Developing Radar Application Using Arduino UNO

ABSTARCT

This project focuses on developing a radar system using an Arduino Uno microcontroller, an ultrasonic sensor, and a servo motor. The ultrasonic sensor, mounted on a servo motor, can rotate horizontally to scan the surrounding area. The sensor emits sound waves and receives the echo reflected from objects, allowing the system to calculate distances based on the time taken for the echo to return. These distance measurements are then processed by the Arduino and visualized using a Processing application, which displays the data on a radar screen. This application has potential uses in various fields such as robotics, security systems, and obstacle detection for autonomous vehicles. The project demonstrates the integration of hardware components with software to create a functional radar system, providing valuable insights into the practical applications of ultrasonic sensing technology.

LIST OF FIGURES

Figure 1. Effect of Surface Dimension on object detection. 10

Figure 4.1 Architectural View.. .14

Figure 4.2.1 Flow Chart...15

Figure 4.2.2 Development life cycle of Radar System. 16

Figure 4.3 UML Class Diagram..17

Figure 4.4.1 UI Design...18

Figure 4.4.2 UI Implementation for the mapping interface. 18

Figure 4.5 Hardware Design...19

Figure 4.6.1 Arduino Uno...20

Figure 4.6.2.1 Ultrsonic Sensor.. .21

Figure 4.6.2.2 Ultrsonic Sensor datasheet.. .22

Figure 4.6.3.1 Servo Motor...23

Figure 4.6.3.2 Servo Motor Datasheet..23

Figure 4.6.4.LCD I2C...25

Figure 6.1.1 Testing 1...38

Figure 6.1.1 Testing 1...38

Figure 6.2 Observing Results..39

CONTENTS

Page

ABSTRACT.....3

LIST OF FIIGURES.....4

CHAPTER ONE

INTRODUCTION.....7

1.1 Background Information.....7

1.2 Problem Definition.....7

1.4 Goal/Contribution.....8

CHAPTER TWO

LITERATURE REVIEW.....9

2.1 Related Works.....9

CHAPTER THREE

REQUIREMENTS/REQUIREMENT ENGINEERING.....11

3.1 Functional Requirements.....11

3.2 Non-Functional Requirements.....12

CHAPTER FOUR

DESIGN and COMPONENTS.....14

4.1 Architectural View.....14

4.2 Flowchart.....15

4.3 UML Class Diagram.....17

4.4 UI Design.....18

4.5 Hardware Design.....19

4.6 Components.....20

CHAPTER FIVE

INTEGRATION and IMPLEMENTATION..26

5.1 Integration Components..26

5.2 Implementation 26

CHAPTER SIX

TEST AND RESULT 37

6.1 Tests 37

6.1 Results 37

CHAPTER SEVEN

CONCLUSION 37

7.1 Conclusion 37

REFERENCES..38

CHAPTER ONE

INTRODUCTION

1.1 Background Information

Radar systems have long been utilized in various industries for object detection, ranging from aviation and maritime navigation to security and robotics. Traditional radar systems are often complex and expensive, limiting their accessibility for smaller-scale applications and educational purposes. With advancements in microcontroller technology and affordable sensor components, it is now possible to develop cost-effective radar systems for a variety of uses. This project leverages the Arduino Uno microcontroller, an ultrasonic sensor, and a servo motor to create a simple yet functional radar system. This system demonstrates how modern technologies can be employed to build accessible and practical solutions for real-world problems.

1.2 Problem Definition

The primary challenge addressed by this project is the development of an affordable and easy-to-implement radar system capable of detecting and measuring the distance of objects. Many existing radar solutions are either too expensive or too complex for hobbyists, students, and small-scale developers. Additionally, integrating a real-time graphical interface to display the detected objects in an intuitive manner is crucial for usability. This project aims to overcome these obstacles by creating a radar system that is not only cost-effective but also user-friendly, providing accurate distance measurements and visual representation of the surroundings.

1.4 Goal/Contribution

1. **User-Friendly Interface**: The radar system is designed to be simple to set up and use, with a focus on providing a clear and intuitive user interface. The use of a Processing application to visualize the radar data ensures that users can easily interpret the measurements and understand the system's output. This makes the system accessible to a wider audience, including those with limited technical expertise.

2. **Time and Resource Efficiency**: By utilizing affordable and readily available components such as the Arduino Uno, ultrasonic sensor, and servo motor, this project minimizes the cost and resources required to build the radar system. The system's design emphasizes simplicity and efficiency, ensuring that it can be assembled and operated with minimal time and effort.

3. **Environmentally Friendly**: The project promotes sustainability by using low-power components and reducing the need for more resource-intensive traditional radar systems. By making radar technology more accessible and affordable, this project also encourages the development of innovative applications that can contribute to environmental monitoring and protection efforts.

In conclusion, this project demonstrates the potential of integrating affordable microcontroller technology with ultrasonic sensing to create a functional radar system. By addressing the challenges of cost, complexity, and usability, this project makes radar technology accessible to a broader audience and opens up new possibilities for its application in various fields. The system's user-friendly interface, resource efficiency, and environmentally friendly design highlight its contributions to both technological advancement and practical problem-solving.

CHAPTER TWO

LITERATURE REVIEW

2.1 Related Works

A great many applications have been found for the ultrasonic sensing technology since its emergence [7] and some of these include home security systems, robotics applications, distance measurement, tank level measurement, in production lines, and proximity detection applications [4]. These innumerable applications have made it possible to solve technical problems faster and cheaper without compromising safety, quality and stability [5].

The authors in [6] addressed the use of ultra-sonic sensing capability for home security. The system's design is in such a way that the ultra-sonic sensor is mounted on a servo motor to allow a full 3600 rotations so that in any direction an intruder approaches, the sensor would be able to detect the proximity. Once an intruder is detected, the system automatically emits a beeping sound and following this, an SMS message is sent to the home owner, informing him of an intruder in the house using a GSM module.

The authors in [7] designed a system to aid vehicle drivers to navigate blind spots easily in traffic by installing an ultra-sonic sensor on the vehicle's behind and light on the dashboard to indicate the proximity of other commuters that are coming behind. The sensor is calibrated as a range detecting apparatus and a threshold value set. The driver gets a flash of light on the dashboard when oncoming vehicles become closer than this threshold value.

In [8], the authors designed and constructed a navigation application for the visually impaired. The system consisted of an embedded device for detecting local obstacles and an android application for issuing direction to the user verbally. The embedded device uses an ultra-sonic sensor for detecting local obstacles like cars, walls etc. and the data obtained from the ultra-sonic sensor is transmitted to the android application where appropriate decisions are made base on the incoming data and the user is advised accordingly.

The remote monitoring system that was designed and constructed in [9] features the use of ultra-sonic sensors for obstacle avoidance. The system comprises of a remote station (the robot) and a base station (user) who can send commands to the remote station for teleoperation. The remote station comprises of a GPS device, a GSM module, an ultra-sonic sensor-based obstacle avoidance system, a web enabled camera for remote monitoring. Based on the incoming data from the obstacle avoidance system, the GPS system, and the camera, the user is able to send commands to the remote station to alter course or otherwise.

Sound Reflection

For any meaningful measurement of the distance travelled by sound, the sound needs to be reflected because sound is only reflected when the dimension of the reflective surface is large compared to the wavelength of the sound [1].

Figure 1 illustrates the effect that the dimension of the target surface has on measurement.

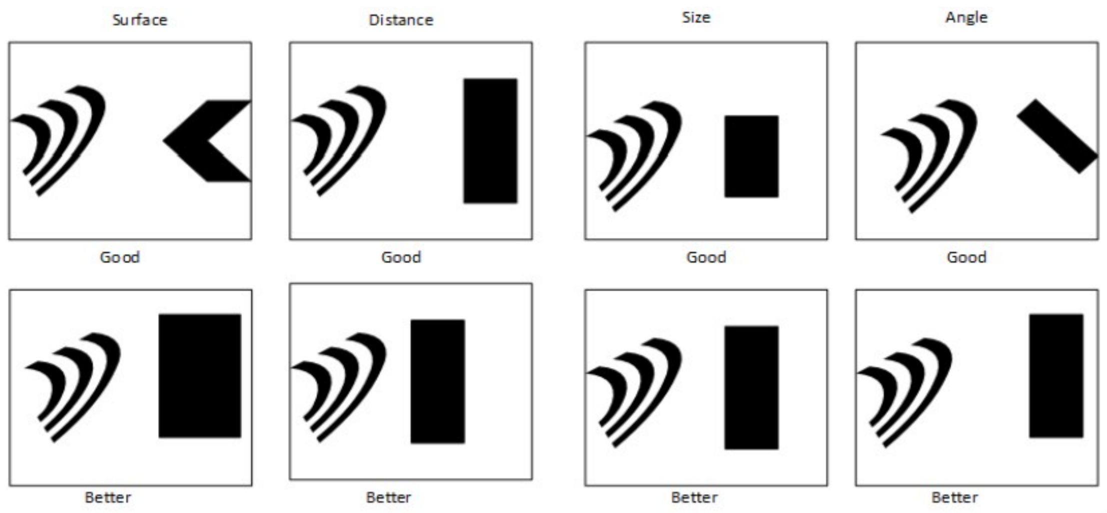


Figure 1. Effect of Surface Dimension on object detection.

Surface

An ideal target surface is hard and smooth. This surface reflects a greater amount of signal than a soft, rough surface. A weak echo is the result of a small or soft object. This reduces the operating distance of an ultrasonic sensor and decreases its accuracy.

Distance

The shorter the distance from the ultrasonic sensor to an object, the stronger the returning echo is. Therefore, as the distance increases, the object requires better reflective characteristics to return a sufficient echo. Angle The inclination of an objects' surface facing the ultrasonic sensor affects how the object reflects. The portion perpendicular to the sensor returns the echo. If the entire object is at a greater angle, the signal is then reflected away from the sensor and no echo is detected [1-2].

Size

A large object has more surface area to reflect the signal than a small one. The surface area recognized as the target is generally the area closest to the sensor.

Applications of Sound Reflection

Ultrasonic radar systems were birthed in the defense industry. Because of its myriad applications in under-water object sensing and identification plus identification and tracking of in-coming missiles and vessels [3], the navy and the air force are major names in the RADAR utilization community. In recent times, applications of this technology have diversified into consumer products such as automatic parking systems, accident prevention systems, and aircraft flight control systems [2]. Another application is in the field of robotics where this technology is now being used by robots as a cheap way to detect, identify and if need be, avoid objects. This is especially effective as robotic home appliances need not have cameras in order to perform their assigned duties like vacuuming, exterminating household pests.

CHAPTER THREE

REQUIREMENTS / REQUIREMENT ENGINEERING

3.1 Functional Requirements

3.1.1 User

Ease of Use: The system must be user-friendly, with an intuitive interface that allows users to easily set up and operate the radar. Users should be able to start the system, view the radar screen, and understand the distance measurements without needing extensive technical knowledge.

Real-Time Data: The radar must provide real-time distance measurements and visual representation of the detected objects. Users should be able to see immediate feedback as the ultrasonic sensor scans the environment.

Adjustable Scan Range: Users should have the ability to adjust the scanning range and angle of the radar to customize the detection area based on their specific needs and environment.

3.1.2 System

Accurate Distance Measurement: The system must accurately measure the distance of objects within the range of the ultrasonic sensor. The accuracy should be sufficient for practical applications, with minimal error in distance calculation.

Data Visualization: The system must process the data collected by the ultrasonic sensor and display it on a radar screen using a Processing application. The visual representation should be clear and easy to interpret.

Hardware and Software Coordination: The system must effectively coordinate the hardware components (Arduino Uno, ultrasonic sensor, and servo motor) with the software components (Arduino code and Processing application) to ensure smooth operation and data synchronization.

3.2 Non-Functional Requirements

Performance: The radar system should operate efficiently, with minimal lag or delay in data processing and visualization. The real-time feedback loop should be maintained to ensure accurate and timely information for the user.

Reliability: The system must be reliable and capable of continuous operation without frequent failures or interruptions. It should be able to handle various environmental conditions without significant performance degradation.

Scalability: The system should be designed in a way that allows for easy scalability. Additional sensors or enhancements can be integrated into the system without requiring major modifications to the existing setup.

Maintainability: The system should be easy to maintain, with clear documentation and accessible components. Users should be able to troubleshoot and replace parts as needed without extensive effort.

Portability: The system should be compact and portable, allowing users to easily transport and deploy it in different locations as needed. The design should ensure that all components are securely integrated and easy to assemble/disassemble.

Energy Efficiency: The system should be energy-efficient, utilizing low-power components to minimize power consumption. This is particularly important for applications where the radar system needs to operate for extended periods without access to a power source.

CHAPTER FOUR

DESIGN AND COMPONENTS

4.1 Architectural View

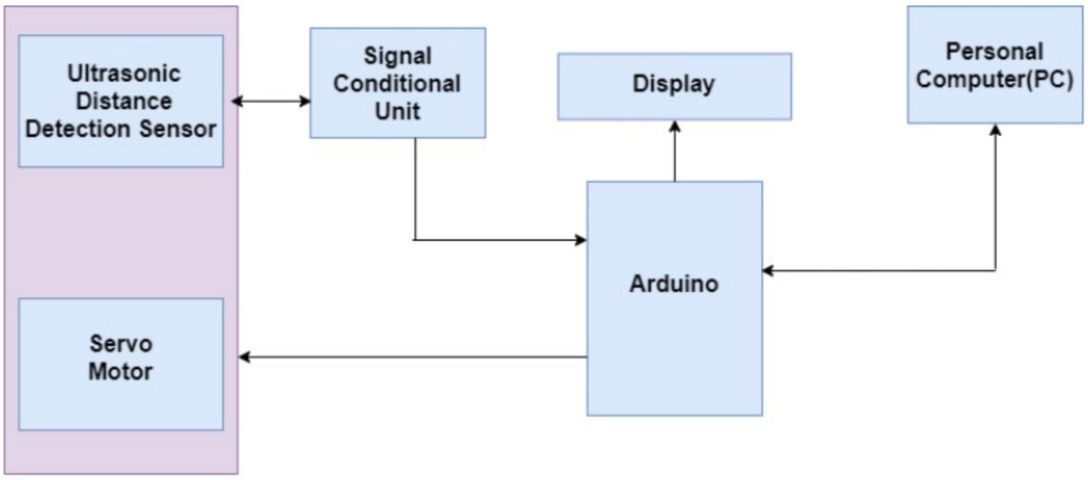


Figure 4.1 Architectural View

The architectural structure of the project will be as shown in the figure.

4.2 FLOW CHART

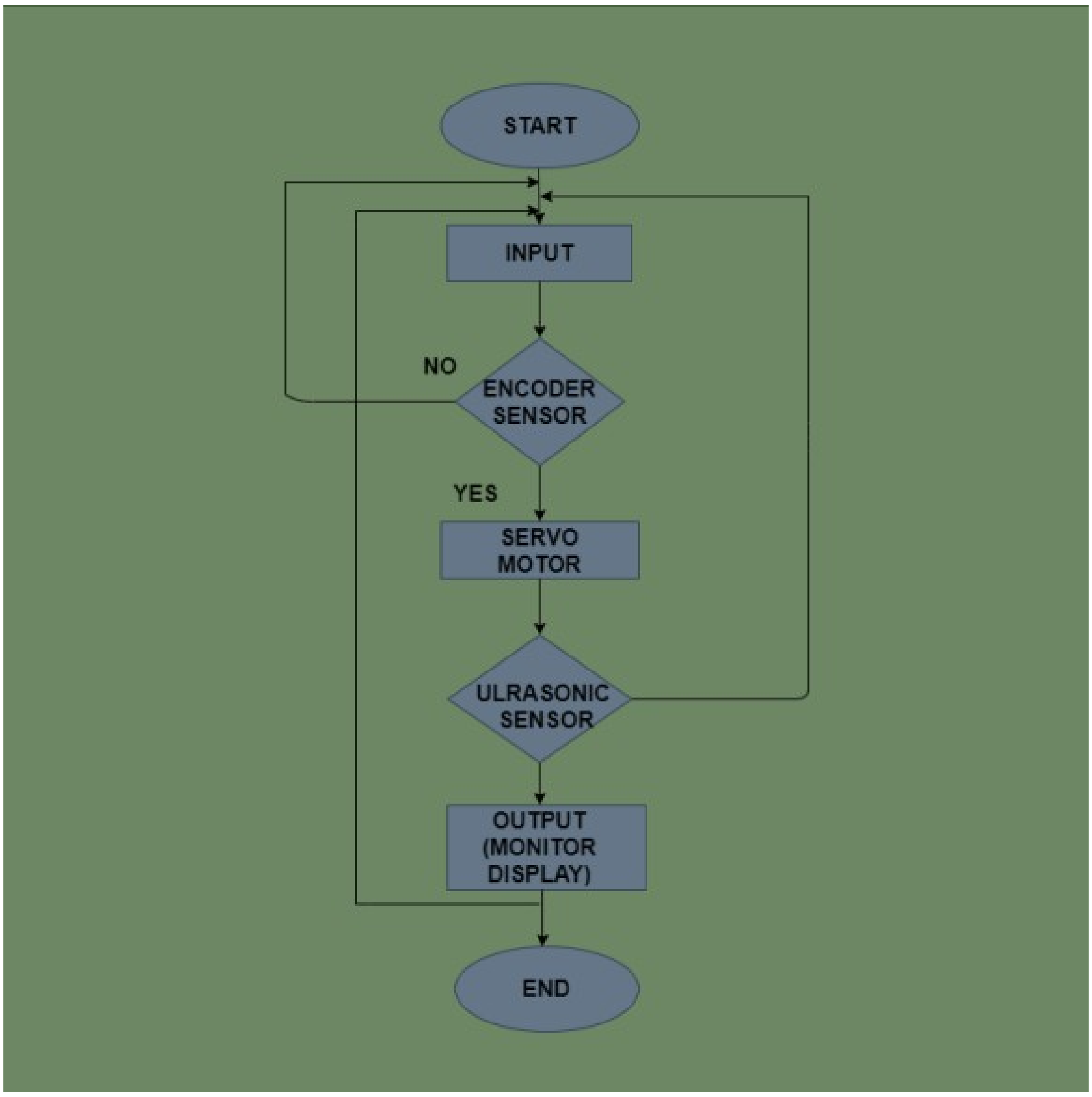


Figure 4.2.1 Flow Chart

As it is known, a flow chart is used to explain a process or to better understand anything. In this flowchart, the process or entire process of our project is explained. This helps in better understanding. Firstly, the process is started with the input provided by the ultrasonic sensor to detect the object, which then supplies the data to the Arduino, providing input to the processing software. The object is detected by rotating the sensor with the help of a servo motor. Next, the process moves to the output stage, where the monitor display shows a graphical representation of the output. This includes the distance, range, and angle of the object detected by the radar. Thus, the flow chart is used to better understand the process.

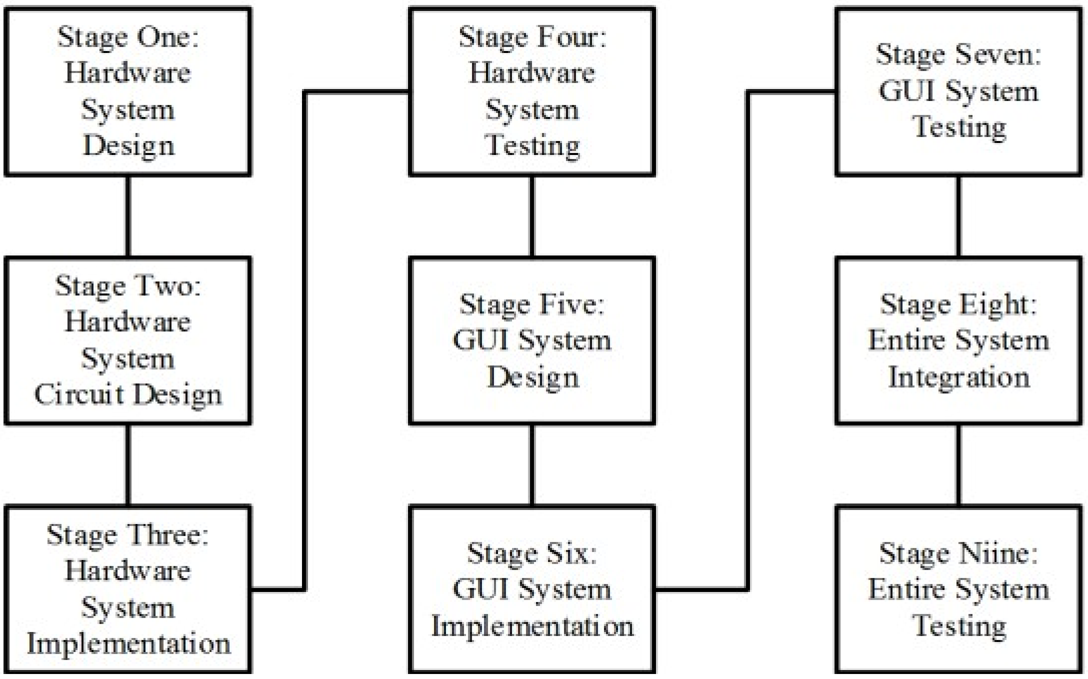

Stage One: Hardware System Design
Stage Two: Hardware System Circuit Design
Stage Three: Hardware System Implementation
Stage Four: Hardware System Testing
Stage Five: GUI System Design
Stage Six: GUI System Implementation
Stage Seven: GUI System Testing
Stage Eight: Entire System Integration
Stage Niine: Entire System Testing

Figure 4.2.2 Development life cycle of Radar System.

4.3 UML Class Diagram

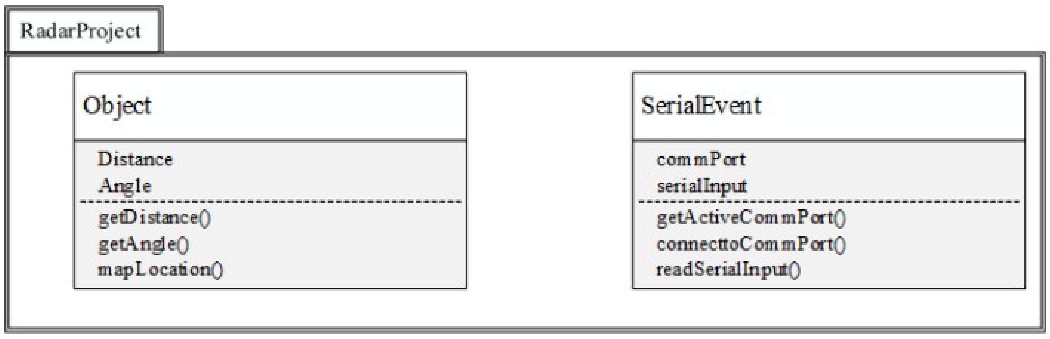


Figure 4.3 UML Class Diagram

In the UML diagram, showing the interaction between two main classes: Object and SerialEvent. The Object class contains attributes Distance and Angle, which store the distance and angle of detected objects, respectively. Methods such as getDistance(), getAngle(), and mapLocation() are included to retrieve these values and map the object's location. The SerialEvent class includes attributes commPort and serialInput, which hold communication port information and serial input data. Methods like getActiveCommPort(), connectToCommPort(), and readSerialInput() are provided to manage the communication port and read data from it. This diagram illustrates the structure and interaction of components within the radar project, focusing on object detection and serial communication.

4.4 UI Design

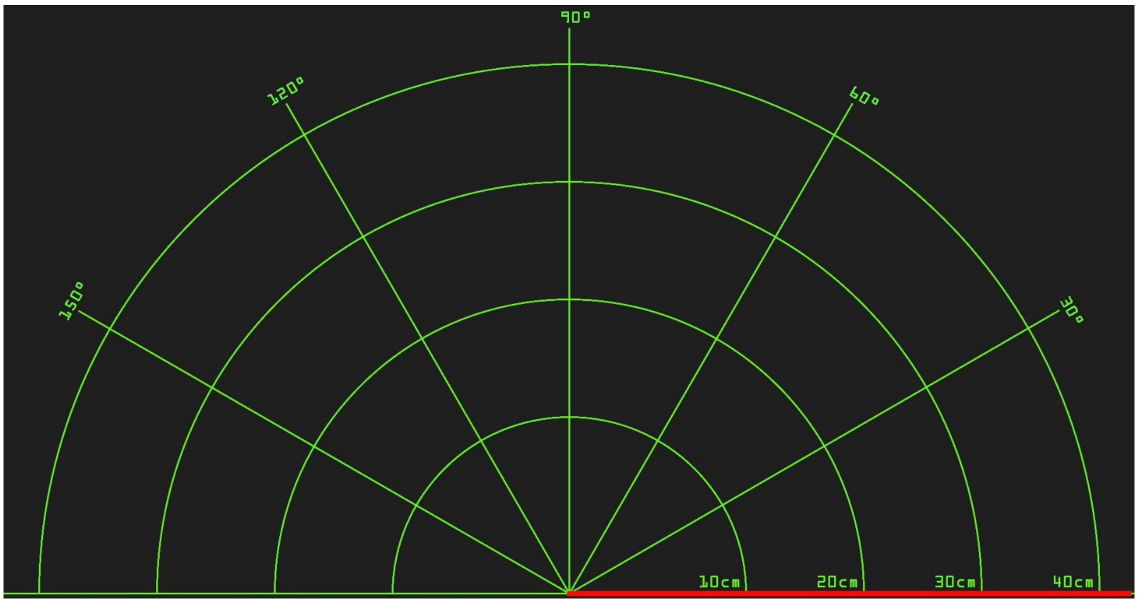


Figure 4.4.1 UI Design

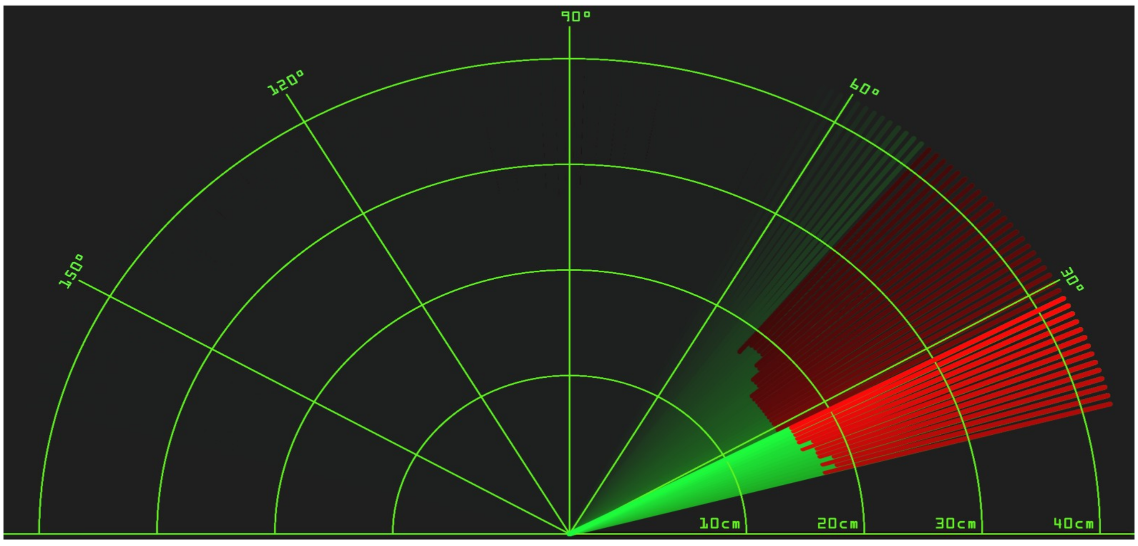


Figure 4.4.2 UI Implementation for the mapping interface.

The GUI is built using the JAVA programming language and the package consist of two classes that are represented in the UML class diagram in Figure 4.3. The final GUI is depicted in figure 4.4.2 and a line sweeps from east to west and back and at each point where the ultra-sonic sensor detects an obstacle, a smudge is printed on the GUI.

4.5 Hardware Design

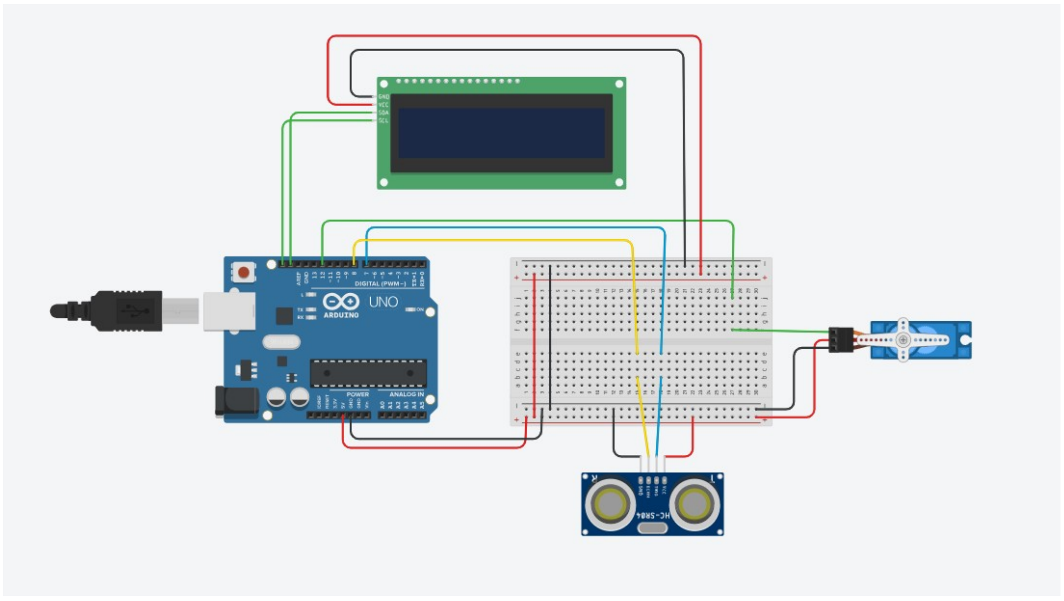


Figure 4.5 Hardware Design

This hardware design created in Tinkercad shows a radar system using an Arduino Uno, an ultrasonic sensor (HC-SR04), a servo motor, and a 16x2 I2C LCD display. The ultrasonic sensor is connected to the Arduino with the trigger pin (Trig) on pin 10 and the echo pin (Echo) on pin 11. The servo motor is attached to pin 12 of the Arduino, allowing it to rotate and scan the environment. The I2C LCD display is connected to the Arduino via the I2C bus, with the SDA and SCL pins connected to the corresponding pins on the Arduino (A4 and A5). The setup includes a breadboard for power distribution and connections, providing a clear and organized layout for the project components. This design allows the radar to measure distances and display the angle and distance of detected objects on the LCD screen.

4.6 Components

4.6.1 ARDUINO UNO: The Arduino Uno is a microcontroller board based on the ATmega328. It has 32k Byte in system programmable flash, 14 digital I/O pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack and a reset button. Arduino is an open-source platform used for constructing and programing of electronics. It can receive and send information to most devices, and even through the internet to command the specific electronic device. It uses a hardware called Arduino Uno circuit board. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. "Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards.

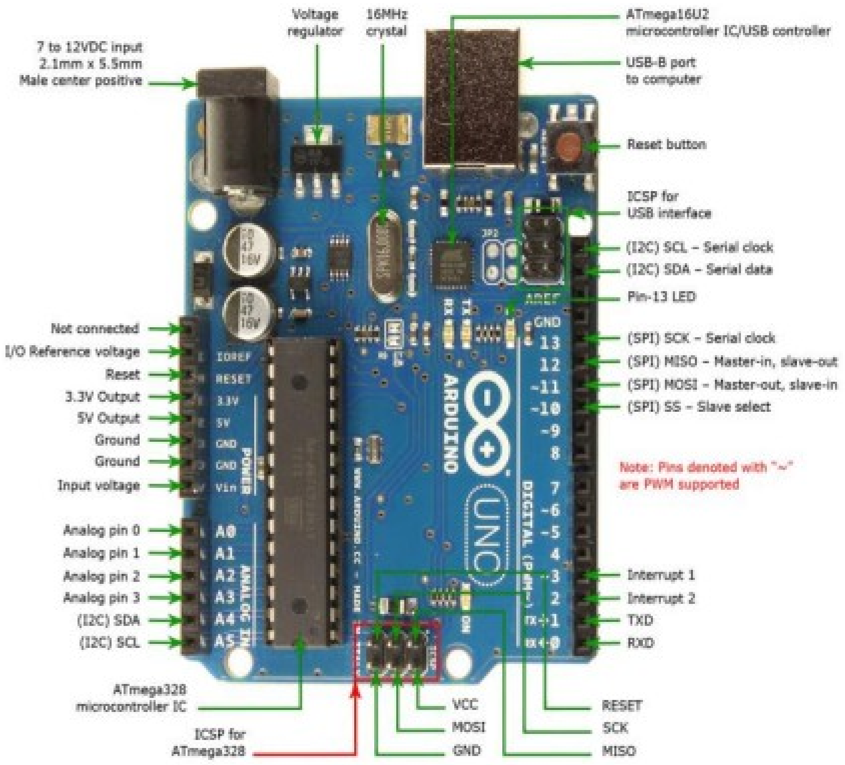


Figure 4.6.1 Arduino Uno

4.6.2 ULTRASONIC SENSOR: Ultrasonic sensors are based on the measurement of the properties of acoustic waves with frequencies above the human audible range often at roughly 40kHz. Three different properties of the received echo pulse may be evaluated for different sensing Purposes: 1)Time of flight, 2)Doppler shift, 3)Amplitude attenuation. Ultrasonic ranging module HCSR04 provides 2cm-400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The ultrasonic sensor works using trigger and echo method. The transceiver module triggers and sends the signal to the water the water sends back an echo signal which is read by the echo i.e. the receiver module. The Ultra sonic sensor calculated distance of the signal and returns the level of the water. The travel time value and the speed value allow the sensor to calculate the level of the water. The figure below is the image of the ultrasonic sensor used in the project.

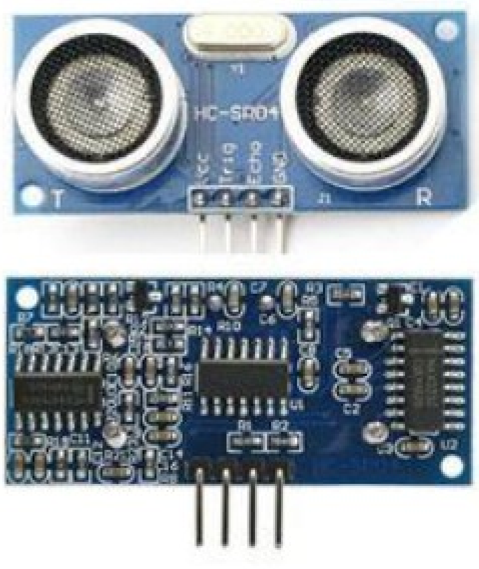


Figure 4.6.2.1 Ultrsonic Sensor

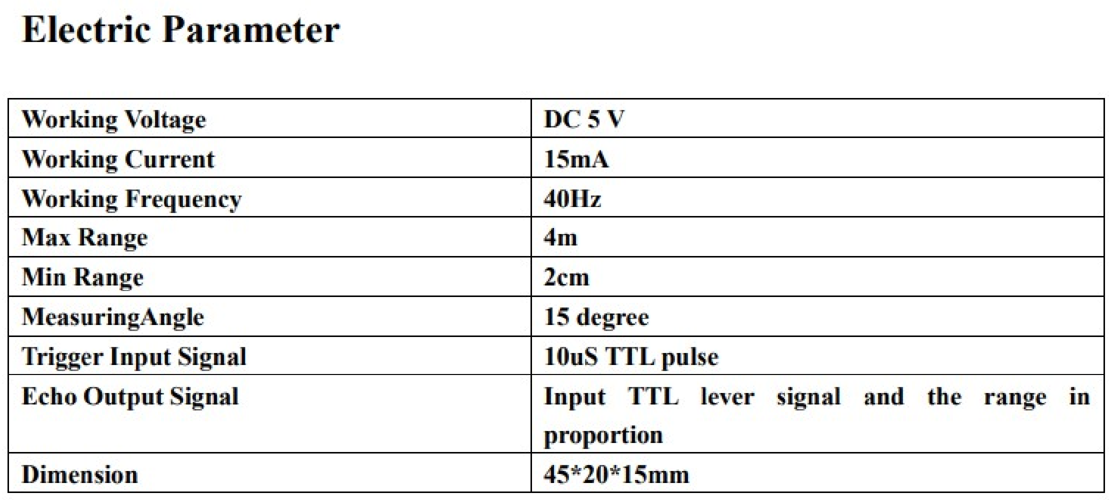
Wire connecting direct as following:

5V Supply

Trigger Pulse Input

Echo Pulse Output

0V Ground



Electric Parameter

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm

Figure 4.6.2.2 Ultrsonic Sensor datasheet

4.6.3 SERVO MOTOR: A servo motor is an electrical device which can push or rotate an object with great precision. If you want to rotate and object at some specific angles or distance, then you use servo motor. It is just made up of simple motor which run through servo mechanism. Servo Motor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. Servos are controlled by sending an electrical pulse of variable width, or pulse width modulation [PWM] through the control wire. Servo motors have three wires: power, ground, and signal. If motor is used is DC powered then it is called DC servo motor, and if it is AC powered motor then it is called AC servo motor. The position of servo motor is decided by electrical pulse and its circuitry is placed beside the motor.

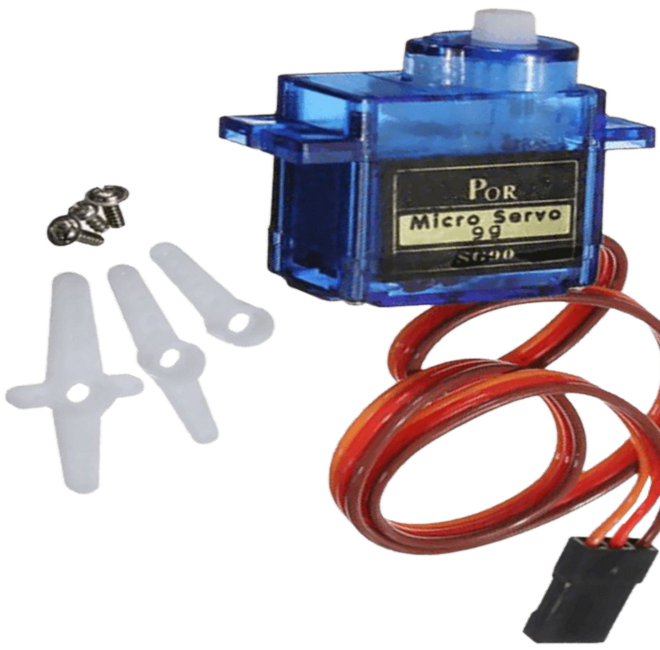
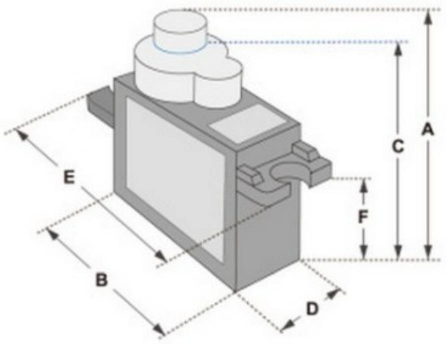
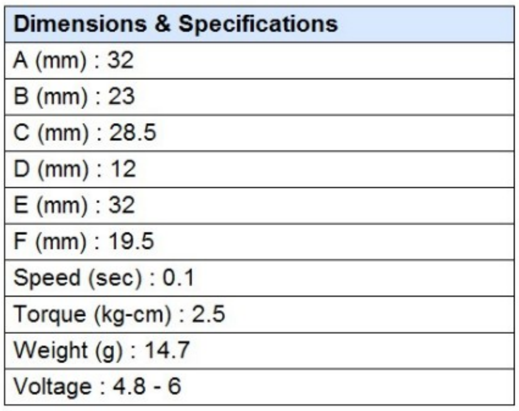


Figure 4.6.3.1 Servo Motor





Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

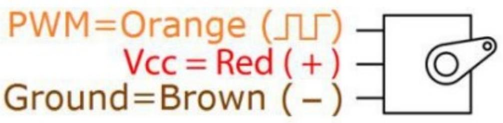


Figure 4.6.3.2 Servo Motor Datasheet

4.6.4 LCD(I2C): An LCD (Liquid Crystal Display) is a flat-panel display technology commonly used in digital watches, calculators, and many portable computers. LCDs operate by using liquid crystals that modulate light. When voltage is applied, the liquid crystals align in a way that affects the light passing through them, thus creating visible images. In the context of microcontroller projects, a common type of LCD used is the 16x2 character display. This particular display can show 16 characters per line and has two such lines. It is used to display alphanumeric characters and symbols, making it ideal for projects where visual feedback is necessary, such as temperature readouts, menu options, or, as in this case, the distance and angle of detected objects in a radar system.

I2C (Inter-Integrated Circuit) is a multi-master, multi-slave, packet-switched, single-ended, serial communication bus. It is used to connect low-speed peripherals to processors and microcontrollers in short-distance, intra-board communication. The I2C bus was developed by Philips Semiconductor (now NXP Semiconductors). I2C uses only two bidirectional lines, Serial Data Line (SDA) and Serial Clock Line (SCL), along with a ground connection, to communicate between devices. Each device connected to the bus is software addressable by a unique address and can operate as either a transmitter or receiver, depending on the device's functionality. In microcontroller projects, I2C is often used for connecting sensors, displays, and other peripherals, allowing efficient communication with minimal wiring. In this radar project, the I2C interface simplifies the connection to the LCD, reducing the number of pins required and allowing for easier integration with other components.

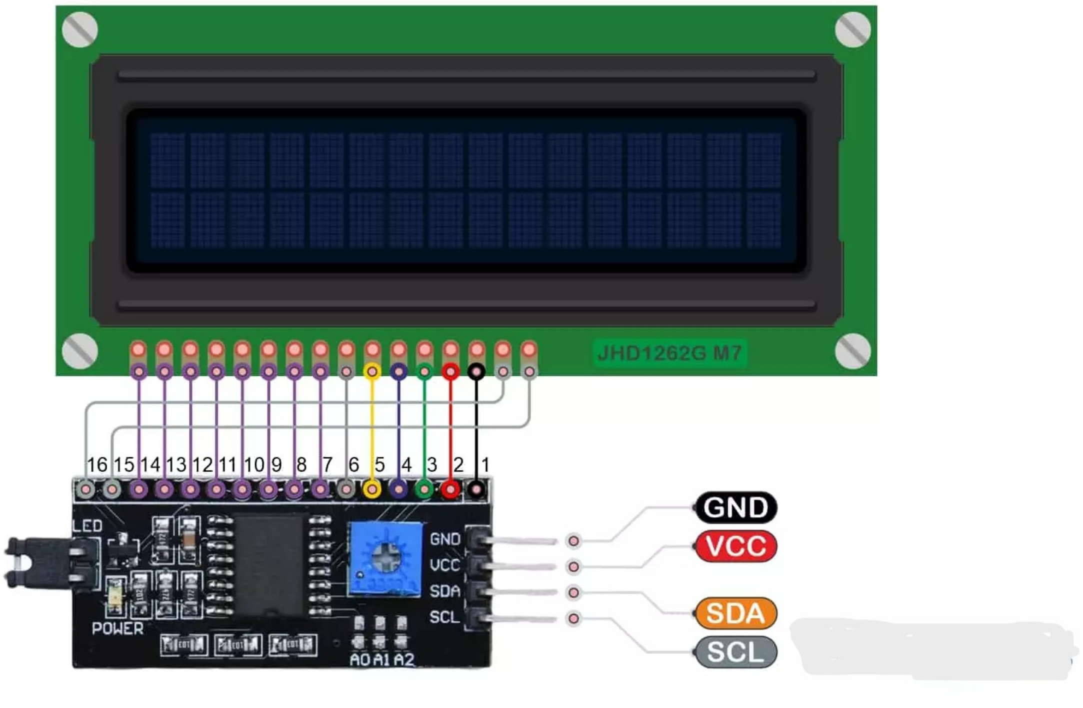


Figure 4.6.4.LCD I2C

4.6.5 Other Components:

Breadboard: A breadboard is a fundamental tool used in electronics prototyping. It's a rectangular board with a grid of holes arranged in rows and columns, typically with metal clips beneath the holes to connect components inserted into them. The holes are used to insert electronic components such as resistors, capacitors, integrated circuits (ICs), and other devices.

Jumper Cables: Jumper cables, also known as jumper wires or simply jumpers, are flexible wires with connectors at each end used to create electrical connections between components on a breadboard or other electronic circuit boards. They are essential for creating circuits without soldering, allowing components to be easily moved or replaced during prototyping. Jumper cables come in various lengths, colors, and connector types (like male-to-male, male-to-female, and female-to-female) to suit different circuit designs and configurations.

CHAPTER FIVE

INTEGRATION and IMPLEMENTATION

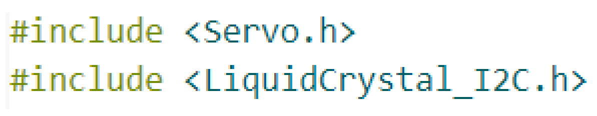
5.1 Integrating Components

In this circuit, an Arduino Uno is used to control an LCD display, a servo motor, and an ultrasonic sensor (HC-SR04). The 5V and GND pins on the Arduino are connected to the power and ground rails on the breadboard, respectively. The LCD display, equipped with an I2C adapter, connects its VCC and GND pins to the 5V and GND rails, while the SDA and SCL pins connect to the A4 and A5 pins on the Arduino. The HC-SR04 sensor's VCC and GND pins are connected to the 5V and GND rails, with its Trig and Echo pins connected to digital pins 9 and 10 on the Arduino. The servo motor's power and ground wires are also connected to the 5V and GND rails, with its signal wire connected to digital pin 11 on the Arduino. The HC-SR04 sensor and servo motor connected each other with ultrasonic sensor holder. This configuration ensures that all components are powered correctly and can communicate with the Arduino for proper operation.

5.2 Implementation

5.2.1 Arduino Coding

5.2.1.1 Include Libraries

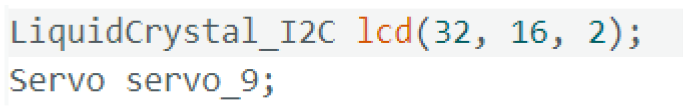


```
#include <Servo.h>
#include <LiquidCrystal_I2C.h>
```

The Servo.h library is included to control the servo motor.

The LiquidCrystal_I2C.h library is included to control the I2C LCD display.

5.2.1.2 LCD and Servo Initialization

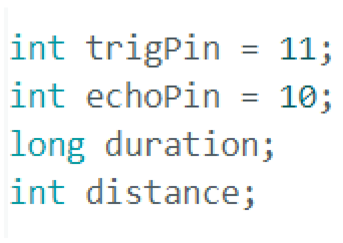


```
LiquidCrystal_I2C lcd(32, 16, 2);
Servo servo_9;
```

An object lcd of the LiquidCrystal_I2C class is created with the specified I2C address, LCD width, and height.

An object servo_9 of the Servo class is created to control the servo motor.

5.2.1.3 Pin Definitions and Variables

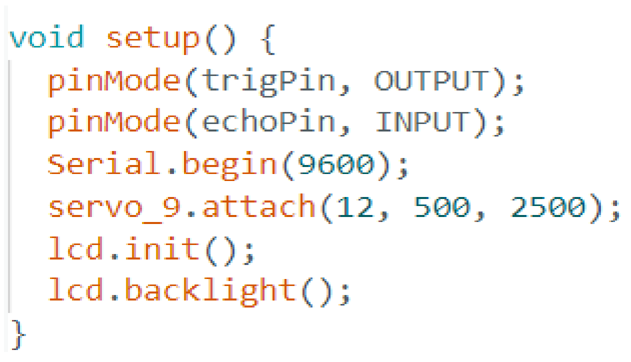


```
int trigPin = 11;
int echoPin = 10;
long duration;
int distance;
```

“trigPin” and “echoPin” are defined to connect to the ultrasonic sensor.

Variables duration and distance are declared to store the pulse duration and calculated distance, respectively.

5.2.1.4 Setup Function



```
void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);
  servo_9.attach(12, 500, 2500);
  lcd.init();
  lcd.backlight();
}
```

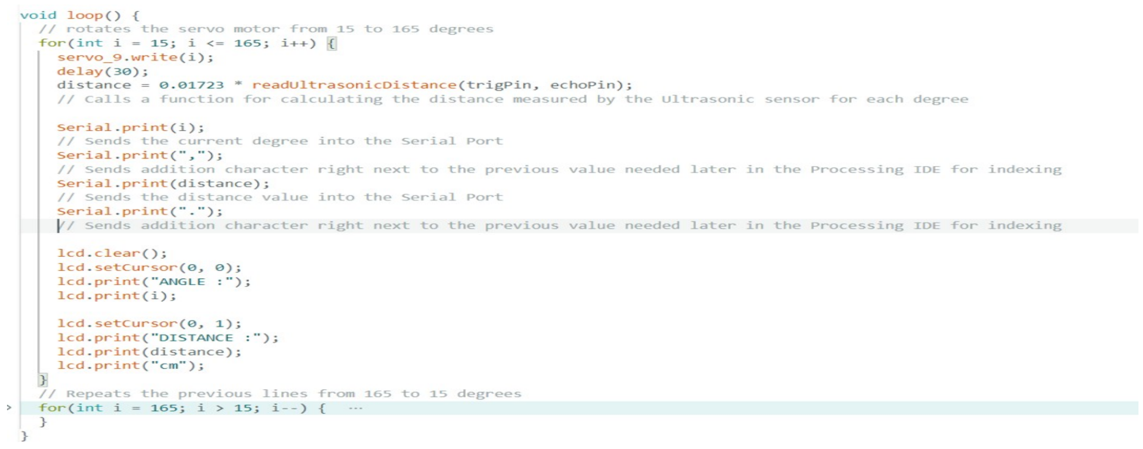
trigPin is set as an output and echoPin as an input.

The serial communication is initialized at 9600 baud rate.

The servo motor is attached to pin 12 with minand max pulse widths specified.

The lcd.init() and lcd.backlight() lines are typically used to initialize and turn on the LCD backlight.

5.2.1.5 Main Loop



```
void loop() {
  // rotates the servo motor from 15 to 165 degrees
  for(int i = 15; i <= 165; i++) {
    servo_9.write(i);
    delay(30);
    distance = 0.01723 * readUltrasonicDistance(trigPin, echoPin);
    // Calls a function for calculating the distance measured by the Ultrasonic sensor for each degree

    Serial.print(i);
    // Sends the current degree into the Serial Port
    Serial.print(",");
    // Sends addition character right next to the previous value needed later in the Processing IDE for indexing
    Serial.print(distance);
    // Sends the distance value into the Serial Port
    Serial.print(".");
    // Sends addition character right next to the previous value needed later in the Processing IDE for indexing

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("ANGLE :");
    lcd.print(i);

    lcd.setCursor(0, 1);
    lcd.print("DISTANCE :");
    lcd.print(distance);
    lcd.print("cm");
  }
  // Repeats the previous lines from 165 to 15 degrees
  for(int i = 165; i > 15; i--) {  ···
  }
}
```

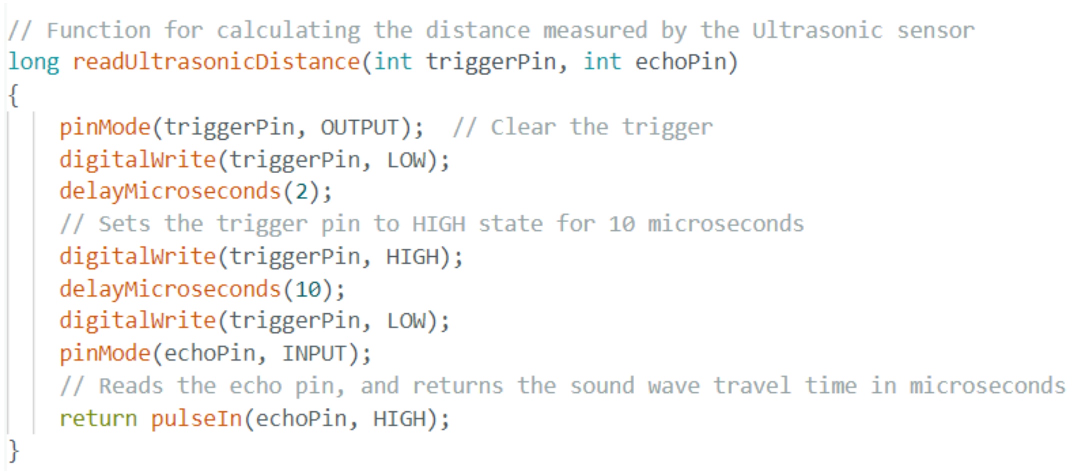
The servo motor is rotated from 15 to 165 degrees in increments of 1 degree.

For each degree, the ultrasonic distance is measured using the readUltrasonicDistance function.

The angle and distance are printed to the serial monitor and displayed on the LCD.

The servo motor then rotates back from 165 to 15 degrees, repeating the measurements and displays.

5.2.1.6 Ultrasonic Distance Measurement Function



```
// Function for calculating the distance measured by the Ultrasonic sensor
long readUltrasonicDistance(int triggerPin, int echoPin)
{
    pinMode(triggerPin, OUTPUT);  // Clear the trigger
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    // Sets the trigger pin to HIGH state for 10 microseconds
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
    pinMode(echoPin, INPUT);
    // Reads the echo pin, and returns the sound wave travel time in microseconds
    return pulseIn(echoPin, HIGH);
}
```

The readUltrasonicDistance function measures the time taken for an ultrasonic pulse to travel to an object and back.

The trigPin is set to output a pulse for 10 microseconds.

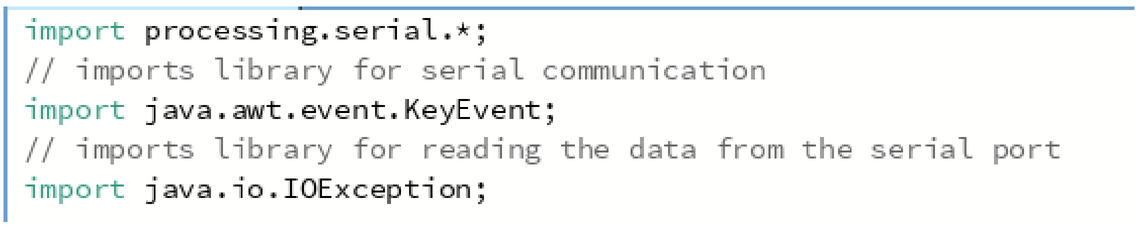
The echoPin is set to input to read the time it takes for the pulse to return.

The function returns the pulse duration in microseconds.

This code allows the Arduino to control a servo motor, measure distances using an ultrasonic sensor, and display the results on an LCD and serial monitor.

5.2.2 Processing Coding

5.2.2.1 Import Libraries



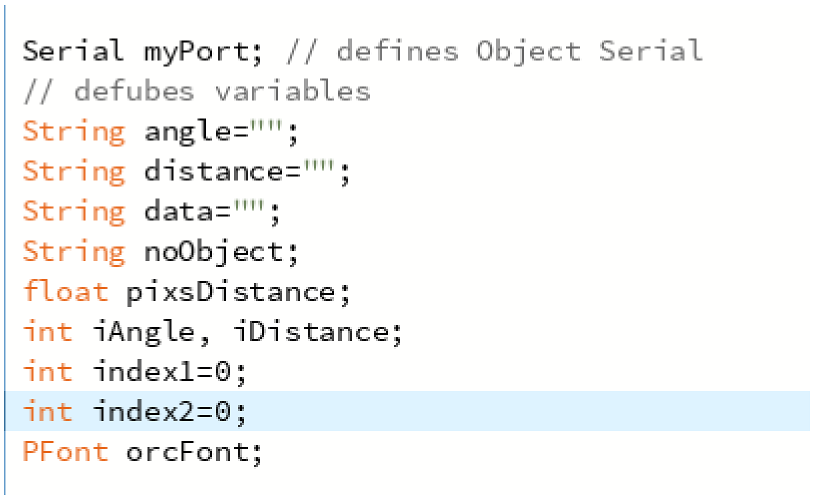
```
import processing.serial.*;
// imports library for serial communication
import java.awt.event.KeyEvent;
// imports library for reading the data from the serial port
import java.io.IOException;
```

The processing.serial.* library is imported to enable serial communication.

The java.awt.event.KeyEvent library is imported to handle key events.

The java.io.IOException library is imported to handle input/output exceptions.

5.2.2.2 Variable Declarations



```
Serial myPort; // defines Object Serial
// defubes variables
String angle="";
String distance="";
String data="";
String noObject;
float pixsDistance;
int iAngle, iDistance;
int index1=0;
int index2=0;
PFont orcFont;
```

A Serial object named myPort is defined for serial communication.

Several String variables are declared to store data received from the serial port.

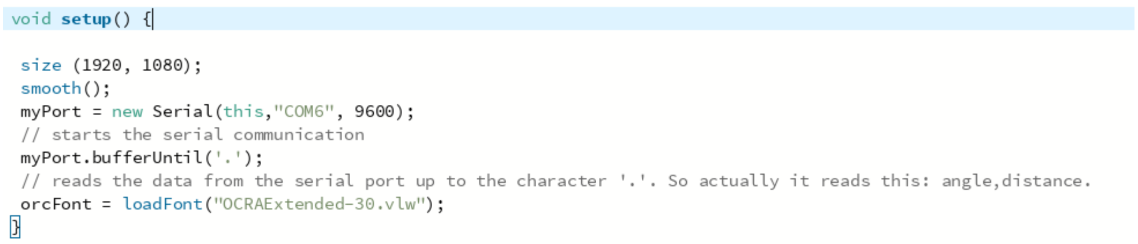
pixsDistance is declared to store the distance in pixels.

iAngle and iDistance are declared to store the converted integer values of the angle and distance.

index1 and index2 are declared for indexing purposes.

orcFont is declared to store the custom font used in the display.

5.2.2.3 Setup Function



```
void setup() {

 size (1920, 1080);
 smooth();
 myPort = new Serial(this,"COM6", 9600);
 // starts the serial communication
 myPort.bufferUntil('.');
 // reads the data from the serial port up to the character '.'. So actually it reads this: angle,distance.
 orcFont = loadFont("OCRAExtended-30.vlw");
}
```

The size() function sets the size of the display window to 1920x1080 pixels.

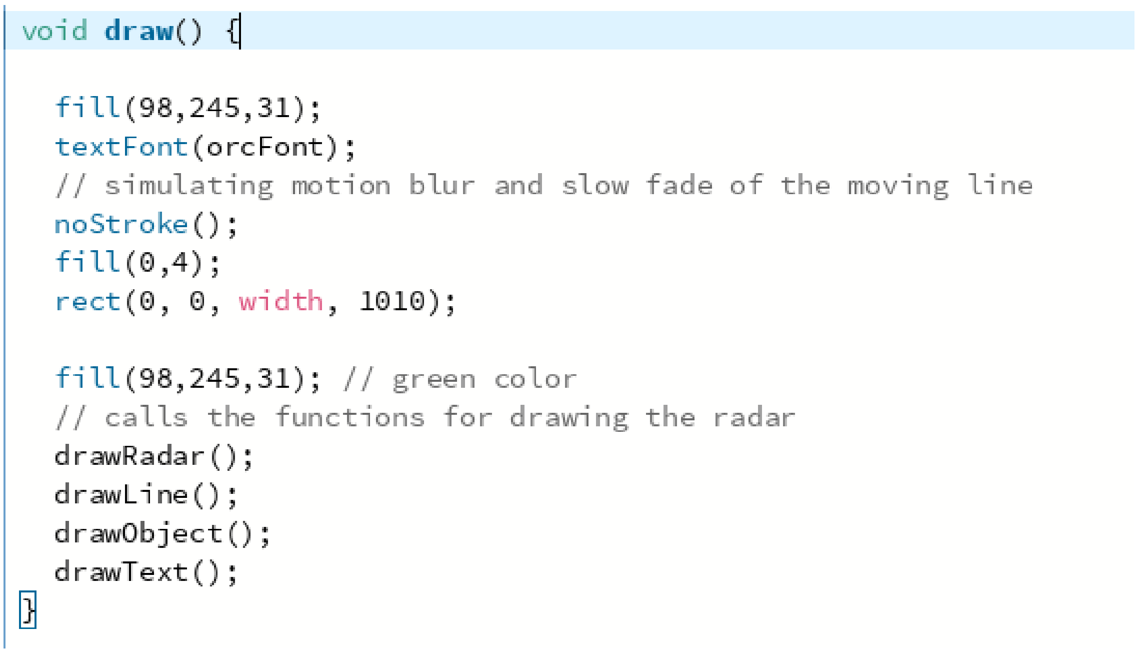
The smooth() function is called to enable smooth drawing.

The Serial object myPort is initialized to communicate on port "COM6" at a baud rate of 9600.

The bufferUntil('.') function is called to read data from the serial port until the character '.' is encountered.

The orcFont variable is initialized with the custom font "OCRAExtended-30.vlw".

5.1.2.4 Draw Function



```
void draw() {

  fill(98,245,31);
  textFont(orcFont);
  // simulating motion blur and slow fade of the moving line
  noStroke();
  fill(0,4);
  rect(0, 0, width, 1010);

  fill(98,245,31); // green color
  // calls the functions for drawing the radar
  drawRadar();
  drawLine();
  drawObject();
  drawText();
}
```

The draw() function continuously executes the code within it, creating an animation.

The fill(98,245,31) function sets the fill color to green.

The textFont(orcFont) function sets the font to the custom font loaded earlier.

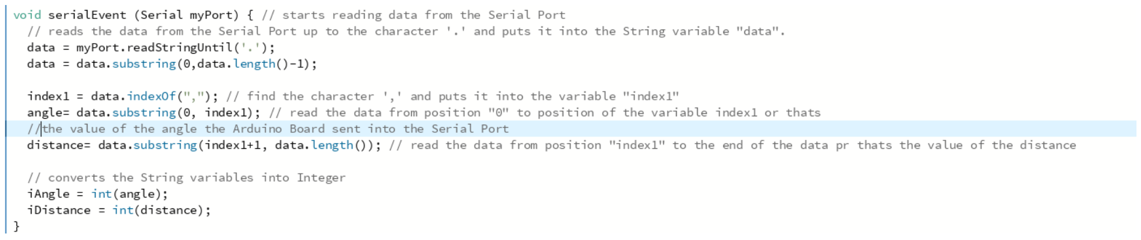
noStroke() disables the drawing of outlines around shapes.

fill(0,4) sets the fill color to almost transparent black.

rect(0, 0, width, 1010) draws a rectangle to create a motion blur effect.

The functions drawRadar(), drawLine(), drawObject(), and drawText() are called to draw the radar, moving line, detected object, and text on the screen.

5.1.2.5 Serial Event Function



```
void serialEvent (Serial myPort) { // starts reading data from the Serial Port
  // reads the data from the Serial Port up to the character '.' and puts it into the String variable "data".
  data = myPort.readStringUntil('.');
  data = data.substring(0,data.length()-1);

  index1 = data.indexOf(","); // find the character ',' and puts it into the variable "index1"
  angle= data.substring(0, index1); // read the data from position "0" to position of the variable index1 or thats
  //the value of the angle the Arduino Board sent into the Serial Port
  distance= data.substring(index1+1, data.length()); // read the data from position "index1" to the end of the data pr thats the value of the distance

  // converts the String variables into Integer
  iAngle = int(angle);
  iDistance = int(distance);
}
```

The serialEvent() function is called whenever data is received from the serial port.

The myPort.readStringUntil('.') function reads data from the serial port until the character '.' is encountered and stores it in the data variable.

The data = data.substring(0,data.length()-1) line removes the last character ('.') from the data string.

The index1 = data.indexOf(",") line finds the position of the comma in the data string.

The angle variable is set to the substring of data from the beginning to the position of the comma.

The distance variable is set to the substring of data from the position after the comma to the end of the string.

The angle and distance strings are converted to integers and stored in the iAngle and iDistance variables, respectively.

5.1.2.6 Draw Radar Function

```
void drawRadar() {
  pushMatrix();
  translate(960,1000); // moves the starting coordinats to new location
  noFill();
  strokeWeight(2);
  stroke(98,245,31);
  // draws the arc lines
  arc(0,0,1800,1800,PI,TWO_PI);
  arc(0,0,1400,1400,PI,TWO_PI);
  arc(0,0,1000,1000,PI,TWO_PI);
  arc(0,0,600,600,PI,TWO_PI);
  // draws the angle lines
  line(-960,0,960,0);
  line(0,0,-960*cos(radians(30)),-960*sin(radians(30)));
  line(0,0,-960*cos(radians(60)),-960*sin(radians(60)));
  line(0,0,-960*cos(radians(90)),-960*sin(radians(90)));
  line(0,0,-960*cos(radians(120)),-960*sin(radians(120)));
  line(0,0,-960*cos(radians(150)),-960*sin(radians(150)));
  line(-960*cos(radians(30)),0,960,0);
  popMatrix();
}
```

The drawRadar() function draws the radar display.

The pushMatrix() function saves the current transformation matrix.

The translate(960,1000) function moves the origin of the coordinate system to (960, 1000).

The noFill() function disables filling shapes.

The strokeWeight(2) function sets the thickness of the lines.

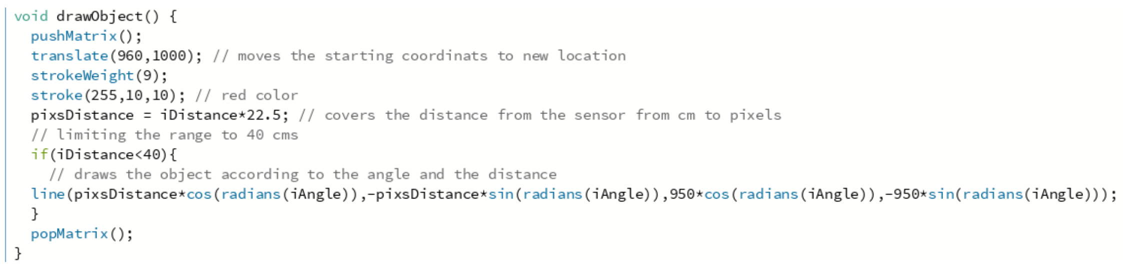
The stroke(98,245,31) function sets the stroke color to green.

The arc() functions draw arc lines to represent the radar.

The line() functions draw lines to represent the angles on the radar.

The popMatrix() function restores the previous transformation matrix.

5.1.2.7 Draw Object Function



```
void drawObject() {
  pushMatrix();
  translate(960,1000); // moves the starting coordinats to new location
  strokeWeight(9);
  stroke(255,10,10); // red color
  pixsDistance = iDistance*22.5; // covers the distance from the sensor from cm to pixels
  // limiting the range to 40 cms
  if(iDistance<40){
    // draws the object according to the angle and the distance
  line(pixsDistance*cos(radians(iAngle)),-pixsDistance*sin(radians(iAngle)),950*cos(radians(iAngle)),-950*sin(radians(iAngle)));
  }
  popMatrix();
}
```

The drawObject() function draws the detected object on the radar.

The pushMatrix() function saves the current transformation matrix.

The translate(960,1000) function moves the origin of the coordinate system to (960, 1000).

The strokeWeight(9) function sets the thickness of the lines.

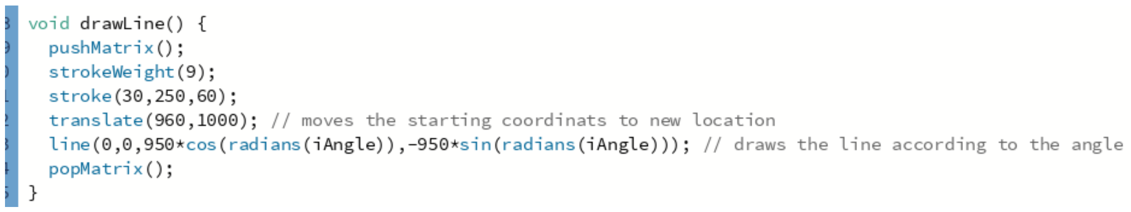
The stroke(255,10,10) function sets the stroke color to red.

The pixsDistance = iDistance*22.5 line converts the distance from centimeters to pixels.

If iDistance is less than 40, a line representing the object is drawn according to the angle and distance.

The popMatrix() function restores the previous transformation matrix.

5.1.2.8 Draw Line Function



```
void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(30,250,60);
  translate(960,1000); // moves the starting coordinats to new location
  line(0,0,950*cos(radians(iAngle)),-950*sin(radians(iAngle))); // draws the line according to the angle
  popMatrix();
}
```

The drawLine() function draws the moving line on the radar.

The pushMatrix() function saves the current transformation matrix.

The strokeWeight(9) function sets the thickness of the lines.

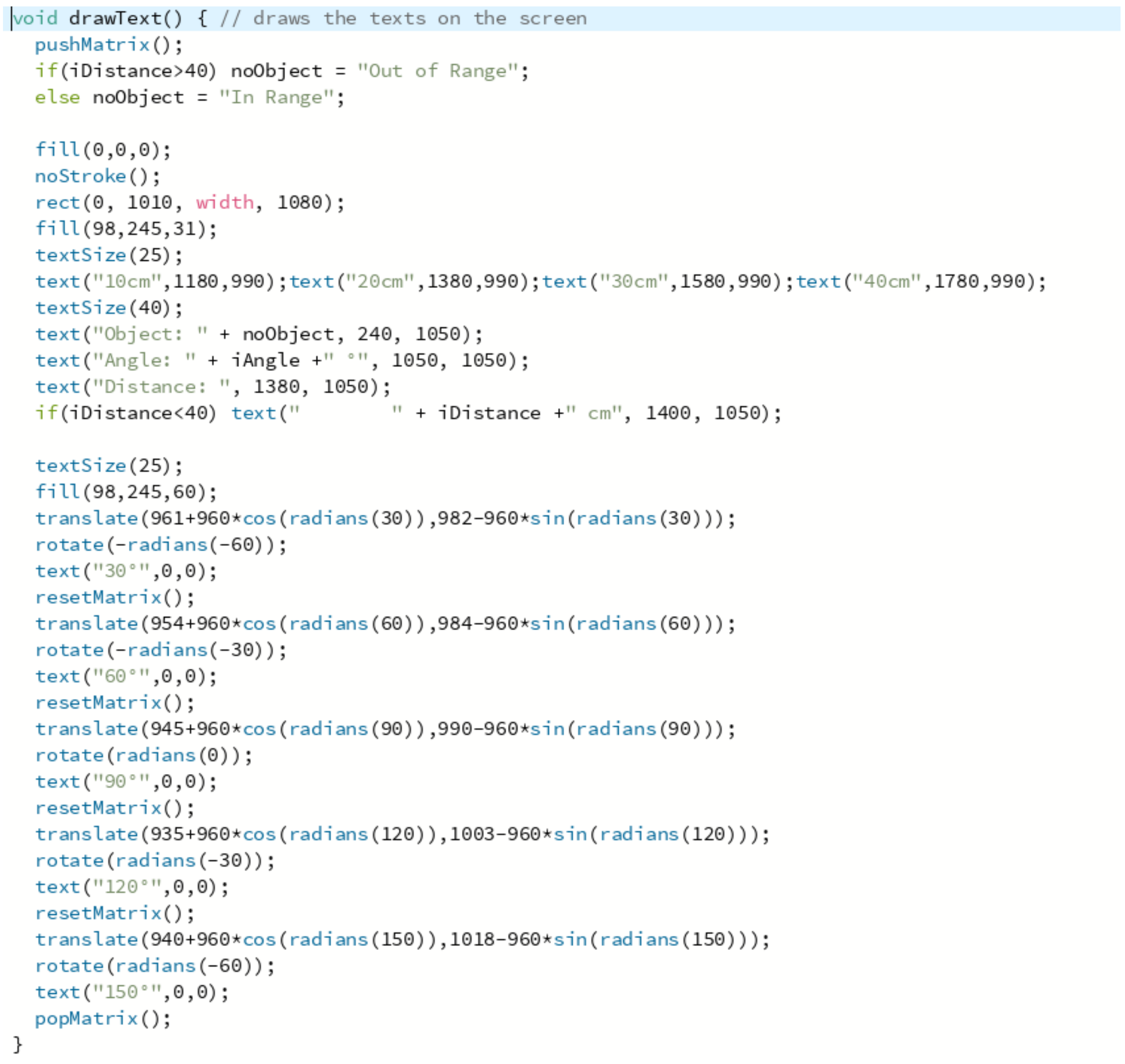
The stroke(30,250,60) function sets the stroke color to green.

The translate(960,1000) function moves the origin of the coordinate system to (960, 1000).

The line(0,0,950*cos(radians(iAngle)),-950*sin(radians(iAngle))) function draws the line according to the angle.

The popMatrix() function restores the previous transformation matrix.

5.1.2.7 Draw Text Function



```
void drawText() { // draws the texts on the screen
  pushMatrix();
  if(iDistance>40) noObject = "Out of Range";
  else noObject = "In Range";

  fill(0,0,0);
  noStroke();
  rect(0, 1010, width, 1080);
  fill(98,245,31);
  textSize(25);
  text("10cm",1180,990);text("20cm",1380,990);text("30cm",1580,990);text("40cm",1780,990);
  textSize(40);
  text("Object: " + noObject, 240, 1050);
  text("Angle: " + iAngle +" °", 1050, 1050);
  text("Distance: ", 1380, 1050);
  if(iDistance<40) text("        " + iDistance +" cm", 1400, 1050);

  textSize(25);
  fill(98,245,60);
  translate(961+960*cos(radians(30)),982-960*sin(radians(30)));
  rotate(-radians(-60));
  text("30°",0,0);
  resetMatrix();
  translate(954+960*cos(radians(60)),984-960*sin(radians(60)));
  rotate(-radians(-30));
  text("60°",0,0);
  resetMatrix();
  translate(945+960*cos(radians(90)),990-960*sin(radians(90)));
  rotate(radians(0));
  text("90°",0,0);
  resetMatrix();
  translate(935+960*cos(radians(120)),1003-960*sin(radians(120)));
  rotate(radians(-30));
  text("120°",0,0);
  resetMatrix();
  translate(940+960*cos(radians(150)),1018-960*sin(radians(150)));
  rotate(radians(-60));
  text("150°",0,0);
  popMatrix();
}
```

Initial Setup: The current transformation matrix is saved with pushMatrix().

Range Check: The noObject variable is set based on whether the detected distance is greater than 40 cm.

Background: A black rectangle is drawn as a background for the text.

Text Setup: The text color is set to green, and text size is adjusted.

Distance Markers: Distance markers (10cm, 20cm, 30cm, 40cm) are drawn at the bottom.

Object Info: Object status, angle, and distance are displayed.

Angle Labels: Labels for angles (30°, 60°, 90°, 120°, 150°) are drawn around the radar using transformations for positioning.

Restore Matrix: The transformation matrix is restored to its previous state with popMatrix().

CHAPTER SIX

TESTS AND RESULTS

6.1 Tests

The testing phase involved verifying the accuracy and functionality of the radar system. The ultrasonic sensor was positioned at various angles using the servo motor, and its distance measurements were compared against known distances to ensure accuracy. Multiple objects were placed at different distances and angles within the sensor's range to test its detection capabilities. The LCD display was monitored to ensure it accurately reflected the measured distances and angles. Additionally, the data transmitted to the Processing IDE was checked for consistency and correctness. Any discrepancies or inaccuracies encountered during these tests were documented and addressed through iterative adjustments in the code and hardware setup. This process ensured that the radar system provided reliable and precise measurements in different scenarios.

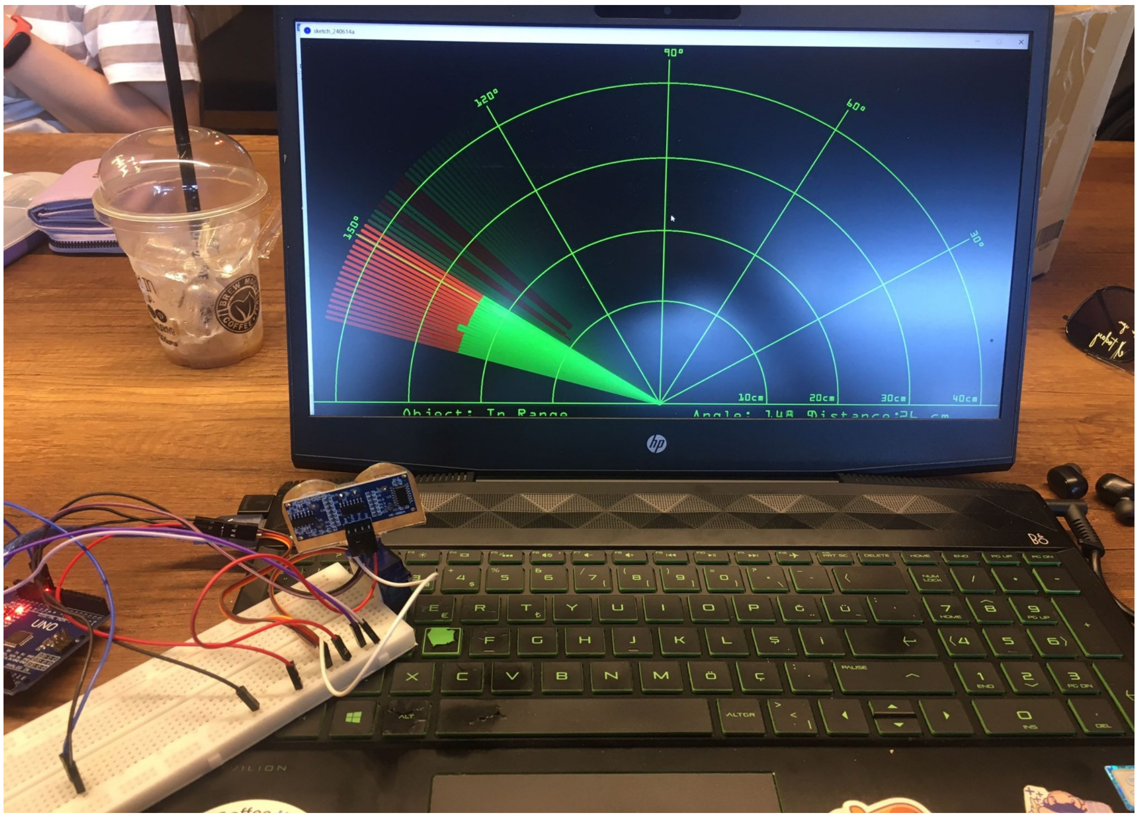


Figure 6.1.1 Testing 1

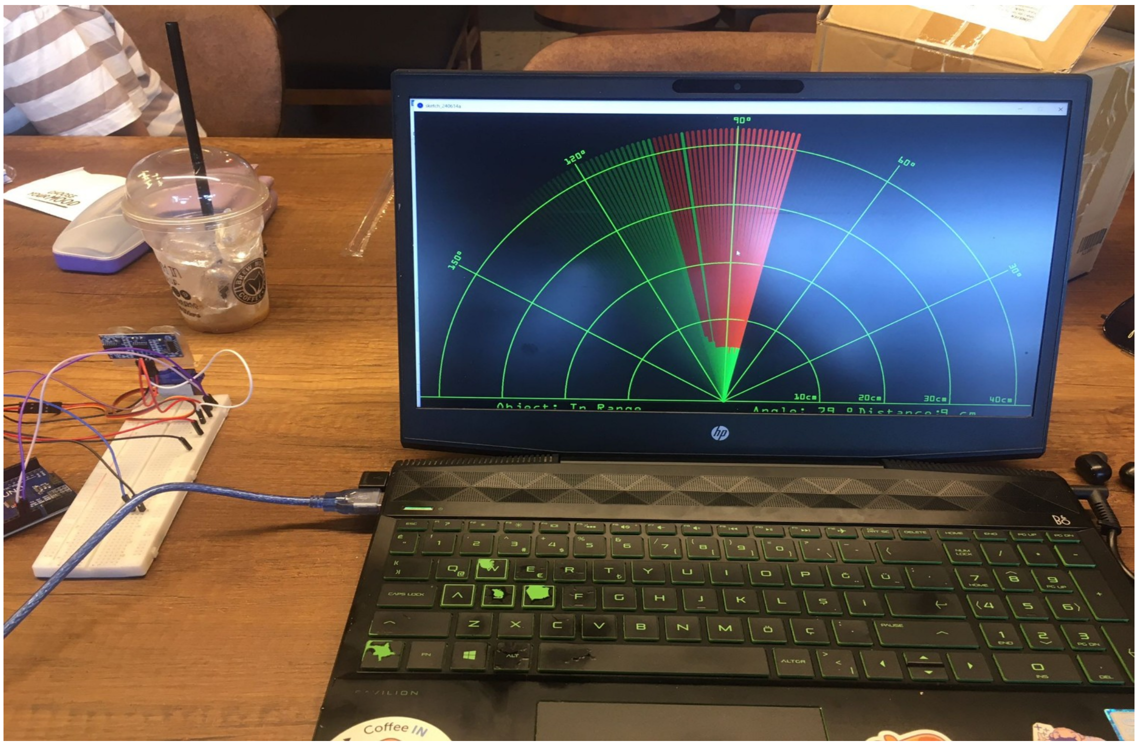


Figure 6.1.2 Testing 2

6.2 Results

The results of the testing phase confirmed that the radar system functioned as intended. The ultrasonic sensor accurately measured distances within its specified range, and the servo motor successfully rotated the sensor to cover the designated angles. The LCD display correctly showed the distance and angle of detected objects, and the data sent to the Processing IDE was consistently accurate. The system reliably detected objects at various distances and angles, with minimal error. The iterative adjustments made during the testing phase effectively resolved any initial discrepancies, leading to a stable and efficient radar system. The project successfully met its objectives, demonstrating the potential for practical applications in areas such as security systems, robotics, and educational projects.

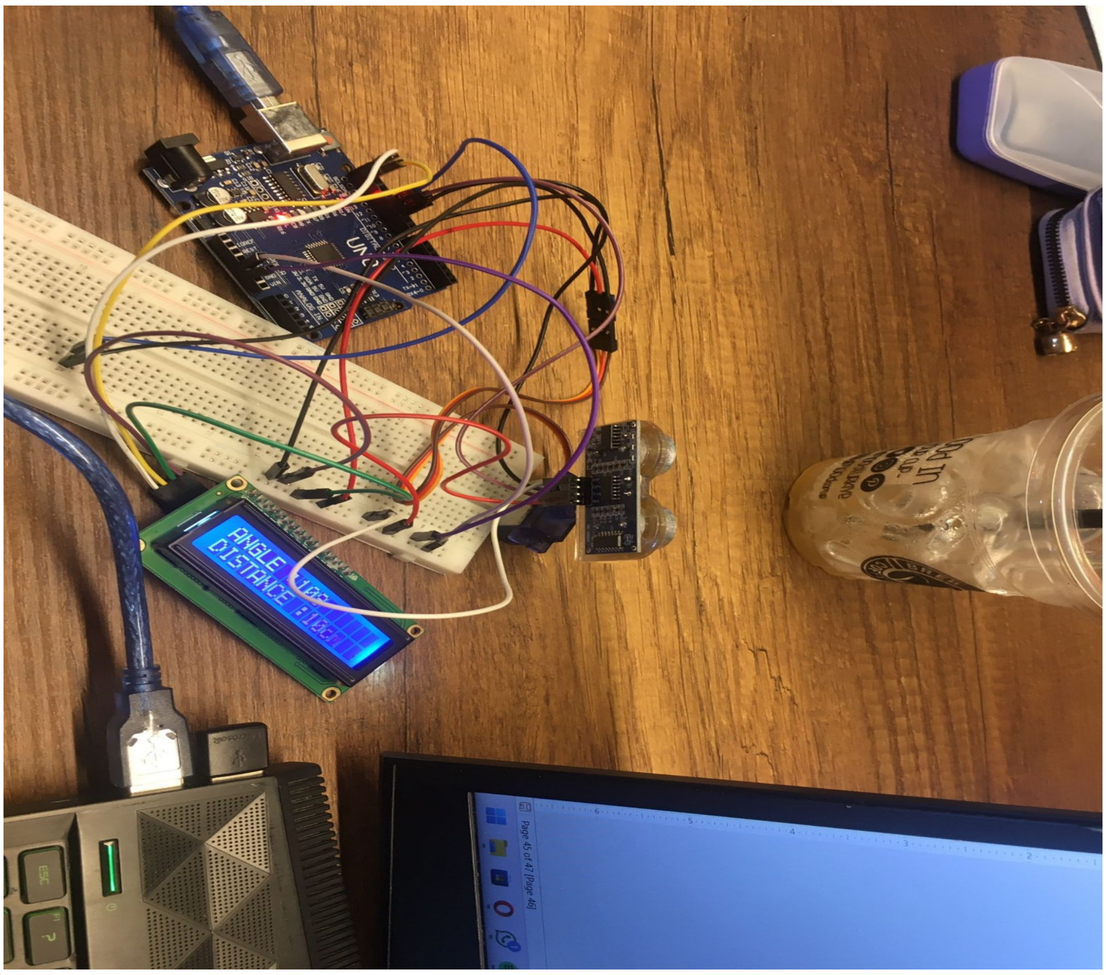


Figure 6.2 Observing Results

CHAPTER SEVEN

CONCLUSION

7.1 Conclusion

In this paper, a lab-scaled radar system was designed and implemented using an Arduino, a servo-motor and an ultrasonic sensor. The developed system is able to read the distance of obstacles and the angle of incident and convert this data into visually represented information. The system performance measures up with other systems at its level as it adequately reports any obstacle it finds in its path and provides an estimated range of the object. A very handy application for this system would be in the area of object detection and avoidance systems for robotics or maybe in intrusion detection systems for location sizes where it may not be economical to use multiple units to provide adequate coverage. The system's range is dependent on the range of the ultrasonic sensor that is used. In this system, the HC-SR04 ultrasonic sensor was used which has a range between 2cm and 40cm.

REFERENCES

[1] D. B. Kadam, Y. B. Patil, K. V. Chougale and. S. S. Perdeshi, "Arduino Based Moving Radar System," International Journal of Innovative Studies in Sciences and Engineering Technology (IJISSET), vol. 3, no. 4, pp. 23-27, 2017.

[2] T. P. Rajan, K. K. Jithin, K. S. Hareesh, C. A. Habeeburahman and. A. Jithin, "Range Detection based on Ultrasonic Principle," International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, vol. 3, no. 2, pp. 7638-7643, 2014.

[3] P. P. Arun, M. A. Sudhakar, P. MeghaSunil and S. S. Balaji, "Ultrasonic Distance Meter," SVERIAN Scientific, pp. 1-4, 2015.

[4] Shamsul A., Tajrian M., "Design of an Ultrasonic Distance Meter", International Journal of Scientific & Engineering Research, pp. 1-10, March 2013.

[5] U. Papa, G. D. Core, "Design of sonar sensor model for safe landing of an UAV", Proc. IEEE Metrol. Aerosp., pp. 346- 350, Jun. 2015.

[6] Abbay P., Akhilesh S., Amrit P., and Prof Kriti, "A Review on Ultrasonic Radar Sensor for Security system", Journal of Emerging Technologies and Innovative Research (JETIR), April 2016.

[7] Babu Varghese, "Collision Avoidance System in Heavy Traffic & Blind spot Assist Using Ultrasonic Sensor", International Journal of Computer Science and Engineering Communications-IJCSEC. Vol. 2, Isuue 1, February, 2014 ISSN: 2347-8586.

[8] S. Bharambe, R. Thakker, H. Patil, K. M. Bhurchandi, "Substitute eyes for blind with navigator using android", Proc. Texas Instrum. India Edu. Conf. (TIIEC), pp. 38-43, Apr. 2013.

[9] D. Sunehra, A. Bano, S. Yandrathi, "Remote monitoring and control of a mobile robot system with obstacle avoidance capability", Proc. Int. Conf Adv. Comput. Commun. Inf. (ICACCI), pp. 1803-1809, Aug. 2015.