

Rader System

A  
Project Report  
On  
**“Rader System “**



**SUBMITTED BY: RAJ**

**SONI (210346620044)**

**MANSI PATIL (210346620075)**

**ABHISHEK KUMAR (210346620209)**

**GUIDED BY:**

**Asst. prof. ASFAK DARVESHWALA**

**DIPLOMA IN INFORMATION TECHNOLOGY  
SUBMITTED TO:**

**PARUL POLYTECHNIC INSTITUTE LIMDA,**

**Wagodhia Limda**

Rader System

**PARUL POLYTECHNIC INSTITUTE**  
**INFORMATION TECHNOLOGY DEPARTMENT**  
**LIMDA, WAGHODIA, VADODARA**



**C E R T I F I C A T E**

This is to certify that **Raj Soni, Mansi Patil & Abhishek Kumar** students of Information Technology, bearing Enrollment No: **210346620044, 210346620075 & 210346620209** have satisfactorily completed their project work as a part of course curriculum in Diploma in (6th Semester) having a project title “Rader System”.

**Signature of Guide**  
**Asst. prof. Prof. Asfak Darveshwala**

**Head of Department**  
**Prof. Vidita Patel**

**PARUL POLYTECHNIC INSTITUTE  
INFORMATION TECHNOLOGY DEPARTMENT  
LIMDA, WAGHODIA, VADODARA**



**C E R T I F I C A T E**

This is to certify that **Raj Soni, Mansi Patil & Abhishek Kumar** students of Information Technology, bearing Enrolment No: **210346620044, 210346620075 & 210346620209** have satisfactorily completed their project work as a part of course curriculum in Diploma in (6th Semester) having a project title “Rader System”.

**DATE:**

**INTERNAL JURY:**

**EXTERNAL JURY:**

## Acknowledgment

This was a challenging project from the starting only and we would never have been able to complete it without the skills and talents of many people around us. This project is the creation of the people as mentioned below.

At the completion of project work successfully, how we can forget Our Guide **Asst. Prof. Asfak Darveshwala** we would glad to express our hearty thanks to her for sharing of her knowledge and we're also thankful to all the staff members for providing us the enthusiastic environment during the project.

My sincere thanks to **Dr. Jatin Vaidya** (Principal), **Asst. Prof. Vidita Patel** (H.O.D. IT) for molding our thoughts and vision towards subject. We would thanks to each and every who have helped us directly or indirectly in our project.

Yours Sincerely,

RAJ SONI (210346620044)

MANSI PATIL (210346620075)

ABHISHEK KUMAR (210346620209)

# Abstract

Embark on a journey into the future as IoT synergizes with radar systems, revolutionizing detection capabilities. This abstract delves into the dynamic fusion of radar technology and the Internet of Things (IoT), creating smart, connected solutions. From smart cities to autonomous vehicles, experience the transformative synergy that transcends limits, elevating situational awareness and reshaping the landscape of monitoring and safety. This exploration unveils a game-changing collaboration between radar technology and IoT, unlocking unprecedented possibilities in our hyperconnected world. Brace yourself for a paradigm shift, as radar systems seamlessly integrate with IoT, paving the way for intelligent solutions that redefine our understanding of detection and connectivity.

# INDEX

<b>Chapter:1 Introduction .....</b>	<b>7-19</b>
1.1 The Arduino UNO?.....	8-9
1.1.1 Why Arduino?.....	9-11
1.1.2 Operation of Arduino .....	11
1.2 The HC-SR04 Ultrasonic Sensor .....	12-16
1.2.1. Features.....	13
1.2.2 HC-SR04: Pin Definition.....	14
1.2.3 Operation of the HC- SR04.....	14-16
1.3 Micro- Servo Motor .....	16-19
<b>Chapter:2 Circuit Schematics.....</b>	<b>20-21</b>
2.1 Arduino Pins Connection .....	20
2.2 Breadboard view .....	20
2.3 Schematic view .....	21
<b>Chapter:3 Development and Simulation .....</b>	<b>22-25</b>
3.1 Arduino IDE .....	22
3.2 Simulation with Proteus.....	23
3.3 Processing .....	23-25
<b>Chapter:4 Explaining the Code .....</b>	<b>26-34</b>
4.1 The Arduino Code.....	26-29
4.2 The Processing Code.....	30-34
<b>Chapter:5 Observation .....</b>	<b>35-48</b>
5.1 Calibration .....	35
5.2 Graphs.....	38
5.1 Challenges.....	41
5.1 Identified problem.....	42
5.1 Proposed Solution .....	43-48
<b>Chapter:6 Conclusion.....</b>	<b>49</b>

## **Chapter-1: Introduction.**

# Rader System

## 1.1 The Arduino Board



Arduino is an open source electronics platform for fast prototyping of projects for users with minimal knowledge or experience in electronics and programming. We have used the Arduino UNO, the most widely used variant of the Arduino.

Technically, Arduino Uno is a microcontroller board based on the 8-bit ATmega328P microcontroller. With 14 digital input/output pins (including 6 PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button, it has everything needed to support the



## Rader System

microcontroller, we simply have to connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

Arduino also comes with the Arduino IDE, in which we can write code and upload to the Arduino. The programming language used for an Arduino, called the Arduino Programming language, is very similar to C/C++ except that we can use inbuilt functions of the Arduino libraries which keep the code very simple and short. propels us towards a future of unprecedented innovation and intelligence.

### 1.1.1 Why Arduino?

We had the following reasons strongly backing our decision to choose the Arduino UNO.

#### **Support:**

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. This has enabled a very large support base for Arduino, more than any other micro controller board ever released in the market. Every problem/bug/question that we ever had throughout the course of this project was promptly answered by questions in online forums, as there was a very high probability that scores of people had already posted the same question and the issue was resolved by other experienced users.

## Rader System

Microcontroller	ATmega328P(8-bit)
Operating Voltage	(DC)5V
Input Voltage	(recommended) 7-12V; (limits) 6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Need of External Programmer	NO
Built-in LED	pin13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Table 1.1: Features of Arduino UNO

**Cost Effective:** The Arduino UNO, its basic variant that every beginner to embedded systems usually starts with, is available for less than \$25. Being an open source hardware and due to its simplicity, it is easily and widely replicable, which is also responsible for the growth of such a large community as mentioned in the earlier reason.

**Features:** It is generally observed that the computing power and inbuilt memory of the Arduino easily supports a wide variety of circuits with multiple components along with continuous serial communication with another device/computer. Compared to the system configuration of the beard, our project would require fairly low amount of resources. The no. of GPIO pins on the UNO is more than sufficient considering the need for the circuitry in our project, as we shall see when we look at the pinouts of the servo motor and the sensor the only two other devices being used in the project. One important advantage of using the UNO is that unlike

## Rader System

most previous programmable circuit boards, the Arduino UNO does not need a separate piece of hardware (called a programmer) in order to load new code onto the board we can simply use a USB cable. The ATmega328 on the Arduino/Genuino Uno comes preprogrammed with a bootloader that allows us to upload new code to it without the use of an external hardware programmer. Moreover, the components we have used work under the same operating voltage and current conditions as an Arduino, especially considering the fact that such components are nowadays sold with their compatibility with Arduino in mind.

Considering the above factors, we found Arduino UNO to be the most appropriate microcontroller board to work with for this project.

### 1.1.2 Operation of Arduino?

Once a program written for an Arduino, called a "Sketch" is uploaded to the board via the Arduino IDE, the Arduino environment performs some small transformations to make sure that the code is correct C/C++. It then gets passed to a compiler (gcc), which turns the human readable code into machine readable instructions (object files). Then, the code gets combined with (linked against) the standard Arduino libraries that provide basic functions like digitalWrite() or Serial.print(). The result is a single Intel hex file, which contains the specific bytes that need to be written to the program (flash) memory of the chip on the Arduino board. This file is then uploaded to the board: transmitted over the USB or serial connection via the bootloader already on the chip or with external programming hardware.

# Rader System

## 1.2 The HC-SR04 Ultrasonic Sensor



Arduino also comes with the Arduino IDE, in which we can write code and upload to the Arduino. The programming language used for an Arduino, called the Arduino Programming language, is very similar to C/C++ except that we can use inbuilt functions of the Arduino libraries which keep the code very simple and short. propels us towards a future of unprecedented innovation and intelligence. The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats or dolphins do. It offers non-contact range detection from 2cm to 400 cm or 1 to 13 feet. T's operation is not affected by sunlight or black material like sharp range finders are (although acoustically soft materials like cloth can be difficult to detect). It comes complete with ultrasonic transmitter and receiver module.

The human ear can only detect sounds between 20Hz-20kHz. The sound waves beyond 20kHz are called ultrasonic waves or ultrasound

The principle of ultrasonic distance measurement uses the velocity of ultrasonic waves spreading in air, measuring the time from launch to reflection when it encounters an obstacle. We then calculate the distance between the transmitter and the obstacle

## Rader System

Electrical Parameters	HC-SR04 Ultrasonic Module
Operating Voltage	(DC)5V(4.5-5.5V)
Quiescent Current(inactivity)	2mA(1.5-2.5mA)
Working Current	15mA(10-20mA)
Operating Frequency	40KHZ(ultrasonic)
Farthest Range	4m
Nearest Range	2cm
Resolution	0.3cm
Measuring Angle	15 Degrees
Input Trigger Signal	10us TTL pulse
Output Echo Signal	Output TTL. level signal, proportional with range
Dimensions	45*20*15mm

Table 1.2: Features of HC-SR04

### **NOTE- Use of only stationary objects**

Since the sensor considers the distance to an obstacle that reflects the ultrasonic wave, we cannot use it to make trustworthy measurements of the position of moving objects. This is majorly due to the fact that this sensor works on the principle of transmitting a set of pulses and receiving their reflection, before transmitting again, as we shall see later Hence, there is no way that we can continuously track the position of an object with such a sensor. As far as we know, that can only be done by some type of a camera, which would've introduced the concepts of computer vision and taken us beyond the scope and costs of the current project.

# Rader System

## 1.1.1 HC- SR04 : Pin Definitions

- Vec-5V Power Supply
- Trig-Trigger Pin (Input)
- Echo Receive Pin (Output)
- GND-Power Ground

The Trig and Echo pins are used for transmission and reception of ultrasonic waves, respectively.

## 1.1.2 Operation of the HC-SR04

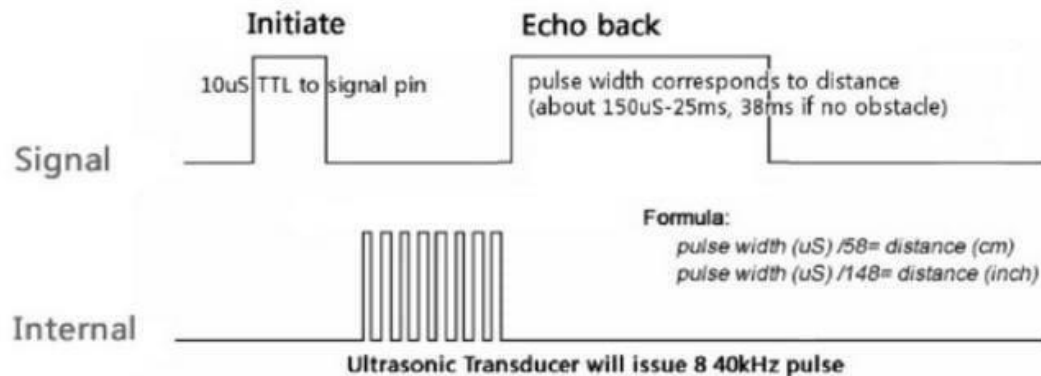


Figure 1.3: Timing Diagram of HC-SR04 Ultrasonic sensor

The timing diagram of HC-SR04 is shown in Figure 1.3. To start measurement, Trig of 5804 must receive a HIGH pulse (5V) for at least 10us, which will initiate the sensor to transmit 8 cycles of

## Rader System

ultrasonic burst at 40kHz. It then waits for the reflected ultrasonic wave. When the sensor detects the ultrasonic wave through the receiver, it will set the Echo pin to HIGH (5V), for as long as it detects an incoming ultrasonic pulse at the receiver.

### Calculation of distance

The delay between transmission and reception of the final pulse (e, between when the Trig pin is set back to LOW and when the Echo pin turns HIGH) is a time period proportional to the distance travelled by it. To obtain the distance, we start a counter the moment the Trig pin is set to LOW which keeps counting till the microcontroller detects a HIGH pulse at the Echo pin. Time Width of pulse, in us (micro second).

Distance in centimeters= Time/58 Distance in inches=

Time/148

as mentioned in its data sheet. We can also utilize the speed of sound, which is 40m/s in a generic case.

This entire process can be translated into the Arduino code as shown in example 1.1.

## Rader System

### Example 1.1 (Arduino Code for calculating distance from HC-SR04 sensor)

```
/* Function for calculating the distance measured by the
   Ultrasonic sensor*/
float calculateDistance(){
  unsigned long T1 = micros();
  digitalWrite(trigPin, LOW); // trigPin needs a fresh LOW
    pulse before sending a HIGH pulse that can be
    detected from echoPin
  delayMicroseconds(2); //DELAY #2: time for which low trig.
    pulse is maintained before making it high
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10); //DELAY #3: Sets the trigPin on
    HIGH state for 10 micro seconds
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH); // Reads the echoPin,
    returns the sound wave travel time in microseconds
  distance = (duration/2)/29.1; //in cm. Calibrated
    form. Datasheet shows "duration/58" as the formula

  return distance;
}
```

### Need of calibration of the Sensor

As we can notice in the given code, we had to change the formula for calculation of distance, as the values were not close to real values of distance of objects kept in front of it, as tested by us. We would discuss it in a later stage.

## 1.3 Micro- Servo Motor



## Rader System



Figure 1.4: Micro- Servo Motor

We had to use a servo motor in order to rotate the HC-SR04 and give it a wider field of view over the area around it. This Micro- Servo can rotate approximately 180 degree (90 in each side) with an operating speed of 0.1s/60degrees. Moreover, it has an operating voltage of 4.8V (5V) which is in the same range as of the Arduinio UNO.

Other details:

- Weight: 9g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf.cm
- Operating speed: 0.1 s/60 degrees
- Operating voltage: 4.8 V (5V)
- Deadband width: 10 us (The servo will not move so long as subsequent commands to it are within the deadband timing pulse width)

## Rader System

- Temperature range: 0-35C

The Micro-Servo has a signal pin to which the arduino sends the value of angle that it should turn to. Figure 1.5 shows the pins of the Micro-Servo.



Figure 1.5: Pins of Micro- Servo

The Ultrasonic Sensor was mounted on top of it and the apparatus connected to the arduino was kept in a modified transparent plastic case, as shown in Figure

## Rader System



We then proceeded to deciding the circuit connections, as shown in the next section.

## Chapter-2: Circuit Schematics.

### 2.1 Arduino Pin Connections

Pin of Arduino	Connected to
Digital I/O (2)	Trig of HC-SR04
Digital I/O (4)	Echo of HC-SR04
Digital I/O (9)	Signal pin of Servo(orange)
Vcc	Vcc of HC-SR04 and Servo (Red)
GND Pin	GND of HC-SR04 and Servo (Black)

### 2.2 Breadboard View

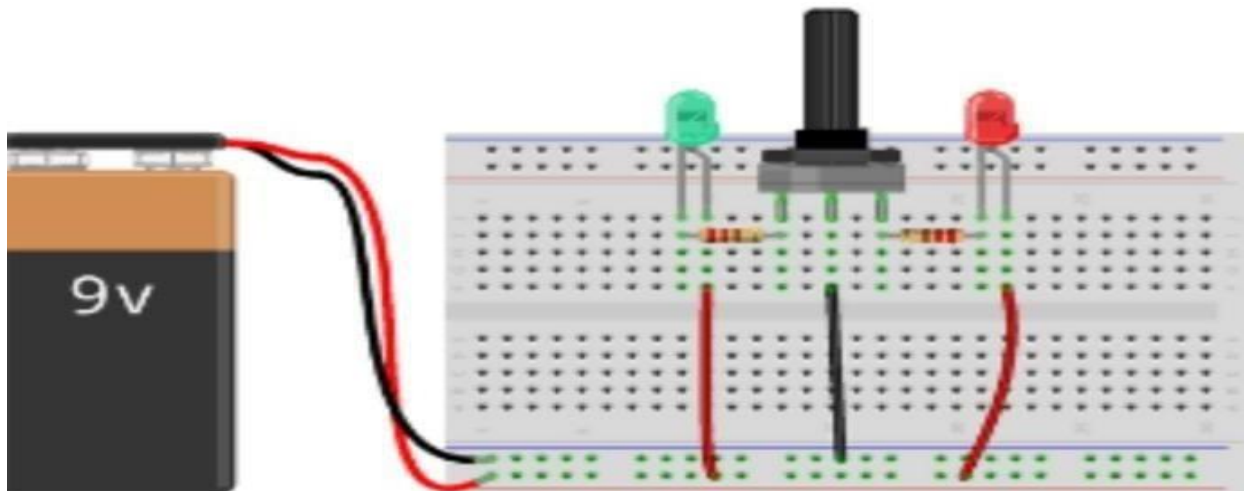


Figure 2.1: Circuit Breadboard view

# Rader System

## 2.3 Schematic View

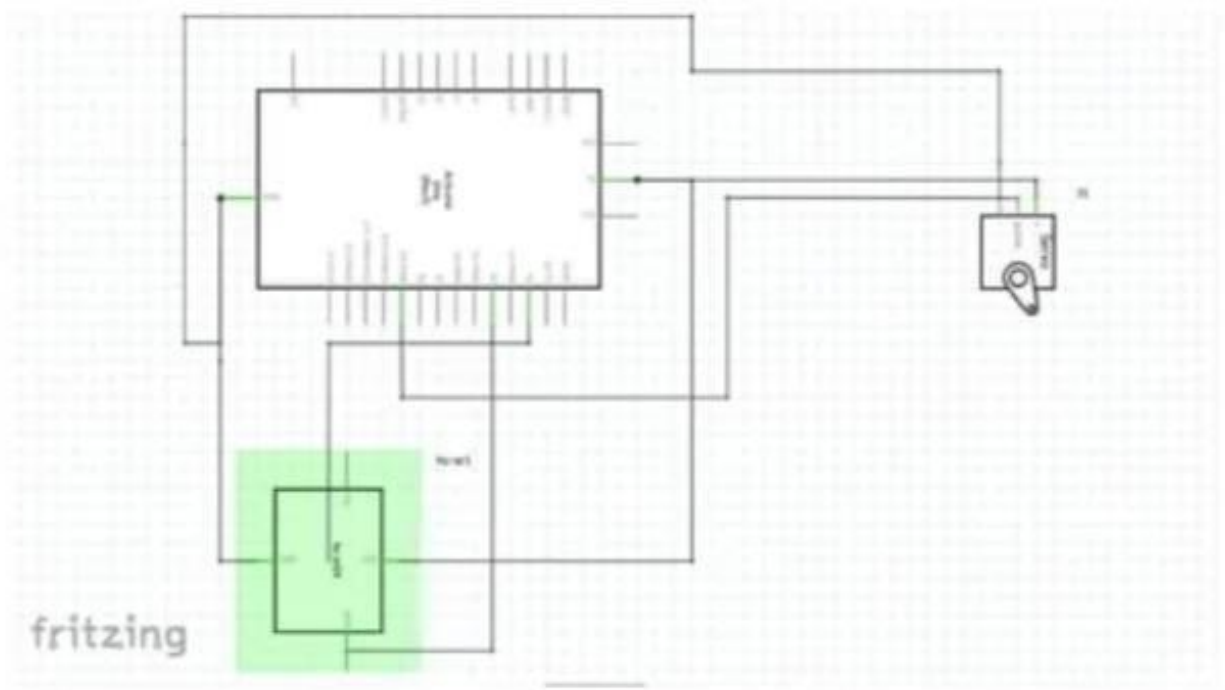


Figure 2.2: Circuit : Schematic view

Above circuit schematics were obtained using the Fritzing software. Once we decided the connections, we had to two more steps left before connecting the actual circuit: setting up the Development Environment and Circuit Simulation.

## Chapter-3: Development and Simulation

### 3.1 Arduino IDE

The code to be burnt to the program memory of an Arduino is written and compiled in the Arduino IDE. The IDE also has a serial monitor, which displays values being received in real-time from the Arduino via serial communication through the serial port (COM ports). The IDE verifies for correct C/C++ syntax and compiles it, before linking it to Arduino Library files. This creates the hex file that contains the code in binary form. It can either be uploaded to the board directly, which the IDE does with the help of the Bootloader program pre-installed on the Arduino, or it can be fed to a simulator, which would simulate and show how the circuit would run and its associated parameters.

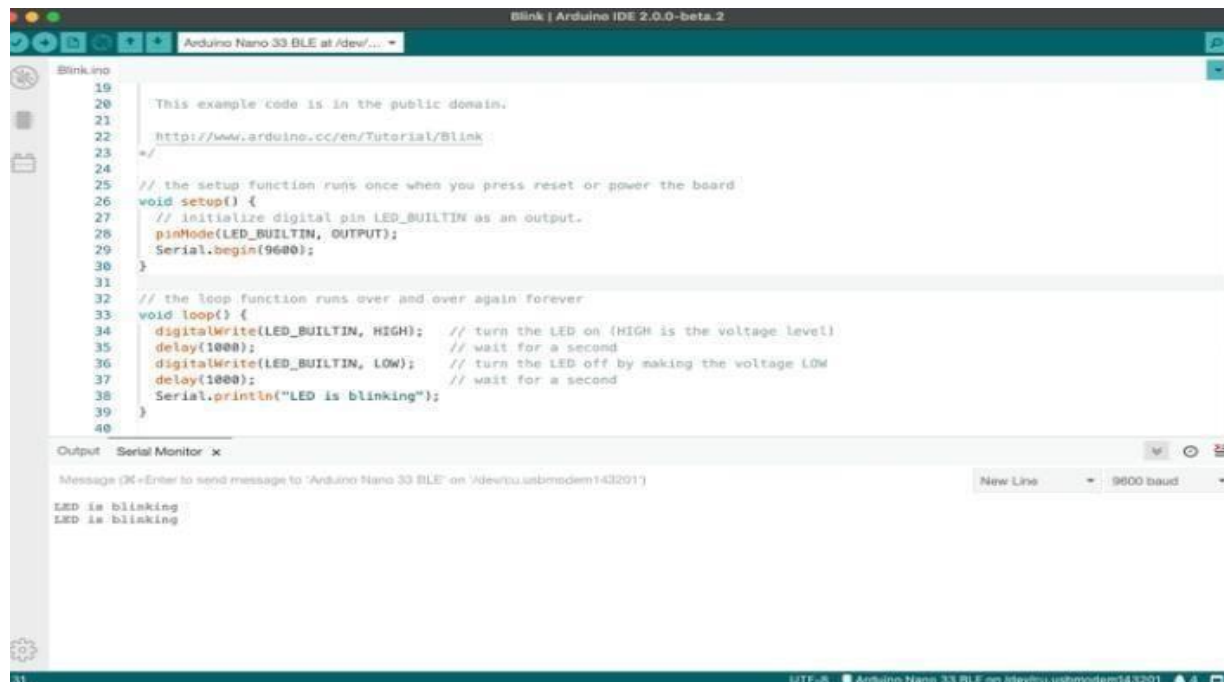


Figure. 3.1: The Arduino IDE

# Rader System

## 3.2 Simulation with Proteus

It is a good practice to simulate a circuit before connecting it, in order to avoid hardware damage/debug issues in a circuit beforehand. We used Proteus Professional 8 for this purpose. Proteus has libraries for most commonly used electronics components, and it also allows us to make our own components, thanks to which we could download unavailable components from the internet, where many users have shared their own custom components.

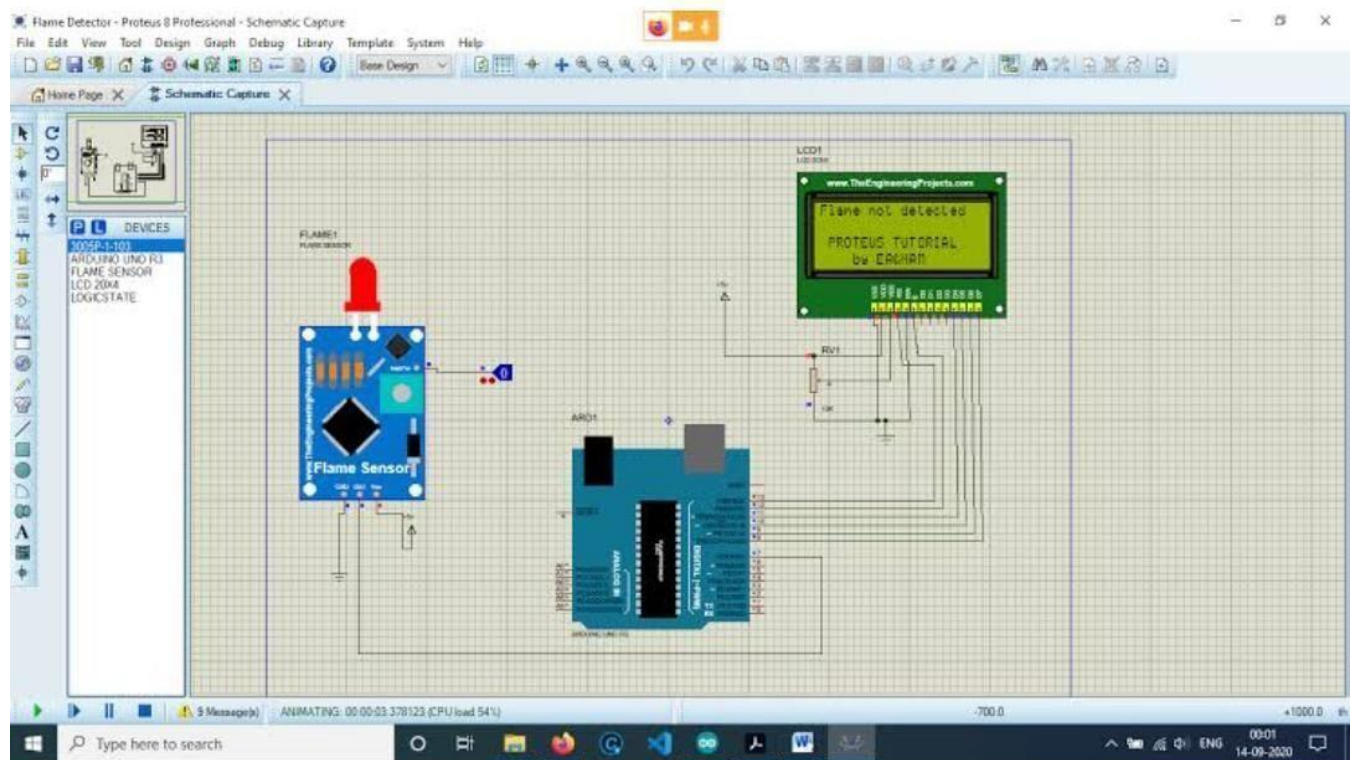



Figure. 3.2: The Proteus Simulator



# Rader System

## 3.3 Processing

Processing is an interactive software to write programs with visual output. We have used Processing 3.2 for generating a 2D radar-map representation of the serial data being obtained from the Arduino through the computer's serial port.



```
import processing.serial.*;
import oscP5.*;
import netP5.*;

float red = 0;

OscP5 oscP5;
Serial Arduino;

void setup() {
  size(320, 480);
  background(100);

  oscP5 = new OscP5(this, 8000);
  Arduino = new Serial(this, Serial.list()[2], 9600);
}

void oscEvent(OscMessage ThisOscMessage) {
  String addr = ThisOscMessage.addrPattern();
  float val = ThisOscMessage.get(0).floatValue();

  if (addr.equals("/1/red/")) {
    red = val;
    println(val);
    Arduino.write("val");
  }
}

void draw() {
  background(red+50, 0, 0);
  fill(0);
  rect(124, 39, 60, 255);
  fill(red, 40, 40);
  rect(124, 255+39, 60, -red);
}
```

```
9.58333
2.083336
9.16667
0.833336
## [2014/8/17 15:48:10] PROCESS @ OscP5 stopped.
## [2014/8/17 15:48:10] PROCESS @ UdpServer.run() socket closed.
```

36

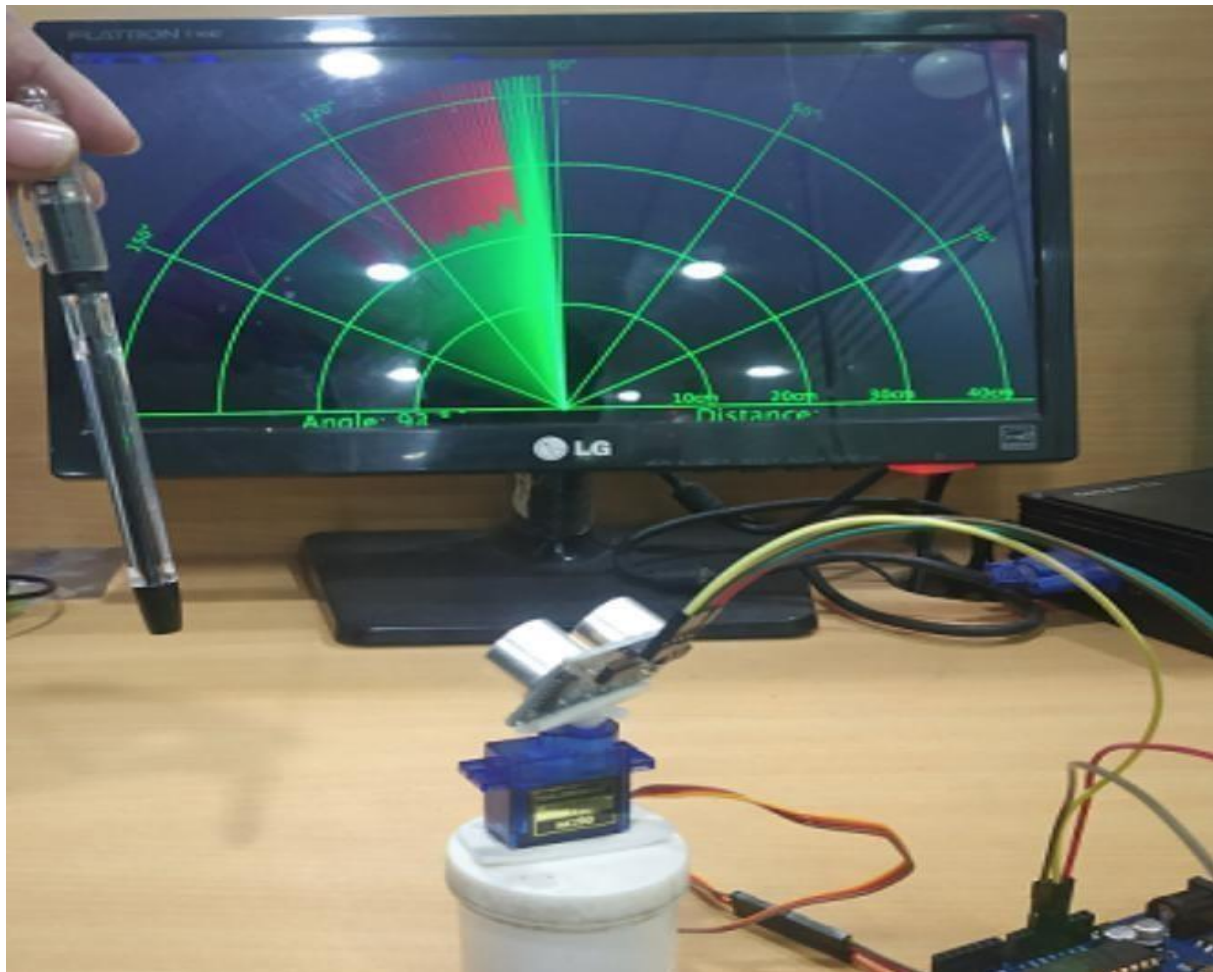
Figure. 3.3: Processing



## Rader System

The code in Processing is written in Java, and primarily consists of drawing the lines and borders of the radar map, and plotting the tracker-bar on the map. We added functionalities such as counting the number of objects, and also tried to calculate the width of the object, apart from just showing the distance of the object.

We made a 360 degrees map so that we could view the area according to our convenience if the angle of the set-up is changed. This angle is the "offset" that we introduced in the Arduino code. The entire Arduino and Processing codes are shown in Appendix C and Appendix D, respectively.



### Chapter-4: Explaining the Code.

Here we'd be briefly going through the flow of the codes written for Arduino and in Processing. The entire Arduino and Processing codes are given (explained with comments) in Appendix C and Appendix D, respectively.

#### 4.1 The Arduino Code

Every Arduino code consists of two standard functions `void setup()` and `void loop()`. The former is used to define and initialise variables, pins and the baud rate of serial communication.

We have programmed the Arduino to control the circuit in the following flow:

1. Rotate the motor to an angle,
2. Use the ultrasonic sensor and calculate the distance,
3. Send both the above values via serial communication,
4. Rotate the motor to the next degree,
5. Repeat steps 2-5.

The following code snippets implement the above given steps:

# Rader System

The following code snippets implement the above given steps:

```
#include <Servo.h>
const int trigPin = 8;
const int echoPin = 9;
long duration; //declare time duration
int distance; //declare distance
Servo myServo; // Object servo

void setup() {
  pinMode(trigPin, OUTPUT); // trigPin as an output
  pinMode(echoPin, INPUT); // echoPin as an input
  Serial.begin(9600);
  myServo.attach(10); // pin connected to Servo
}

void loop() {
  // rotating servo i++ depicts increment of one degree
  for(int i=0;i<=180;i++){
    myServo.write(i);
    delay(30);
    distance = calculateDistance();
    Serial.print(i);
    Serial.print(",");
    Serial.print(distance);
    Serial.print("\n");
  }
```

## Rader System

```
// Repeats the previous lines from 180 to 0 degrees
for(int i=180;i>0;i--){
  myServo.write(i);
  delay(30);
  distance = calculateDistance();
  Serial.print(i);
  Serial.print(" ");
  Serial.print(distance);
  Serial.print(" ");
}
}

int calculateDistance(){
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance= duration*0.034/2;
  return distance;
}
```

Keeping the above code in a void loop() ensures that the steps get repeated in a loop, as desired.

# Rader System

## 4.2 The Processing Code

The JAVA code written for processing divides its work into a few main functions, which are customary for every code written in Processing. The functions are.

- void setup()  
to initialise variables and baud rate of serial communication,
- void draw()  
to make a 2D/3D drawing on the screen on which to show the output,
- void serialEvent()  
the function dealing with incoming/outgoing serial data, and operations on it

Customised functions were used to draw several parts of the displayed graph, which were called from void draw(). These are:

- void drawRadar()  
to draw the basic circular layer of graphics
- void drawLine()  
to draw a line with trails which would be used to sweep over the area implying scanning
- void drawObject()  
to draw a colored line over the scanning figure to indicate presence of an object whose length would be dependant on the distance of the object from the sensor

## Rader System

- `void drawText()`  
to place text at different places on the map to show the serial and calculated data values.

The Java Code we ran in Processing is given below:

```
import processing.serial.*;
import java.awt.event.KeyEvent;
import java.io.IOException;
Serial myPort; // defubes variables
String angle="";
String distance="";
String data="";
String noObject;
float pixsDistance;
int iAngle, iDistance;
int index1=0;
int index2=0;
PFont orcFont;
void setup() {
  size(1366, 768);
  smooth();
  myPort = new Serial(this,"COM3", 9600); // changethis acc
  myPort.bufferUntil('.'); // reads the data from the serial port up to the character .?
}
void draw() {
  fill(98,245,31);
  // simulating motion blur and slow fade of the moving line
  noStroke();
  fill(0,4);
  rect(0, 0, width, height-height*0.065);
  fill(98,245,31); // green color
  // calls the functions for drawing the radar
  drawRadar();
  drawLine();
  drawObject();
  drawText();
}
```



## Rader System



# Rader System

```
    text("Distance: ", width-width*0.36, height-
height*0.0277);
    if(iDistance<40) {
        text(" " + iDistance + " cm", width-width*0.225, height-
height*0.0277);
    }
    textSize(25);
    fill(98,245,60);
    translate((width-width*0.4994)+width/2*cos(radians(30)),
(height-height*0.0907)-width/2*sin(radians(30)));
    rotate(-radians(-60));
    text("30?",0,0);
    resetMatrix();
    translate((width-width*0.503)+width/2*cos(radians(60)),
(height-height*0.0888)-width/2*sin(radians(60)));
    rotate(-radians(-30));
    text("60?",0,0);
    resetMatrix();
    translate((width-width*0.507)+width/2*cos(radians(90)),
(height-height*0.0833)-width/2*sin(radians(90)));
    rotate(radians(0));
    text("90?",0,0);
    resetMatrix();
    translate(width-width*0.513+width/2*cos(radians(120)),
(height-height*0.07129)-width/2*sin(radians(120)));
    rotate(radians(-30));
    text("120?",0,0);
    resetMatrix();
    translate((width-
width*0.5104)+width/2*cos(radians(150)),(height-
height*0.0574)-width/2*sin(radians(150)));
    rotate(radians(-60));
    text("150?",0,0);
    popMatrix();
}
```

### Chapter-5: Observation.

We observed data from both the serial monitor of the Arduino IDE, as well as from the graph in processing. Since we wanted our device to give the best results as quickly as possible, we decided to calibrate it.



Figure 5.1: Data observed in serial monitor of the Arduino IDE

## 5.1 Calibration

### 5.1.1 Calibration of measurement

Arduino also comes with the Arduino IDE, in which we can write code and upload to the Arduino. The programming language used for an Arduino, called the Arduino Programming language, is very similar to C/C++ except that we can use inbuilt functions of the Arduino libraries which keep the code very simple and short. propels us towards a future of unprecedented innovation and intelligence. We realised that actual measurements of distance/count/width of an object placed in the vicinity of the apparatus were different from the ones provided by our device. Hence, we proceeded to calibrate each part of our device that was capable of taking a measurement.

The quantities that were measured in our project before going for any further calculations were: Angle and Distance. The rest were only obtained by playing around with these two main quantities. Hence, we needed them to be as accurate as possible.

## Rader System

We did not notice any error in the value of angle that the motor was rotating to, hence we did not have to do any sort of calibration for it.

To calibrate the distance, we placed several objects at fixed distances from the sensor on a white chart paper, with values of distances marked on it with the help of a ruler. We then made measurements keeping the sensor still, and made an adjustment to the formula in the code by observing the multiplication factor that needed to be introduced into the previous formula in order to get the correct values. Hence, the formula was changed from "duration/58" to "duration/58.2".

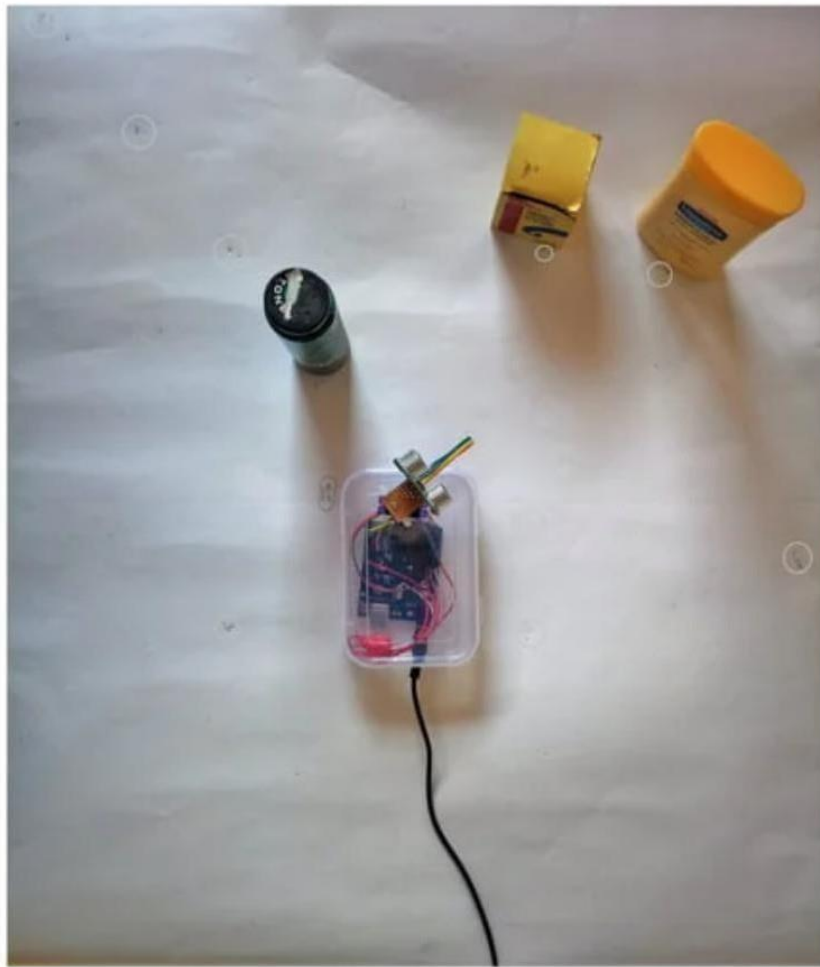


Figure 5.2: Calibration on chart paper with marked distance

## Rader System

### 5.1.2 Optimising Speed

We wanted to optimise our device to work as quickly as possible, without affecting normal execution of the program. So we optimised our code, considering the delays in physical movement, time periods of recieved/transmitted waves and time taken to process the data.

We took into account the several delays introduced in the code, along with the delay in rotation of the motor (specified by its speed given in its datasheet, tested to be accurate) and delay in calculation of distance.

Accordingly, we minimised each of the values given as arguments to the Delay() function used at different parts of the Arduino code, to the safest least value possible. Each of the delays with descriptions can be observed in the Arduino code given with comments in Appendix

## 5.2 Graph

We obtained the following (error vs. position) graphs for three different kinds of objects kept in front of the sensor. In each case, we noted down values for both the sensor being stationary, and rotating. The three different kinds of objects were

Object1= Hollow plastic bottle (width=6cm) Object2 = Solid DC battery (width=3cm)  
Object3= Metallic pen (width=0.8cm)

## Rader System

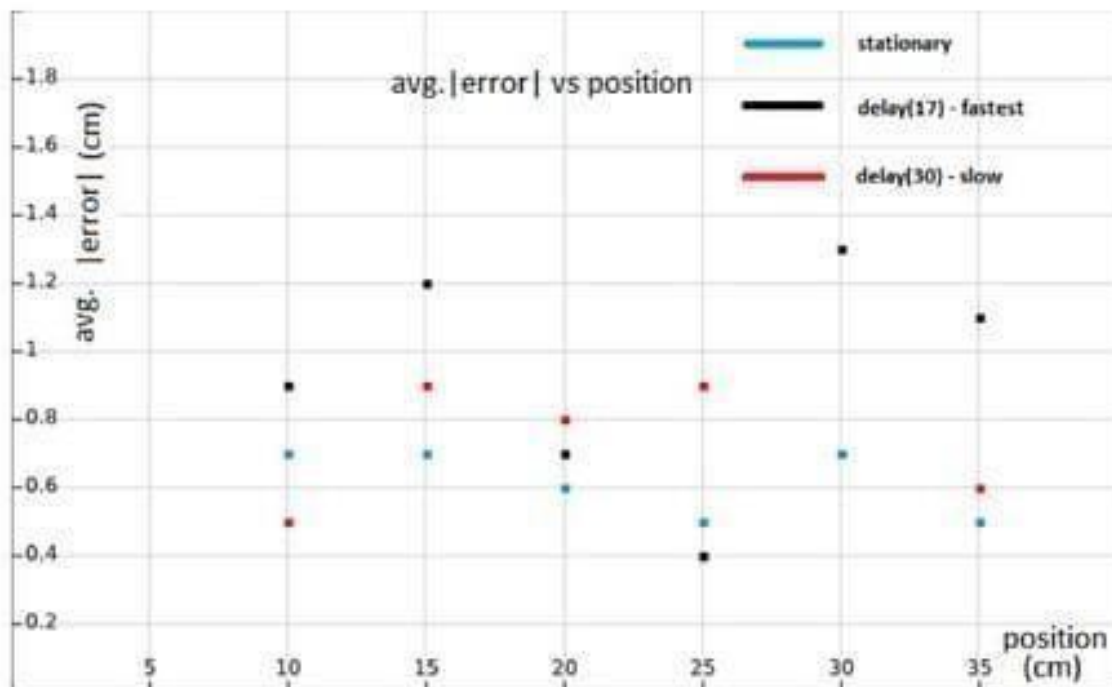


Figure 5.3 Object1- avg.l error l vs position

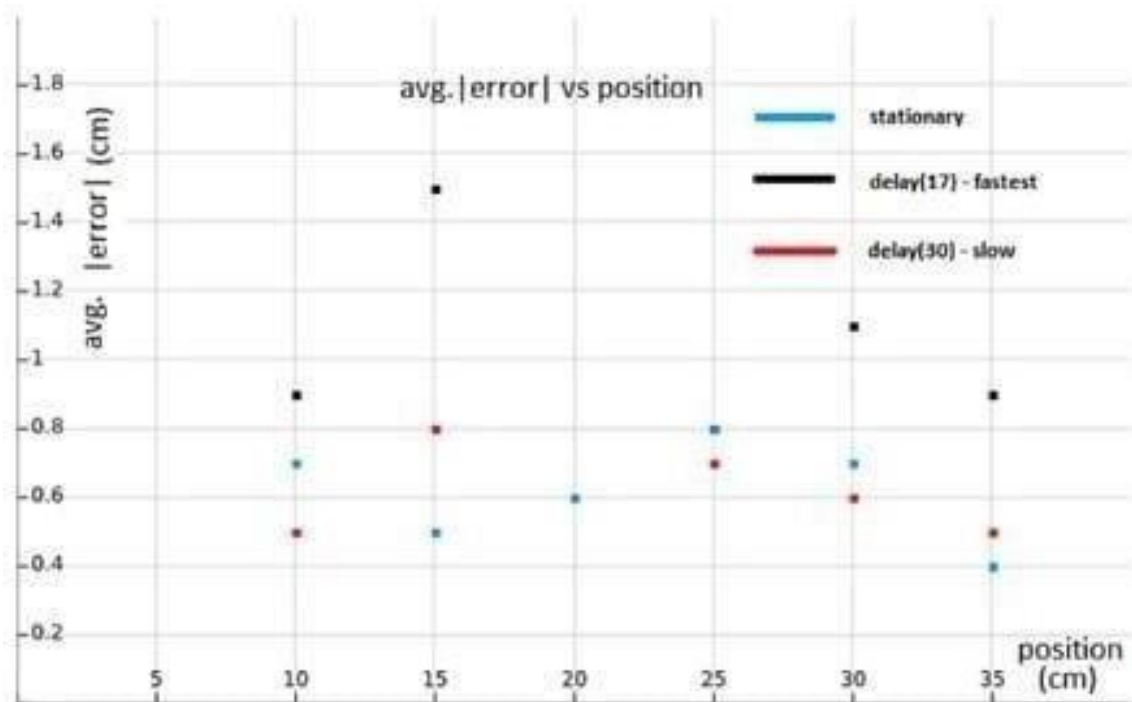


Figure 5.4 Object2- avg.l error l vs position

## Rader System

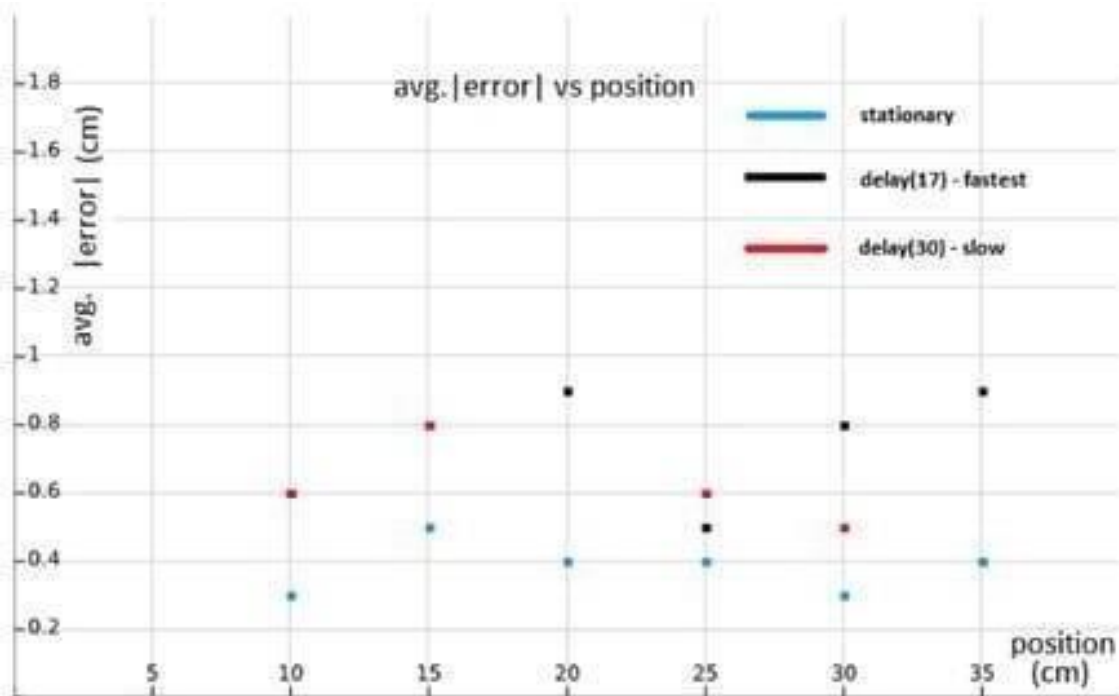
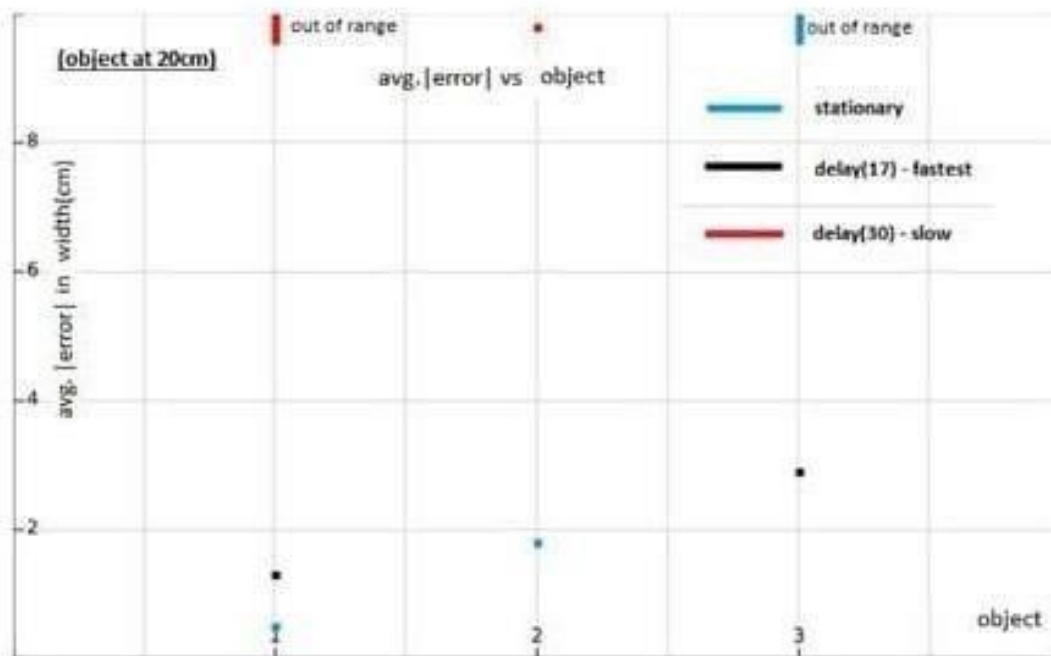


Figure 5.5 Object3- avg.l error l vs position

Finally, we also plotted the error on data of width of object vs. position, which was something like this:



## Rader System

Figure 5.6 Object wise- avg.l error l in width at x=20cm

### 5.3 Challenges

As observed from the graphs plotted in the previous section, we are not able to determine any particular increasing/decreasing order being followed by the errors in measurement of the "distance" quantity.

Moreover, by increasing the speed of measurement of the device (by reducing the manual delays introduced inside the Arduino code within safe limits), the error does not necessarily seem to increase with speed. The "width of the object" quantity seems to be correct at one instant and completely wrong at any other instant with very random error values.

However, what we did observe were sudden jumps in errors. Among several trials, their occurrence increased with the speed. That is, the device was detecting stray values, and the amount of stray values per experiment increased with the speed of the device.

### 5.4 Identified Problem

With the help of ideas from several online electronics forums, we carefully studied the flow of the code, which exposed the flaw in our logic. The problem lied in the fact that the code took variable amount of time in calculating the distance of an object, proportional to the distance itself. This fact could be derived from the line in the code

```
Duration= pulseIn(echoPin, HIGH);
```

where the function pulseIn starts a counter which counts till the value at echoPin is HIGH. Here, our program only waited to receive the first returning pulse, before quickly sending the output and turning the motor to the next angle. At some instances, this could mean that by the time the

## Rader System

sensor was ready to take a new measurement, the older echo pulses were still arriving at the receiver, hence a wrong measurement. The older echo pulses could still be around due to

(a) The program having quickly moved to the next set of commands and having made the sensor ready to receive ultrasonic waves this, within the time in which the remaining of the 8 pulses were still being received at the receiver, or

(b) Due to the waves being spherical in nature and the field of view of the sensor being 15 degrees, the waves could've been reflected by some undesired obstacles in the field of view. It could also be the result of multiple reflections.

### 5.5 Proposed Solution

Therefore, we edited the algorithm in such a way, that between each degree of rotation of the motor, the program takes the exact amount of time to run. That is, the program makes up for the time left out compared to the extreme-time-taking case by waiting. We did this by using the fact to our advantage that the maximum time taken by the `calculateDistance()` function is when there is no obstacle. The HC-SR04 sensor waits for 38ms in case no object is detected, before giving a HIGH echo signal. Hence, we ensure that the program always takes a total of 38ms for each iteration of the loop. For this, we used the `micros()` function inside the `calculateDistance()` function of the arduino code, which calculates the number of microseconds elapsed till the code execution started. Here's the added portion of code which did our solution:



## Rader System

**Listing 5.1: Corrected calculateDistance() function**

```
// Function for calculating the distance measured by the
// Ultrasonic sensor
float calculateDistance(){
    unsigned long T1 = micros();
    digitalWrite(trigPin, LOW); // trigPin needs a fresh LOW
    pulse before sending a HIGH pulse that can be detected
    from echoPin
    delayMicroseconds(2); // DELAY #2: time for which low trig
    pulse is maintained before making it high
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10); // DELAY #3: Sets the trigPin on HIGH
    state for 10 micro seconds
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH); // Reads the echoPin,
    returns the sound wave travel time in microseconds
    // distance = duration * 0.034 / 2;
    distance = (duration / 2) / 29.1; // in cm, datasheet
    gives "duration / 58" as the formula

    // To avoid sending data at variable time intervals due to
    // varying time duration taken between execution of above
    // code inside this function depending on distance of
    // obstacle
    // if no object, echo pulse is HIGH after 38ms
    while(micros() - T1 < 38000)
    {
        ;
    }

    return distance;
}
```

Here are the graph replotted for the data obtained after correction,

## Rader System

## Rader System

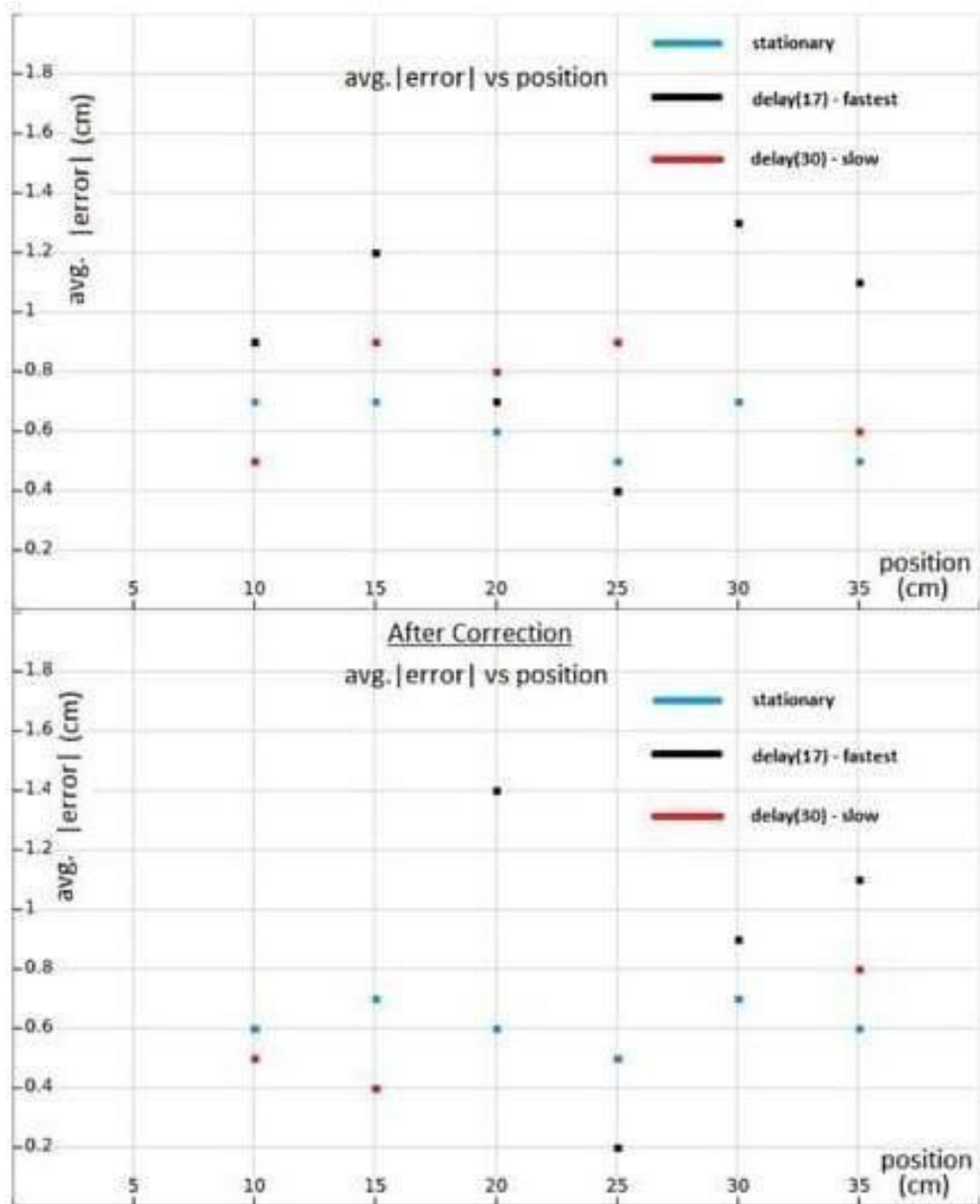


FIGURE 5.7: Object1-avg |error| vs position, before vs after correction

## Rader System

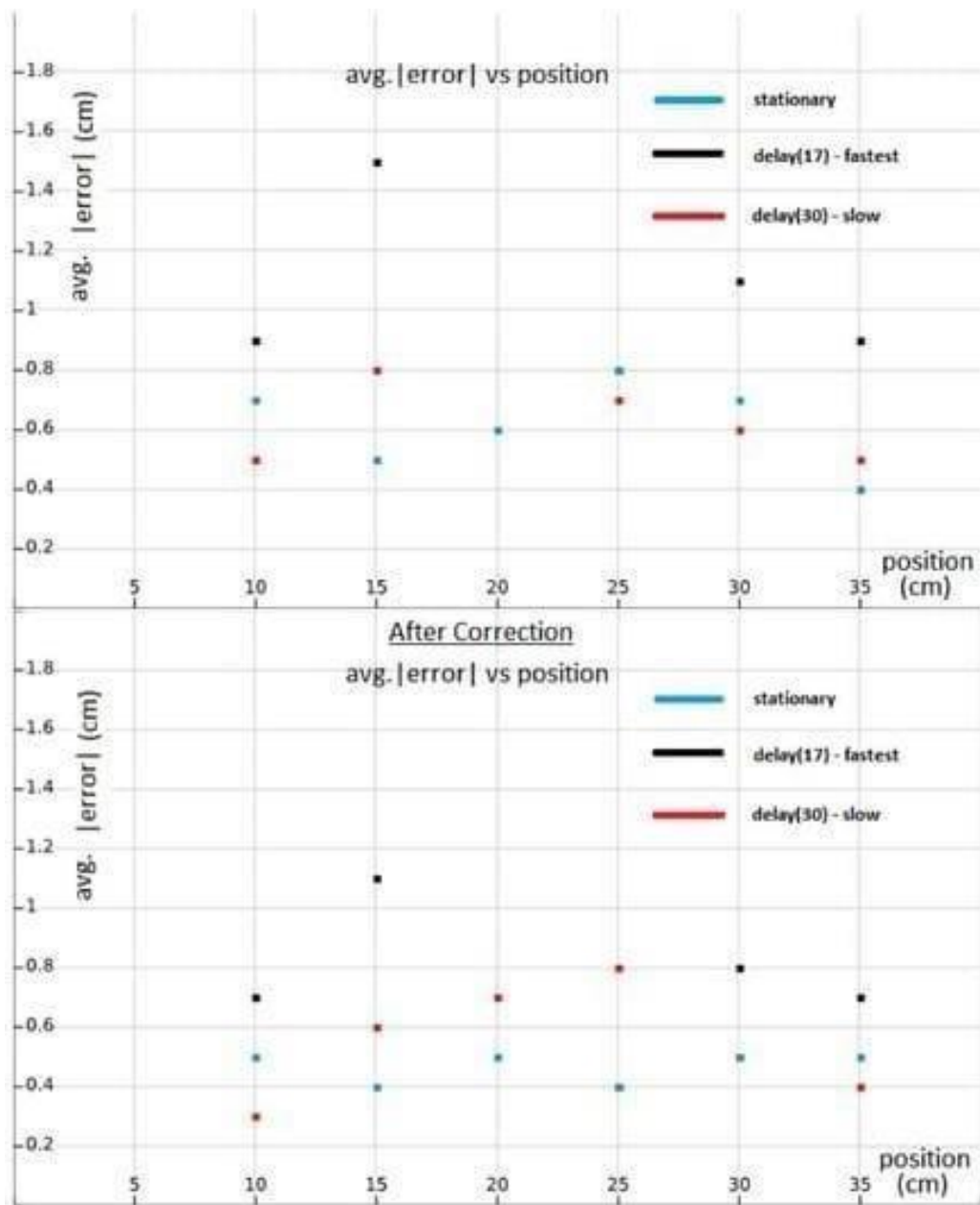


FIGURE 5.8: Object2-avg |error| vs position, before vs after correction

## Rader System

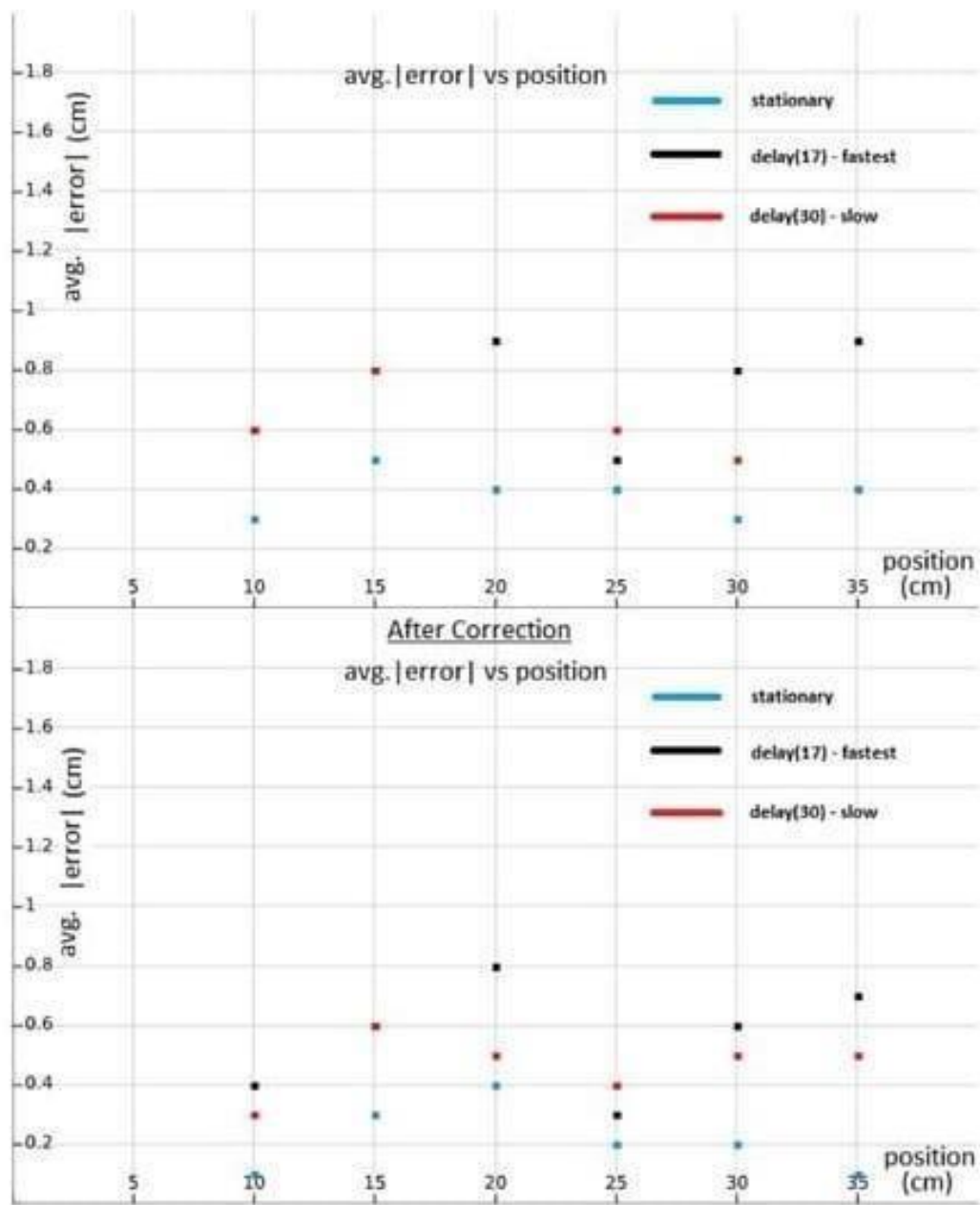


FIGURE 5.9: Object3-avg |error| vs position, before vs after correction

## Rader System

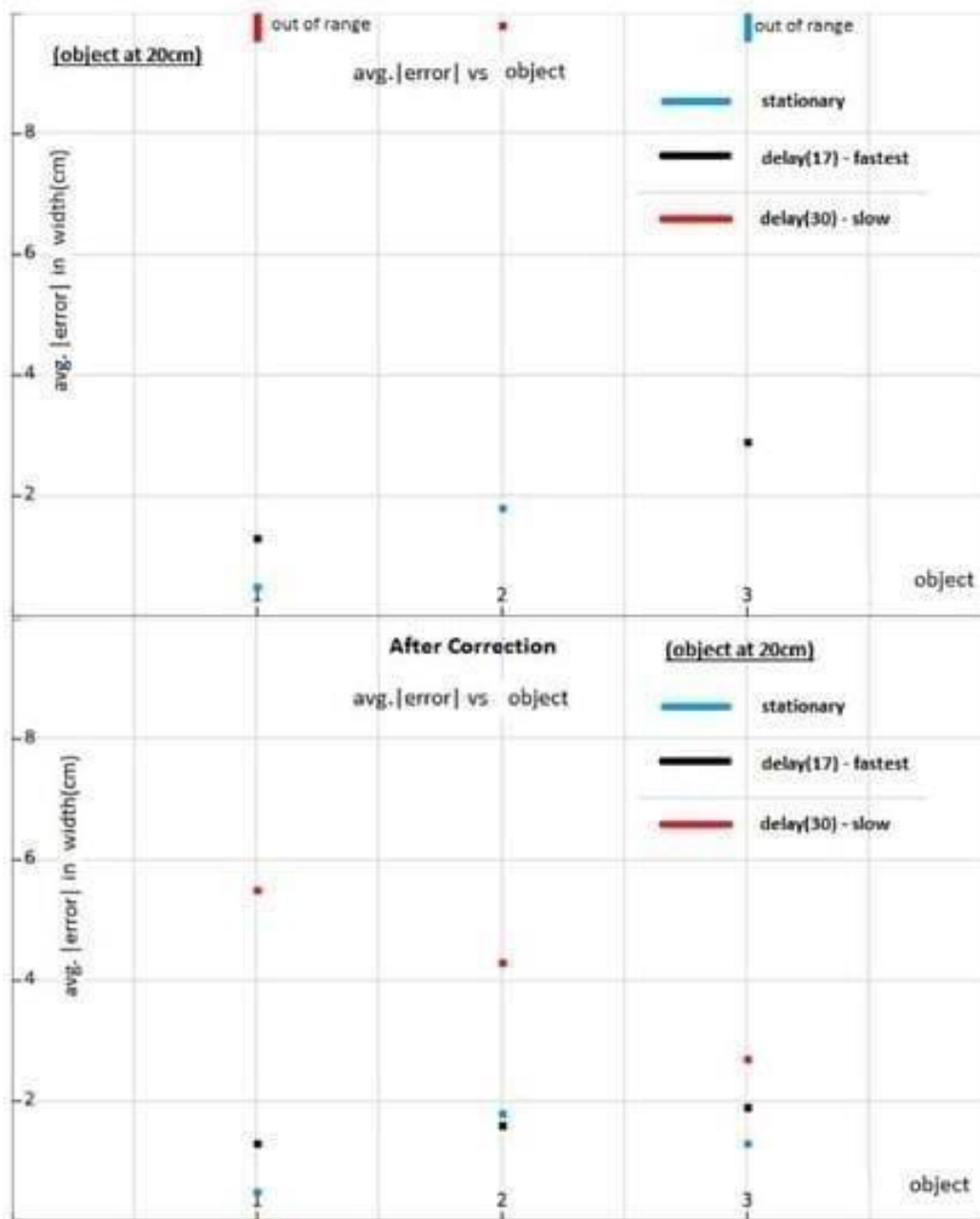


FIGURE 5.10: Object wise -avg | error | in width at x=20c, before vs. after correction

### **Chapter-6: Conclusion.**

We continue to observe that there are several challenges, some unexplainable, in obtaining a stable and accurate reading using the HC- SR04 ultrasonic sensor. After we curbed some of the error, we tried to understand what more could've possibly gone wrong with the algorithm/circuitry. This lead us to raise questions at the depth of its internal working. which was a dead end, as we did not have any understanding of the complicated circuitry in it.

We conclude by stating that the HC-SR04 Ultrasonic Sensor is a good device for detecting objects, but measurements taken from it can at best be suggestive, but not conclusive of the data about present objects/obstacle.

We believe that for best results, especially in a robust application, this device must be used for detecting "presence of objects and so far as to get a fair idea, the data that it provides. But, to measure exact distances of objects lying in front of a sensor, especially if in motion, we must use a different type of sensor, possibly a camera, and even a depth sensor like devices running on Augmented Reality incorporate.

## Rader System