greater than 20kHz. Since the frequency of ultrasound is out of the human audibility range, it can be transmitted and received for distance measurements without causing distress to the people. The distance measurements will help the robot to follow people and avoid obstacles.

1.2 INTRODUCTION

Ultrasonic waves are sound waves with a higher frequency which can be used for object tracking, via the sonar method. This involves using an ultrasound transmitter that transmits an incident wave at an angle. Some of the wave is reflected back and some of it is absorbed by the object. Since the speed of the wave will be the same as the speed of sound in air, using the information about the angle of incidence and the time taken to receive the transmitted signal, the distance to the object can be determined. Our robot will use this principle to detect or ‘see’ objects around. Other methods involve using rangefinders, and infra-red cameras, or tracking using GPS coordinates via blue-tooth in smartphones. We have chosen to work with ultrasonic sensors to reduce the cost and complexity of the project to better understand its principle of working.

PROGRESS

The project was completed according to the Grantt chart included in the proposal.

The following materials were acquired and assembled¹:

- Arduino UNO²
- HC-SR04 ultrasonic distance sensor x1
- Ultrasonic sensor holder x1
- L293D Motor Driver IC x1
- USB cable x1
- Breadboard x1
- DC Motor x2
- Servo Motor x1

- Chassis x1
- Wheels x2
- Battery, Battery Holder x1
- Switch
- Jumper wires
- Soldering iron
- Multi-meter
- Screws, Screw Driver

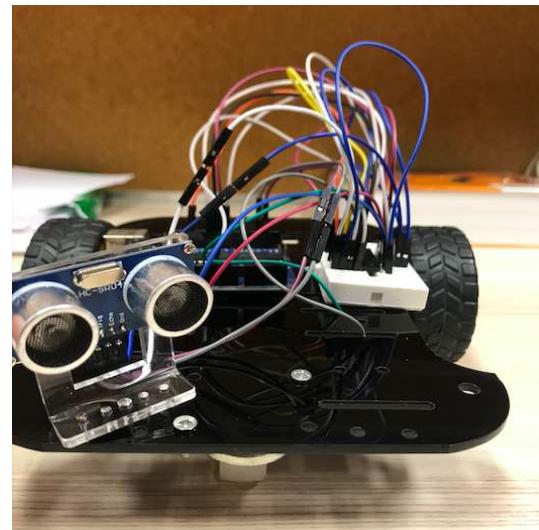


Fig 1: The robo-pet

First, the working of the ultrasonic sensor was tested for accuracy, precision and efficiency using Arduino code. The sensor kept returning only two values, 0cm and 1149cm for distance regardless of any obstacle in front of it. Replacing the sensor with a new one did not resolve the issue, nor did changing the Arduino board help. Finally, a different set of wires were used and this resulted in accurate distance readings, the precision of which was up-to the cm (sufficient for this project). Thus, some wires may be damaged leading to an open-circuit so current can not flow, and so, it is important to test each wire before using it to prevent such an issue from recurring.

Thereafter, all wires had to clear the continuity test - a small voltage is applied between the ends to determine whether current flows or not - before installation in my

*Electronic address: aimun.jawed@ug.bilkent.edu.tr

¹ All materials can be ordered online from robotistan.com

² It is advisable to order more than one HC-SR04 sensor and arduino board because some sensors and boards are faulty, as we learned in our project.

project. To achieve this purpose, I used the multi-meter which has the capability to perform this continuity test - denoted by a diode symbol. Two wires were soldered to each motor, and then connected to the breadboard. I learned how to solder, and use the breadboard in conjunction with the L293D Motor IC via H-bridge, which is slightly more technical than the L293D module.

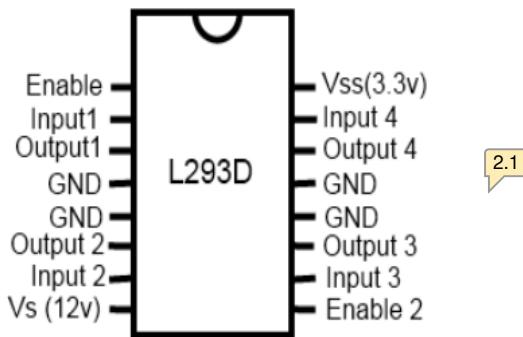


Fig 2: L293D Motor IC Pin Guide [9]

The IC is placed at the gutter of the breadboard with pins inserted on either side. 5V from the Arduino is used to power the VCC of the Ultrasonic sensor and also of the Motor IC at three different points, Enable, VSS and VS, while the GND connections from Arduino are made at all the GND points of the motor IC and the HC-SR04 sensor. Finally, the motor pins are connected to Outputs 1 – 4 and Inputs 1 – 4 are connected to Arduino digital pins 8 – 11 respectively.

Figure 3 shows these connections with the same colour coding as the actual wires to make it easier to track and troubleshoot any issues.

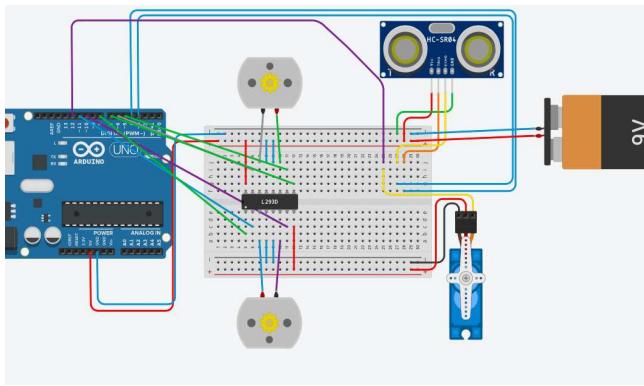


Fig 3: The schematics

After the robot was fully wired, a sample code was programmed in Arduino [1] to test the connections. The code was giving `stk500_ref()` errors about the programmer not responding. This issue is finally resolved by creating a new port for uploading the code, but the issue reappeared after several more runs. It can be resolved in two ways: installing CH403 driver (which is for windows or Linux), or connecting a new Arduino board with a new USB cable, as any one of them could have the problem. A second test code [8] was used to test the working of the motors.

Once we have tested all components individually, we combine the test codes into one and send output to serial monitor to check if the robot is 'seeing' properly. We can now add the conditional statements in the code and simultaneously check the serial monitor. For any distance value less than 5cm, the motors must not receive any power. For distances between 5 and 20, both motors should rotate clockwise, thereby following the target object in front of it.

PRINCIPLE OF WORKING

The ultrasound sensor is connected to the microcontroller which uses electrical signal to trigger the sensor to send 8 acoustic wave bursts and starts a time counter until the signal(echo) is reflected back from any surrounding objects and detected by the sensor [1]. The sensor then outputs a signal with the same time difference as that between the trigger and the echo. The duty cycle of HC-SR04 is $10\mu s$. [6]

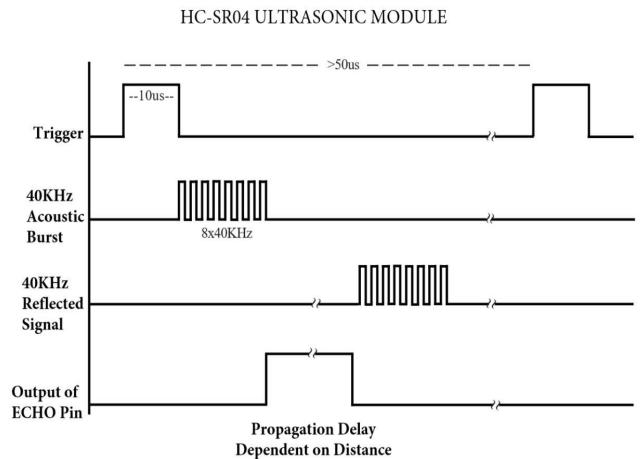


Fig 4: Representation of trigger signal, acoustic bursts, reflected signal and output of echo pin. (Source: HC-SR04 User Guide [4])

$$\text{Distance(cm)} = \frac{\text{echo signal width}(\mu\text{s})}{58}$$

The signal travels the distance, l twice due to the reflection, and therefore, can be calculated by:

$$l = \frac{\text{time taken} * \text{speed}}{2}$$

where speed of the wave = $343ms^{-1}$ at $20^\circ C$

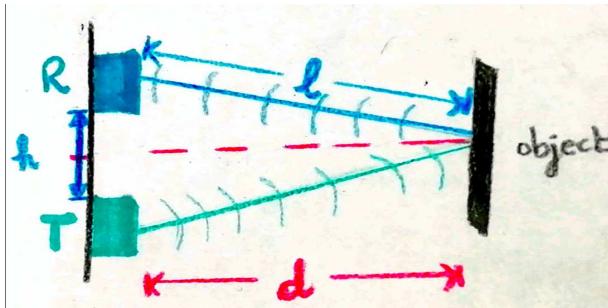


Fig 5: Top view: How the sensor measures distances.
Recreated [6]

Figure 5 shows the distances, where h is the distance between the centre of the transmitter and the receiver which is around $0.03m$ for the HC-SR04 sensor [9]. d can be calculated using [4]:

$$d = \sqrt{l^2 - (\frac{h}{2})^2}$$

These calculations are automatically done by the micro-controller and were printed on screen to check their accuracy.

The robot is to be programmed to detect and follow any object which satisfies $5 < d < 20$, and will avoid any obstacle for which $d < 5$.

METHODOLOGY

Phase I

In the first phase, we connected the HC-SR04 sensor to the Arduino. The sensor has a transmitter, labelled T and the receiver, labelled R. The sensor has 4 pins. The first and last one will be connected to 5V and GND on the Arduino UNO. The middle two pins on the sensor are Trigger and Echo, the former sends the signal and the latter receives the echo. These are connected to pin 6 and 5 respectively.

Phase II

The body of the robot is designed on a chassis which can hold the Arduino, breadboard and the battery, while the wheels and motors are attached on the right and left side, with a ball caster for balance and ease with motion at the front.

Phase III

The ultrasound sensor is mounted at the front of our robot. The height of the sensor is not important for the purpose of this project, but a greater height can ease the process of guiding the robot. The sensor is tested for

accuracy by taking several distance measurements using built-in Arduino libraries for obstacle detection. These were compared against distances measured using a precise metre rule and it was determined that there was no error percentage so the sensor is fairly accurate for the scope of this experiment. Table I shows the measurements printed by the sensor on the serial monitor compared to actual distance values, giving an average error percentage of 0%. Figure 6a shows how the serial plotter displays the data graphically. The spatial arrangement can be seen in Figure 6b.

Average Error:

$$= \frac{(4 - 4) + (6 - 6) + (8 - 8) + (13 - 13) + (21 - 21)}{5} \\ = 0$$

TABLE I: Distance Values and Error Percentage

Actual Distance (cm)	Arduino Reading (cm)	Difference in distance (cm)
4	4	0
6	6	0
8	8	0
13	13	0
15	15	0
21	21	0

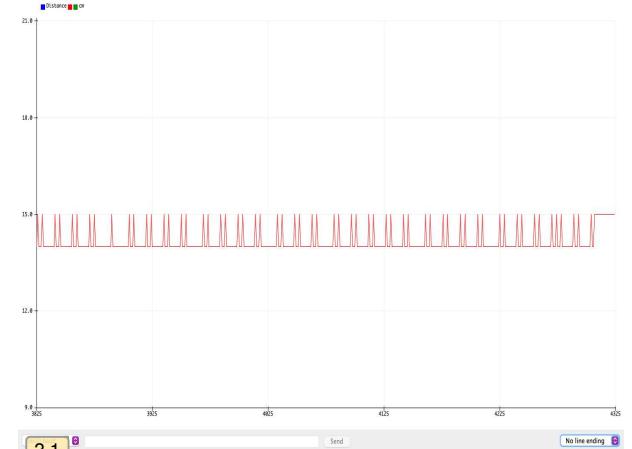


Fig 6a: Arduino serial plotter shows the distance of a wall in front of the robot



Fig 6b: Spatial arrangement for the data in Figure 6a

Phase IV

Jumper wires were soldered to the motors and all other components were arranged on the chassis so that the wires do not interfere with the robot's movement. Finally, the robot was programmed using Arduino.

INDEPENDENT VARIABLES

The distance to objects around is the independent variable. The variation of this distance as the robot traverses around will trigger the response by the motors.

DEPENDENT VARIABLES

The speed of the robot's wheels will vary depending on distance output from the sensor.

ANALYSIS OF RESULTS

The results from the transducer were used to form graphics on the computer similar to a radar to show the presence of the obstacle, enabling us to visualise what the robot 'sees'. The data from the ultrasonic sensor was sent to the Serial Monitor of Arduino. These distances were then sent to a Processing IDE that uses Java to display it graphically³. Due to some problems with the code in Java, the graphics did not come out as expected. We then, used Python to see the results as shown in Fig 7b and Fig 8b. The actual space is shown in Fig 6a and Fig 7a, respectively. The objects are both placed at a distance of 80 cm from the robot's 'eyes', and almost perpendicular to it.



Fig 7a: Spatial arrangement for an object with small diameter

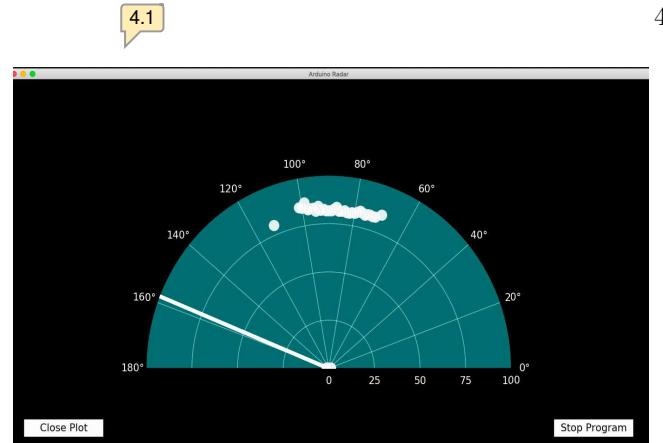


Fig 7b: Radar emulator for object in Figure 7a [11]



Fig 8a: Spatial arrangement for an object with large diameter

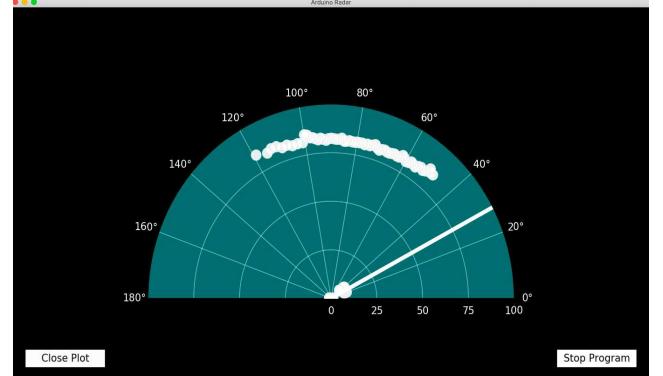


Fig 8b: Radar emulator for object in Figure 8a [11]

Here, we see that the robot returns slightly different distance values if the object is cylindrical. This is not an error. In fact, the values should be different, depending on the angle the line of the ultrasound ray makes with the curved surface of the object. We also observe that objects with a very large diameter, as shown in figure 7a, will be treated as separate objects by the robot, instead of one big obstacle. Again, this is not a problem because the robot is still making accurate detection of the presence of the obstacle.

We adjusted the code with respect to the input which could be the obstacle or the target, to increase or decrease the output (motor power, or robot speed) accordingly. The values defined for target to be followed were $5 < d < 20$, and for obstacle, $d < 5$. These values can

³ To allow serial port communication between Arduino and Processing IDE, ensure that the serial monitor window of arduino is closed otherwise the port will appear as busy.

be changed depending on the performance of the robot and the ease with which it can be guided to follow. For example, keeping one's hand too close will cause it to reverse its wheels and change direction, but keeping hand in 'safe distance', enables it to follow like a pet. If there is no obstacle, nor any object in front of it, it will stop its motion.

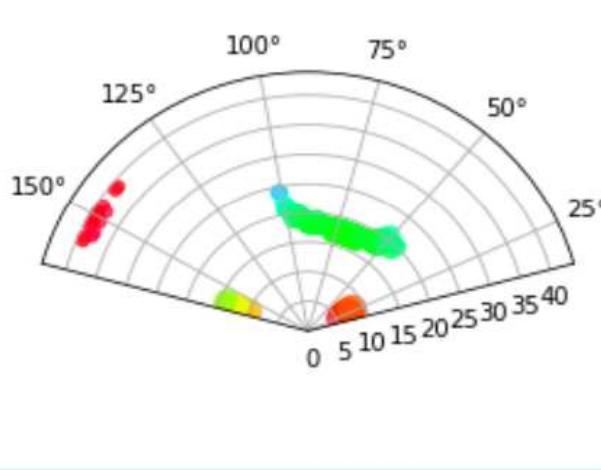


Fig 9: Polar plot showing objects at different angles and distances to the robot ⁴

Figure 9 is plotted in Python using data from Arduino's Serial Monitor. It graphically displays how the robot will make its decision (according to the algorithm we have described above). The green spots represent the objects that the robot interprets as 'safe to follow'. The orange and yellow ones are obstacles that the robot will avoid, and the red spots show the objects that are so far that the robot will ignore.

Error Analysis

Temperature might cause erroneous distance readings. The difference in actual value and the measured value was studied in Table I. Since, the average error was 0%, we can conclude that the sensor gave accurate distance measurement for the scope of this project. The serial monitor gave different distance readings for the same obstacle sometimes which had more to do with the type of surroundings, and the material that the object was made up of, than the temperature. So it was concluded that temperature need not be controlled for our project. It was also observed that, while the robot recorded accurate distances for objects far away (that is, where distance was greater than 3), the robot was unable to detect any distances closer than 3cm, as shown in the serial monitor readings in Fig 10. The spatial arrangement for these readings is shown in Fig 11.

⁴ All distances are in cm

```
08:57:53.024 -> Distance: 3 cm
20:57:53.824 -> Distance: 5 cm
20:57:53.858 -> Distance: 5 cm
20:57:53.882 -> Distance: 5 cm
20:57:53.892 -> Distance: 5 cm
20:57:53.928 -> Distance: 5 cm
20:57:53.928 -> Distance: 5 cm
20:57:53.962 -> Distance: 5 cm
20:57:53.962 -> Distance: 5 cm
20:57:53.999 -> Distance: 5 cm
20:57:53.999 -> Distance: 5 cm
20:57:54.037 -> Distance: 5 cm
20:57:54.037 -> Distance: 4 cm
20:57:55.073 -> Distance: 4 cm
20:57:56.173 -> Distance: 4 cm
20:57:57.311 -> Distance: 4 cm
20:57:58.406 -> Distance: 5 cm
20:57:58.406 -> Distance: 4 cm
20:57:59.499 -> Distance: 3 cm
20:57:59.499 -> Distance: 3 cm
```

Fig 10: What the robot detects when the object is too close, and the object surface is reflective

To deal with this issue, we programmed the robot to treat anything within 5cm as an obstacle. Thus, even though the closest reading possible was 3cm, anything between 0 and less than 5cm will be automatically treated as an obstacle that needs to be avoided, even if its distance value shown on the serial monitor is incorrect. Note, also that, the distance values of the object are fluctuating. This is because the object is made of shiny plastic (Fig 11) ^{5,1}, that is causing reflection, causing the distance readings to fluctuate.

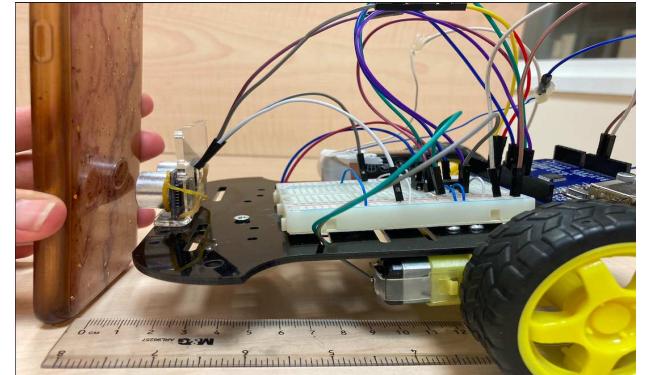


Fig 11: Spatial Arrangement for Fig 10 distance values. Note the reflection on the object.

The power given to the micro-controller affected the efficiency of the motors' response. Therefore, 9V battery is sufficient and better than others. Temperature does not have to be controlled for accurate distance measurements even though speed of the ultrasonic wave will change as temperature drops. But since in this project, a distance measurement accurate to the nearest cm was enough for the robot, we did not have to consider adjusting speed of the wave with respect to temperature changes. False signals from the environment, and material of the object (acoustic impedance) were interfering with the efficiency of this project, as some or most of the sound wave was getting absorbed depending on the material. This can be seen in the radar image, Fig 7b. The dot near 115 degrees

does not actually exist as an obstacle in the real set-up (Fig 7a). The robot probably detected this from delayed angular reflection of the ultrasonic wave from the actual obstacle in consideration. Such minor errors sometimes cause the robot to detect false obstacles and give the appropriate response, such as halting or moving back, away from the obstacle. These errors can be eliminated by using higher frequency waves that will be less affected by the material of the objects around them.

Another issue that arose was the inability of the wheels to turn smoothly, thereby causing the robot to make a turn, where it should be moving straight. This sometimes happened if the Arduino or breadboard were too close to the wheel, and sometimes due to some friction. This can be fixed by using better quality wheels that fix into the motor axle properly to reduce frictional effects.

CONTINGENCY PLAN

Since the project was carried out by a one-member group, having unprecedented lockdown did not have any affect on the project. Some materials were unavailable due to COVID-19 and so they were removed. One of the Arduino boards was shorted, and it was not possible to order another, but a friend lent their board and USB cable, which solved the issue. The project can be made more advanced by including more HC-SR04 and two IR sensors.

CONCLUSION

More sensors can be added to improve its performance and efficiency. As technology is advancing, we can expect robots to become a part of everyone's life in the near future. Such a technology can serve as an aid for the elderly or people in need of special assistance [2]. While it cannot and should not replace actual pets, an advanced version of this robot can be useful for people who would prefer a low-maintenance company.

CODES

Code for Testing HC-SR04 Sensor:

```
#include <NewPing.h>

#define TRIGGER_PIN 3

//And this code will stop motors
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, LOW);
digitalWrite(motorPin4, LOW);
```

```
#define ECHO_PIN      2
#define MAX_DISTANCE 20

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
    Serial.begin(9600);
    delay(50);
}

void loop() {
    Serial.print("Distance is: " );
    Serial.println(sonar.ping_cm());
    delay(1000);
}
```

Code for Testing Motors [8]:

```
//Motor Control -
Motor A: motorPin1,motorpin2
Motor B: motorpin3,motorpin4

//Motor A clockwise for 2s
digitalWrite(motorPin1, HIGH);
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, LOW);
digitalWrite(motorPin4, LOW);
delay(2000);
//Motor A counter-clockwise for 2s
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, HIGH);
digitalWrite(motorPin3, LOW);
digitalWrite(motorPin4, LOW);
delay(2000);

//Motor B clockwise for 2s
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, HIGH);
digitalWrite(motorPin4, LOW);
delay(2000);
//TMotor B counter-clockwise for 2s
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, LOW);
digitalWrite(motorPin4, HIGH);
delay(2000);
```

```
}
```

Code for final run:

```
//L293D
//Motor A
int motorPin1 = 12; // Pin 15 of L293
int motorPin2 = 10; // Pin 10 of L293
//Motor B
int motorPin3 = 11; // Pin 7 of L293
int motorPin4 = 9; // Pin 2 of L293

//Ultrasonic Sensor
//int trigPin = 6;
//int echoPin = 5;

#define echoPin 6
#define trigPin 5

//Define Variables
long duration;
int distance;

//This will run only one time.
void setup(){

    //Set pins as outputs
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
    pinMode(motorPin3, OUTPUT);
    pinMode(motorPin4, OUTPUT);

    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    Serial.begin(9600);
}

void loop(){
// digitalWrite(trigPin , HIGH);
// delayMicroseconds(1000);
// digitalWrite(trigPin , LOW);
//
// duration = pulseIn(echoPin , HIGH);
// distance = (duration/2) / 28.5;

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    // Sets the trigPin HIGH for 10^-6s
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    // Reads the echoPin,
    // returns the sound wave travel time in 10^-6s
    duration = pulseIn(echoPin, HIGH);
    // Calculating the distance
    distance = duration * 0.034 / 2;
    //Speed of sound wave
    divided by 2 (go and back)
}
```

```

// Displays the distance on the Serial Monitor
Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" cm");

if(distance < 5)
{
    digitalWrite(motorPin1, LOW);
    digitalWrite(motorPin2, HIGH);
    digitalWrite(motorPin3, LOW);
    digitalWrite(motorPin4, HIGH);
    delay(500);
    digitalWrite(motorPin1, HIGH);
    digitalWrite(motorPin2, LOW);
    digitalWrite(motorPin3, LOW);
    digitalWrite(motorPin4, HIGH);
}

if(distance > 4  && distance < 20)
{
    digitalWrite(motorPin1, HIGH);
    digitalWrite(motorPin2, LOW);
    digitalWrite(motorPin3, HIGH);
    digitalWrite(motorPin4, LOW);
}

if(distance > 20)
{
    digitalWrite(motorPin1, LOW);
    digitalWrite(motorPin2, LOW);
    digitalWrite(motorPin3, LOW);
    digitalWrite(motorPin4, LOW);
}
}

```

Code for Radar Imaging (Arduino) [11]:

```

#include <Servo.h>

Servo servo_1;
//servo controller(multiple can exist)

int trig = 5; // trig pin for HC-SR04
int echo = 6; // echo pin for HC-SR04
int servo_pin = 12; // PWM pin for servo control

int pos = 0;    // servo starting position
float duration,distance;

void setup() {
  Serial.begin(115200);
  Serial.println("Radar Start");
  servo_1.attach(servo_pin);
  // start servo control
}

```

```

pinMode(trig,OUTPUT);
pinMode(echo,INPUT);
}

void loop() {
  for (pos = 15; pos <= 165; pos += 1) {
    //goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    servo_1.write(pos);
    // tell servo to go to position in variable
    //'pos'
    delay(60);
    // delay to allow the servo to reach the
    //desired position
    dist_calc(pos);
  }

  for (pos = 165; pos >= 15; pos -= 1) {
    // goes from 180 degrees to 0 degrees
    servo_1.write(pos);
    // tell servo to go to position in variable
    //'pos'
    delay(60);
    dist_calc(pos);
  }
}

float dist_calc(int pos){
  // trigger 40kHz pulse for ranging
  digitalWrite(trig,LOW);
  delayMicroseconds(2);
  digitalWrite(trig,HIGH);
  delayMicroseconds(10);
  digitalWrite(trig,LOW);
  // convert from duration for pulse to reach
  //detector (microseconds) to range (in cm)
  duration = pulseIn(echo,HIGH);
  // duration for pulse to reach detector
  //(in microseconds)
  distance = 100.0*(343.0*(duration/2.0))/1000000.0;
  // 100.0*(speed of sound*duration/2)/
  microsec conversion

  Serial.print(pos); // position of servo motor
  Serial.print(","); // comma separate variables
  Serial.println(distance); // print distance in cm
}

```

Code for Radar Imaging (Python) [11]:

```

#using matplotlib
# ** Works with any motor that outputs angular rotation
# ** and with any distance sensor (HC-SR04, VL53L0x,LIDAR)
#
import numpy as np

```

```

import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from matplotlib.widgets import Button
import serial,sys,glob
import serial.tools.list_ports as COMs
#
#
#####
# Find Arudino ports, select one, then start communication with it
#####
#
def port_search():
    if sys.platform.startswith('win'): # Windows
        ports = ['COM{0:1.0f}'.format(ii) for ii in range(1,256)]
    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        ports = glob.glob('/dev/tty[A-Za-z]*')
    elif sys.platform.startswith('darwin'): # MAC
        ports = glob.glob('/dev/tty.*')
    else:
        raise EnvironmentError('Machine Not pyserial Compatible')

    arduinos = []
    for port in ports: # loop through to determine if accessible
        if len(port.split('Bluetooth'))>1:
            continue
        try:
            ser = serial.Serial(port)
            ser.close()
            arduinos.append(port) # if we can open it, consider it an arduino
        except (OSError, serial.SerialException):
            pass
    return arduinos

arduino_ports = port_search()
ser = serial.Serial(arduino_ports[0],baudrate=115200) # match baud on Arduino
ser.flush() # clear the port
#
#####
# Start the interactive plotting tool and
# plot 180 degrees with dummy data to start
#####
#
fig = plt.figure(facecolor='k')
win = fig.canvas.manager.window # figure window
screen_res = win.wm_maxsize() # used for window formatting later
dpi = 150.0 # figure resolution
fig.set_dpi(dpi) # set figure resolution

# polar plot attributes and initial conditions
ax = fig.add_subplot(111,polar=True,facecolor='#006d70')
ax.set_position([-0.05,-0.05,1.1,1.05])
r_max = 100.0 # can change this based on range of sensor
ax.set_ylim([0.0,r_max]) # range of distances to show
ax.set_xlim([0.0,np.pi]) # limited by the servo span (0-180 deg)
ax.tick_params(axis='both',colors='w')
ax.grid(color='w',alpha=0.5) # grid color
ax.set_rticks(np.linspace(0.0,r_max,5)) # show 5 different distances

```

```

ax.set_thetagrids(np.linspace(0.0,180.0,10)) # show 10 angles
angles = np.arange(0,181,1) # 0 - 180 degrees
theta = angles*(np.pi/180.0) # to radians
dists = np.ones((len(angles),)) # dummy distances until real data comes in
polys, = ax.plot([],linestyle=' ',marker='o',markerfacecolor = 'w',
                  markeredgecolor='#EFEFEF',markeredgewidth=1.0,
                  markersize=10.0,alpha=0.9) # dots for radar points
line1, = ax.plot([],color='w',
                  linewidth=4.0) # sweeping arm plot

# figure presentation adjustments
fig.set_size_inches(0.96*(screen_res[0]/dpi),0.96*(screen_res[1]/dpi))
plot_res = fig.get_window_extent().bounds # window extent for centering
win.wm_geometry('+'+{0:1.0f}+{1:1.0f}+'.\
                format((screen_res[0]/2.0)-(plot_res[2]/2.0),
                       (screen_res[1]/2.0)-(plot_res[3]/2.0))) # centering plot
fig.canvas.toolbar.pack_forget() # remove toolbar for clean presentation
fig.canvas.set_window_title('Arduino Radar')

fig.canvas.draw() # draw before loop
axbackground = fig.canvas.copy_from_bbox(ax.bbox) # background to keep during loop

#####
# button event to stop program
#####

def stop_event(event):
    global stop_bool
    stop_bool = 1
prog_stop_ax = fig.add_axes([0.85,0.025,0.125,0.05])
pstop = Button(prog_stop_ax,'Stop Program',color='#FCFCFC',hovercolor='w')
pstop.on_clicked(stop_event)
# button to close window
def close_event(event):
    global stop_bool,close_bool
    if stop_bool:
        plt.close('all')
    stop_bool = 1
    close_bool = 1
close_ax = fig.add_axes([0.025,0.025,0.125,0.05])
close_but = Button(close_ax,'Close Plot',color='#FCFCFC',hovercolor='w')
close_but.on_clicked(close_event)

fig.show()

#####
# infinite loop, constantly updating the
# 180deg radar with incoming Arduino data
#####
#
start_word,stop_bool,close_bool = False,False,False
while True:
    try:
        if stop_bool: # stops program
            fig.canvas.toolbar.pack_configure() # show toolbar
            if close_bool: # closes radar window
                plt.close('all')
            break
    
```

```

ser_bytes = ser.readline() # read Arduino serial data
decoded_bytes = ser_bytes.decode('utf-8') # decode data to utf-8
data = (decoded_bytes.replace('\r','')).replace('\n','')
if start_word:
    vals = [float(ii) for ii in data.split(',')]
    if len(vals)<2:
        continue
    angle,dist = vals # separate into angle and distance
    if dist>r_max:
        dist = 0.0 # measuring more than r_max, it's likely inaccurate
    dists[int(angle)] = dist
    if angle % 5 ==0: # update every 5 degrees
        pols.set_data(theta,dists)
        fig.canvas.restore_region(axbackground)
        ax.draw_artist(pols)

    line1.set_data(np.repeat((angle*(np.pi/180.0)),2),
                  np.linspace(0.0,r_max,2))
    ax.draw_artist(line1)

    fig.canvas.blit(ax.bbox) # replot only data
    fig.canvas.flush_events() # flush for next plot
else:
    if data=='Radar Start': # stard word on Arduno
        start_word = True # wait for Arduino to output start word
        print('Radar Starting...')

    else:
        continue

except KeyboardInterrupt:
    plt.close('all')
    print('Keyboard Interrupt')
    break

```

BIBLIOGRAPHY

- [1] Ansar, M. Follow Me Robot, Youtube Channel, GitHub
- [2] Beck, Alan. Robotic Dog, Advancing Science Serving Society.
- [3] DigiKey Electronics
- [4] Jones, Marlin P. Ultrasonic Module HC-SR04 User Guide
- [5] McZingga. Macduino, 2013
- [6] Reese, Lynnette. The working principle, applications and limitations of ultrasonic sensors, 2019 microcontroller.com
- [7] Sayeed, Abu. <https://www.instructables.com/Controlling-Two-DC-Motors-With-L293D-IC/>
- [8] Codebender.cc. How to use L293D Motor Drivers, Instructables. <https://www.instructables.com/How-to-use-the-L293D-Motor-Driver-Arduino-Tutorial/>
- [9] Spivy, Austin. <https://dzone.com/articles/driving-a-dc-motor-with-an-arduino-and-the-l293d-m>
- [10] Zhmud, Vadim et al. Application of ultrasonic sensor for measuring distances in robotics, 2018. Journal of Physics: Conference Series. 1015. 032189. 10.1088/1742-6596/1015/3/032189.

- [11] Hrisko, Joshua. Radar Emulator with Arduino + Python, 2020. Makers Portal.
<https://makersportal.com/blog/2020/3/26/arduino-raspberry-pi-radar>

Index of comments

- 1.1 Nice abstract
- 1.2 Give some citations of previous works in the intro part.
- 2.1 adapted from Ref. [9]
- 2.2 use \begin{equation}
- 3.1 what is this bar ? use savefig or a similar command to save your figure
- 4.1 PERFECT !
- 5.1 Such a cool car (?!?)