Internship Project Based Report on

ANDHRA PRADESH STATE SKILL DEVELOPMENT CORPORATION

An Industrial training report submitted as a part of Industrial Training at

APSSDC

By **ROSHINISHA BEGUM**

(Reg.No.322114214071)

Under the Esteemed guidance of

M. UMA MAHESWARA RAO (M.TECH)

DEPARTMENT OF ELECTRICAL ENGINEERING

ANDHRA UNIVERSITY COLLEGE OF ENGINEERING FOR WOMEN

ANDHRA UNIVERSITY

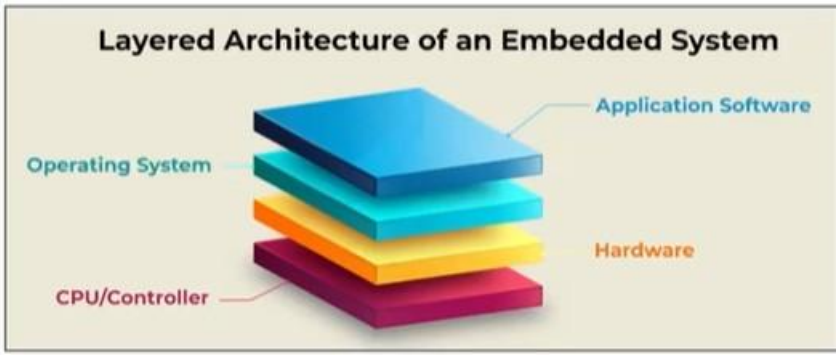
VISAKHAPATNAM-530017

2022-2026

CHAPTER 1: HISTORY OR THEORY OF EMBEDDED SYSTEMS

1.1 Introduction to Embedded Systems

An **embedded system** is a computer system that is designed to perform a specific task or function within a larger system. Unlike general-purpose computers, embedded systems are dedicated to one or a few functions and are typically embedded as part of a complete device including hardware and mechanical parts. They operate under real-time constraints and are designed to be highly reliable, efficient, and responsive.



Embedded systems consist of:

- A **microcontroller or microprocessor**
- **Memory**
- **Input/output interfaces**
- Application-specific **software**

They are used in numerous real-time applications such as industrial control, medical devices, consumer electronics, robotics, and automation systems.

1.2 Evolution of Embedded Systems

The concept of embedded systems dates back to the 1960s with the development of the **Apollo Guidance Computer** by NASA. Over time, technology has rapidly evolved:

- **Early Phase**: Embedded systems used basic **microprocessors**, which required external components like memory and I/O interfaces. They were bulky and less efficient.
- **8051 Microcontroller**: One of the first and most widely used microcontrollers in embedded applications. It integrated memory, I/O, and processing units into a single chip.
- **AVR Microcontrollers**: Introduced by Atmel, offered better power efficiency and were widely used in hobby and academic projects (e.g., Arduino boards).
- **ARM Processors**: Today's embedded systems often use **ARM-based microcontrollers**, known for low power consumption, high speed, and advanced features like floating point units, multiple timers, and various communication protocols.

This evolution made embedded systems more compact, powerful, and suitable for advanced real-time control applications.

1.3 Why Embedded Systems for Radar Applications?

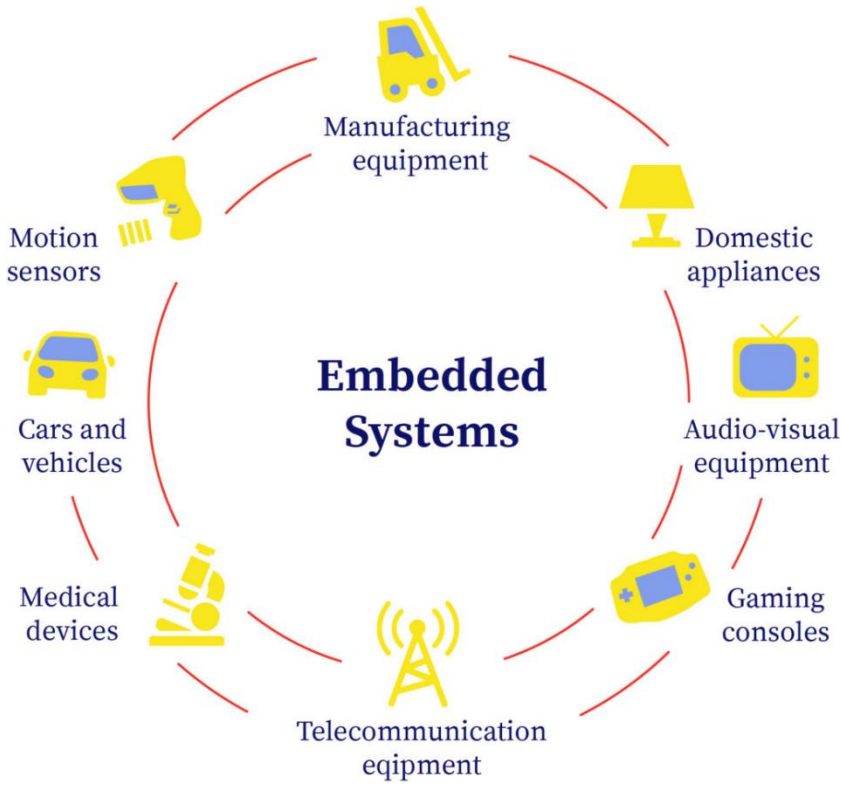
Radar systems involve the transmission and reception of electromagnetic waves to detect objects, measure distance, and estimate speed. These operations require **real-time data processing**, signal filtering, and timely response — all of which are best handled by embedded systems.

Embedded systems are suitable for radar-based applications because:

- They can process sensor data in real time.
- They are small and easy to integrate into compact systems.
- They consume low power, suitable for mobile or automotive platforms.
- They offer flexibility through programmable software for tuning and calibration.

1.4 Applications in Automotive, Defence, and Surveillance

Embedded systems play a critical role in modern **radar-based applications** across various fields:



- **Automotive**:
  - Collision avoidance systems
  - Parking assistance using short-range radar
  - Blind-spot monitoring
- **Defense**:
  - Surveillance radars
  - Target tracking systems
  - Missile guidance
- **Surveillance and Security**:
  - Motion detection in restricted areas
  - Intruder detection using radar sensors
  - Smart traffic monitoring systems

CHAPTER 2: AURDINO & HC-SR04

2.1 AURDINO UNO

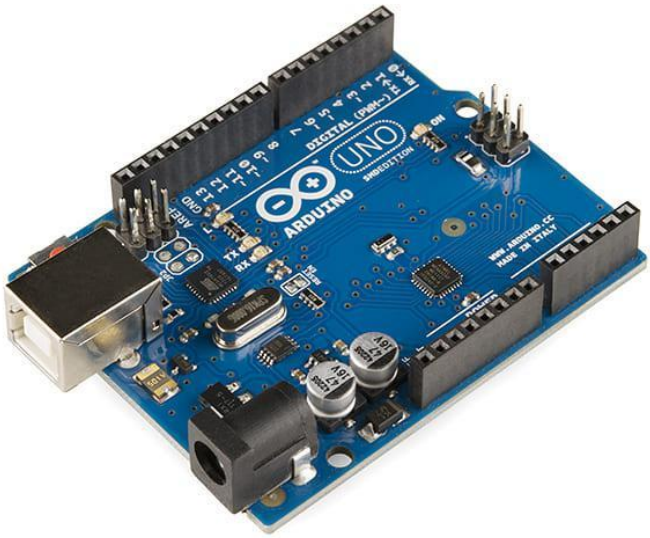
Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are capable of reading inputs (e.g., light on a sensor, a finger on a button) and turning it into an output (e.g., activating a motor, turning on an LED). Developed in 2005, Arduino quickly became a key tool for creating devices that interact with their environment using sensors and actuators.

Arduino's Global Impact

Arduino has revolutionized the world of electronics and programming by making it accessible to students, hobbyists, artists, and professionals around the globe. The platform's simplicity and open-source nature have enabled a large community of users to share projects and ideas, fostering innovation and creativity.

The Arduino UNO

The Arduino UNO is the most popular and widely used board in the Arduino family. It is perfect for beginners and provides all the basic functionalities needed for simple projects.



The Arduino UNO board.

2.2 KEY FEATURES OF ARDUINO UNO:

- Microcontroller: ATmega328P
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Digital I/O Pins: 14 (6 PWM outputs)
- Analog Input Pins: 6
- DC Current per I/O Pin: 20 mA
- Flash Memory: 32 KB (ATmega328P) of which 0.5 KB used by bootloader
- SRAM: 2 KB (ATmega328P)
- EEPROM: 1 KB (ATmega328P)
- Clock Speed: 16 MHz
- Pin Configuration and Memory

Pin Explanation:

- Digital Pins (0-13): Used for general-purpose input/output. Pins 0 and 1 are used for serial communication.
- Analog Pins (A0-A5): Used for analog input, can read sensors and other analog devices.
- Power Pins: Includes 3.3V, 5V, GND, and Vin. Vin is used to provide an external voltage to the Arduino.
- PWM Pins (~3, ~5, ~6, ~9, ~10, ~11): Used for pulse-width modulation to control things like the brightness of an LED or the speed of a motor.
- AREF Pin: Used to set an external reference voltage for the analog inputs.

Memory:

- Flash Memory: 32 KB used for storing the Arduino program (code).
- SRAM: 2 KB used for creating and manipulating variables during program execution.
- EEPROM: 1 KB used to store long-term information.

Getting Started with Tinkercad

Tinkercad is an easy-to-use web-based application for designing and simulating electronics projects. Here's how to get started with Arduino on Tinkercad:

Registration and Setup:

1. Create an Account: Visit the Tinkercad website and create an account.
2. Login and Navigate to Circuits: After logging in, click on “Circuits” from the Tinkercad dashboard.
3. Create a New Circuit: Click on “Create new Circuit” to start designing.

Adding Components:

Search for Components: Use the search bar to find Arduino UNO, breadboards, LEDs, resistors, and other components.

- Drag and Drop: Click on the desired component and drag it into the workspace.

- Connect Components: Connect the components by clicking on the pins and dragging wires between them.

2.3 What is Ultrasonic Sensor : Working & Its Applications

Ultrasonic sensors are in available for the past many decades and these devices continue to hold huge space in the sensing market because of their specifications, affordability, and flexibility. As the automation industry has been progressing, the employment of ultrasonic sensors in multiple domains such as drones, EV vehicles is emerging. In the year 1914, **Fessenden** developed the first modern transducer employed in sonar where it can be able to find the items in water but not the direction of items. And then in the year, 1915 Langevin introduced the contemporary model of ultrasonic which resolved the problem of Fessenden. **ultrasonic sensor definition**, how it works, it's specifications, its integration with Arduino, and its advantages are explained clearly in this article.

What is Ultrasonic Sensor?

Ultrasonic sensors are electronic devices that calculate the target's distance by emission of ultrasonic sound waves and convert those waves into electrical signals. The speed of emitted ultrasonic waves traveling speed is faster than the audible sound.

There are mainly two essential elements which are the transmitter and receiver. Using the piezoelectric crystals, the transmitter generates sound, and from there it travels to the target and gets back to the receiver component.

To know the distance between the target and the sensor, the sensor calculates the amount of time required for sound emission to travel from transmitter to receiver. The calculation is done as follows:

$$\mathbf{D = 1/2\ T * C}$$

Where 'T' corresponds to time measured in seconds

'C' corresponds to sound speed = 343 measured in mts/sec

Ultrasonic sensor working principle is either similar to sonar or radar which evaluates the target/object attributes by understanding the received echoes from sound/radio waves correspondingly. These sensors produce high-frequency sound waves and analyze the echo which is received from the sensor. The sensors measure the time interval between transmitted and received echoes so that the distance to the target is known.

Ultrasonic Sensor Specifications

Knowing the specifications of an ultrasonic sensor helps in understanding the reliable approximations of distance measurements.

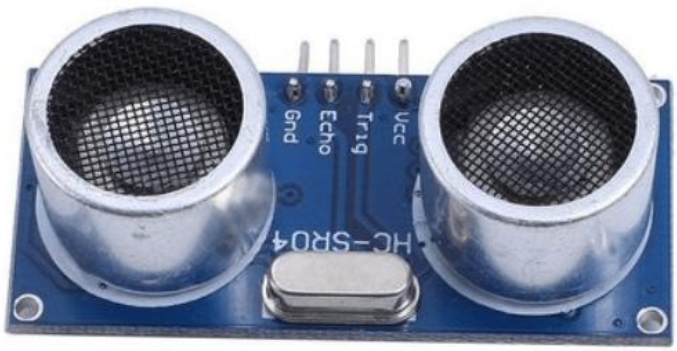
- The sensing range lies between 40 cm to 300 cm.
- The response time is between 50 milliseconds to 200 milliseconds.
- The Beam angle is around $5^{0.}$
- It operates within the voltage range of 20 VDC to 30 VDC
- Preciseness is ±5%
- The frequency of the ultrasound wave is 120 kHz
- Resolution is 1mm

- The voltage of sensor output is between 0 VDC – 10 VDC
- The ultrasonic sensor weight nearly 150 grams
- Ambient temperature is -25^{0}C to +70^{0}C
- The target dimensions to measure maximum distance is 5 cm × 5 cm

Ultrasonic Sensor Arduino

This section explains the interfacing of the ultrasonic sensor with an Arduino by considering HC-SR-04 where it explains the ultrasonic sensor pinout, its specifications, wiring diagram, and how the sensor with Arduino connection.

The ultrasonic sensor pin diagram is:



Ultrasonic Sensor Pin Diagram

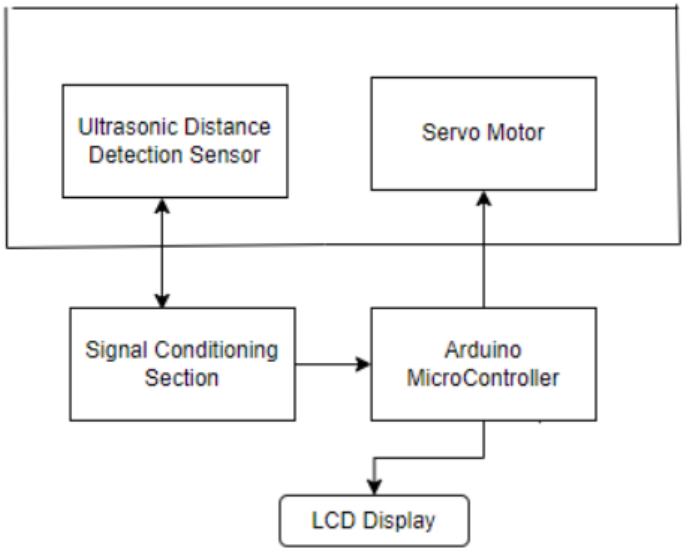
Vcc – This pin has to be connected to a power supply +5V.

TRIG – This pin is used to receive controlling signals from the Arduino board. This is the triggering input pin of the sensor

ECHO – This pin is used for sending signals to the Arduino board where the Arduino calculates the pulse duration to know the distance. This pin is the ECHO output of the sensor.

GND – This pin has to be connected to the ground.

The below picture shows the **ultrasonic sensor block diagram** for distance measurement.



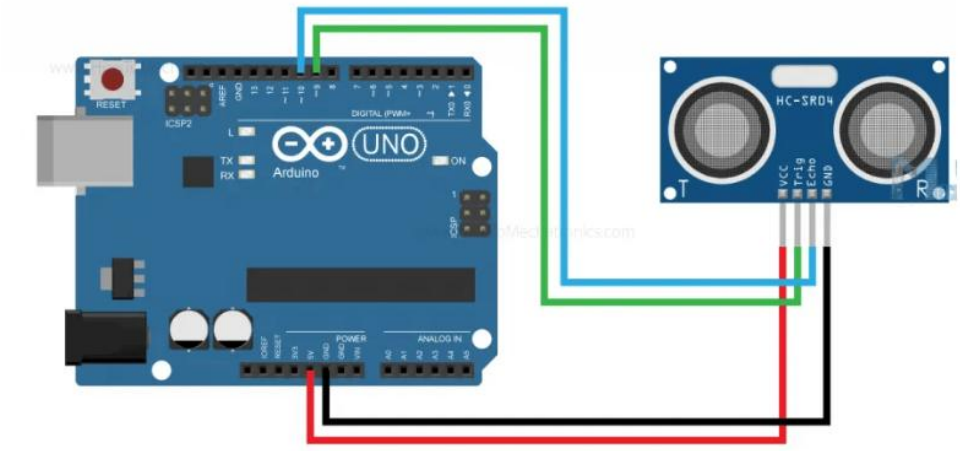
Ultrasonic Sensor Block Diagram

The target's distance is calculated using an ultrasonic distance sensor and the output from the sensor is provided to the signal conditioning section and then is processed using an Arduino microcontroller. The results from the microcontroller are fed to the LCD display and then moved to a personal computer.

The ultrasonic sensor can be connected to the servo motor to know the polar distance of the sensor up to 1800 rotations approximately.

Working

In general, an ultrasonic sensor has two sections which are the transmitter and receiver. These sections are closely placed so that the sound travel in a straight line from the transmitter to the target and travels back to the receiver. Making sure to have minimal distance between transmitter and receiver section delivers minimal errors while calculations.



Integration of Ultrasonic Transducer with Arduino

These devices are also termed ultrasonic transceivers because both the transmitter and receiver sections are combined in a single unit which considerably minimizes the PCB footprint.

Here, the sensor operates as a burst signal and it is transmitted for some period. Later the transmission, there exists a **silent period** and this period is termed **response time**. The response time indicates that it is waiting for the reflected waves.

The shape of the acoustic waves that leave the transmitter section resembles the same shape of the light emitted from a laser so beam angle and spread have to be measured. When the sound waves move away from the transmitter, the detection area increases vertically and sideways too. Because of the varying detection area, the coverage specification is considered either as beam angle/beamwidth other than the standard area of detection.

It is more recommended to observe the beam angle pattern for the sensor whether it is the complete angle of the beam or the angle of variation corresponding to the straight line that forms a transducer. Mostly, a thin beam angle results in a higher detection range, and a broader beam angle corresponds to a lesser detection range.

The transmitted/acoustic signals might find a hindrance or not. When there is any hindrance, the acoustic wave bounces back from the hindrance. This bounced signal is termed ECHO. This echo travels to the receiver.

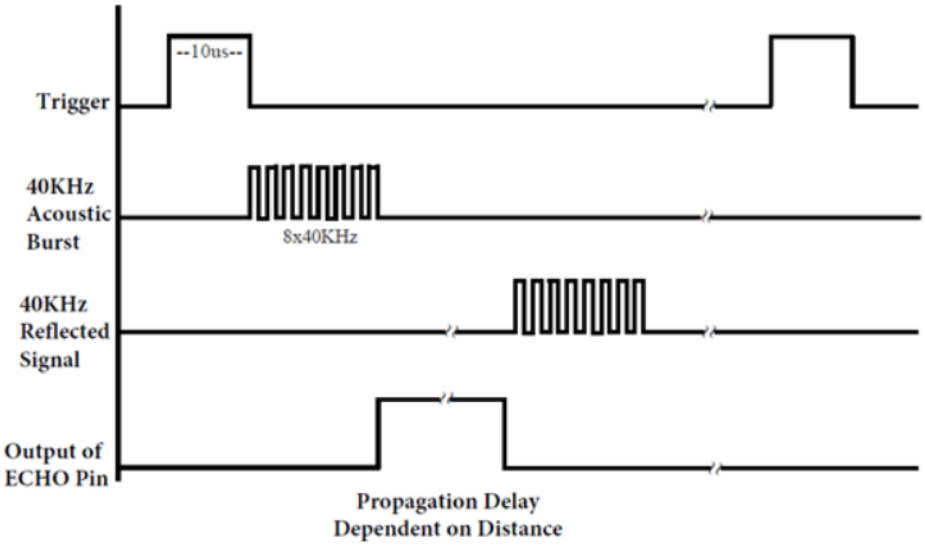
Then the received signal is either filtered or amplified and then transformed into a digital signal. With the time between transmission and reception of acoustic waves, the distance between the ultrasonic system and hindrance can be known.

Let us consider an example to know the **Ultrasonic sensor timing diagram**.

Consider HC-SR-04 ultrasonic sensor where we should provide trigger pulse. It produces a sound wave with a frequency of 40 kHz (corresponds to 8 pulses). This makes the ECHO pin to the HIGH state. The echo pin will stay in a HIGH state until and unless it receives the ECHO sound. Therefore, the echo pin width is calculated as the time for the sound to travel from an object and get back. From the time, the distance is measured and this is termed sound speed.

The rangeability of HC-SR-04 is 2 cm – 400 cm.

The below picture shows the timing diagram of HC-SR-04.



Ultrasonic Sensor Timing Diagram

Procedure to analyze the timing diagram:

1. To the Trig pin, the trigger pulse should be supplied for at least 10 µsec.
2. Then the device automatically transmits eight pulses of 40 kHz and waits for the rising edge to appear on the output pin.
3. When the echo pin observed a rising edge, start the timer and observe the time required to appear falling edge at the echo pin.
4. When the echo pin shows a falling edge, observe the timer count. The count of the timer indicates the time taken by the sensor for object detection and getting back from the object.

As we know that

D = S * T

D = Distance

S = Speed

T = Time

Total distance is measured as = (343* Time at HIGH ECHO)/2

Note: '343' in the above formula indicates the sound speed in air medium considered at room temperature.

The total distance is divided by 2 because the sound wave travels from the source to the object and then returns back to the source.

The code for **ultrasonic sensor with Arduino** is explained as follow

CHAPTER 3: EMBEDDED SYSTEMS (MICROCONTROLLER DISCUSSION)

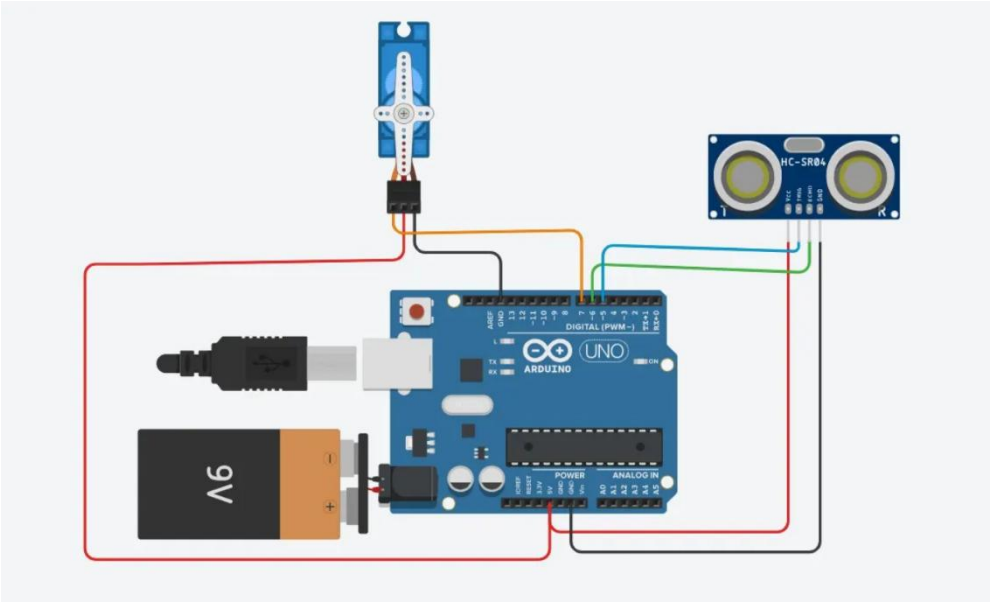
3.1 Introduction

This chapter explains the design and functioning of a radar-based obstacle detection system built using an embedded microcontroller. Unlike conventional designs that use LEDs or buzzers, this project uses a **servo motor** for physical actuation, controlled based on radar input. Additionally, **Processing software** is used for real-time visualization or data communication from Arduino.

3.2 Main Components Used

Component	Description
Microcontroller	**Arduino Uno (ATmega328P)** – Core processor that receives radar input and controls servo
Radar Sensor	**RCWL-0516** Microwave Doppler Radar Sensor – Detects motion via Doppler shift
Output Actuator	**Servo Motor (e.g., SG90)** – Moves/rotates in response to detected motion
Interface Software	**Processing IDE** – Receives serial data from Arduino and visualizes object detection
Power Supply	5V regulated USB or battery input for Arduino and components

Circuit Diagram of Arduino Radar Project



The circuit diagram of this Radar Project is very simple as it involves very little hardware.

Components Required

Hardware

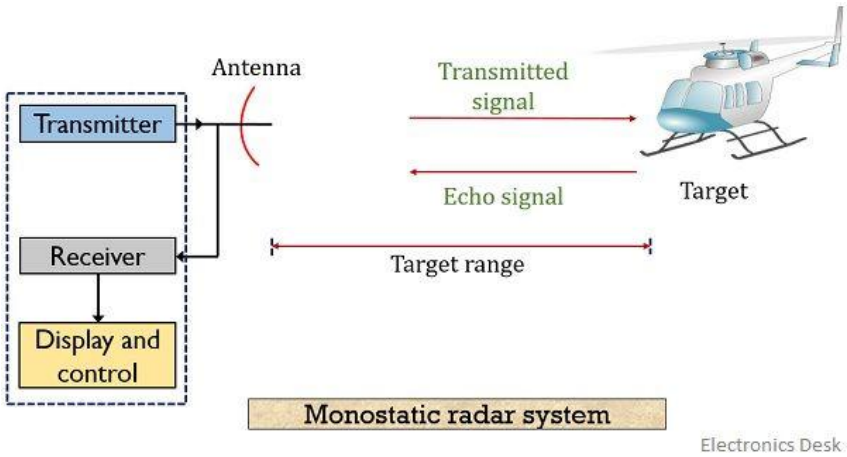
- Arduino UNO
- HC-SR04 Ultrasonic Sensor
- TowerPro SG90 Servo Motor
- Mounting Bracket for Ultrasonic Sensor (optional)
- Connecting Wires
- Jumper Cables
- 5V Power Supply
- USB Cable (for Arduino)

Software

- Arduino IDE
- Processing Application

3.3 Working Principle

1. **Ultra sonic sensor** sensing.
2. It outputs a **digital HIGH** signal when motion is present.
3. Arduino reads this signal and:
    - **Rotates the servo motor** to a predefined angle (e.g., 90°).
    - **Sends detection status to Processing** via the serial port.
4. **Processing Software**:
    - Reads serial data from Arduino.
    - Displays graphical output (e.g., alert screen, bar/indicator).



3.4 Microcontroller Pin Configuration

Device	Arduino Pin
RCWL Output	D2 (Input)
Servo Motor	D9 (PWM Output)
Serial Comm.	TX/RX (USB to Processing software)

3.5 Signal Processing & Communication

- **Input**: Digital signal from RCWL-0516 (HIGH/LOW).
- **Output**: PWM signal to servo motor.
- **Data Transmission**: Serial communication (Serial.println) to Processing software.
- **Timing**: Controlled using delay() for response duration.

3.6 WORKING

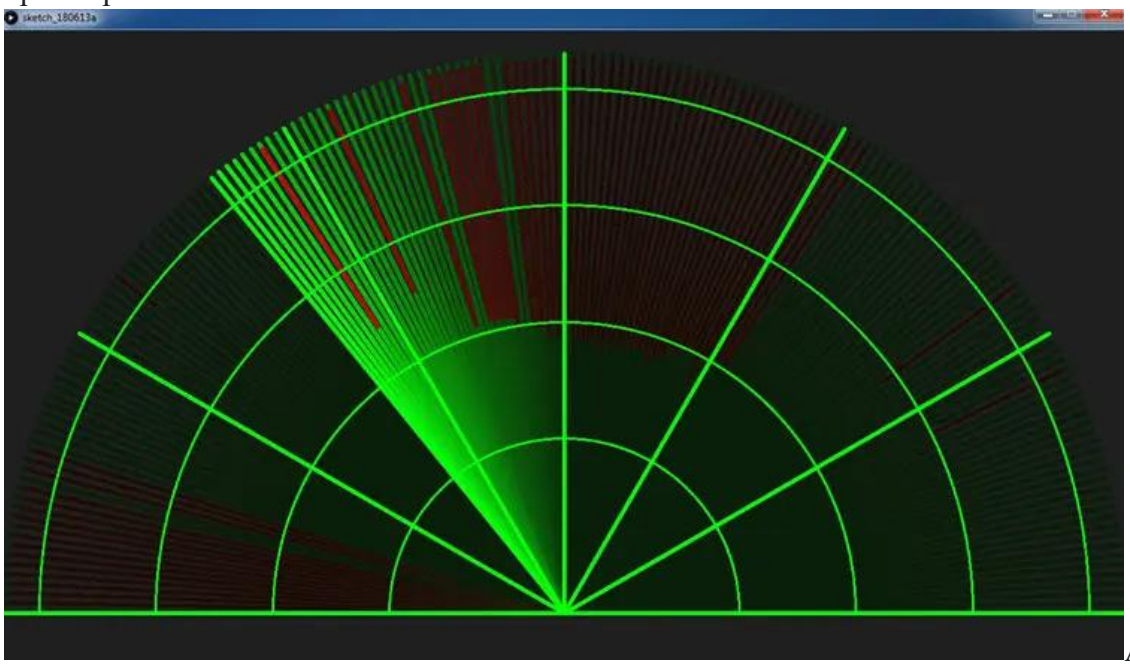
Initially, upload the code to Arduino after making the connections. You can observe the servo sweeping from 0 to 180 and again back to 00. Since the Ultrasonic Sensor is mounted over the Servo, it will also participate in the sweeping action.

Now, open the processing application and paste the above given sketch. In the Processing Sketch, make necessary changes in the COM Port selection and replace it with the COM Port number to which your Arduino is connected to.

If you note the Processing Sketch, I have used the output display size as 1280×720 (assuming almost all computers now-a-days have a minimum resolution of 1366×768) and made calculation with respect to this resolution.

In the future, I will upload a new Processing sketch where you can enter the desired resolution (like 1920×1080) and all the calculations will be automatically adjusted to this resolution.

Now, run the sketch in the Processing and if everything goes well, a new Processing window opens up like the one shown below.



A Graphical representation of the data from the Ultrasonic Sensor is represented in a Radar type display. If the Ultrasonic Sensor detects any object within its range, the same will be displayed graphically on the screen.

CHAPTER 4: PROJECT DISCUSSION

4.1 Overview

This project implements a **radar-based motion detection system** using the **RCWL-0516 microwave sensor**, **Arduino Uno**, a **servo motor**, and **Processing IDE** for visualization. The radar detects motion, and the Arduino drives the servo to a specific angle and communicates the event visually via a graphical interface on the PC.

4.2 Step-by-Step Working

1. **Radar Wave Emission**
   The RCWL-0516 sensor continuously emits low-power microwave signals.
2. **Motion Detection (Doppler Effect)**
   When an object (e.g., human) moves within its range (~7 meters), it reflects the waves back, causing a Doppler shift.
3. **Digital Signal Output**
   The sensor sends a **HIGH signal (5V)** on its output pin whenever motion is detected.
4. **Microcontroller Response**
   - The Arduino reads the digital signal from the radar sensor.
   - If motion is detected:
     - It moves the **servo motor** to a fixed angle (e.g., 90°) to simulate an actuator response.
     - Sends a serial message to **Processing IDE** like `"Motion Detected"`.
   - If no motion:
     - Servo returns to 0°.
     - Sends `"No Motion"` to Processing.
5. **Visualization with Processing Software**
   A GUI developed in the Processing environment displays motion status, logs time, or visualizes object detection using bar graphs, color change, or alerts.

4.3 Code Logic :

AURDINO CODE:

```
// Includes the Servo library

#include <Servo.h>.

// Defines Tirg and Echo pins of the Ultrasonic Sensor

const int trigPin = 10;

const int echoPin = 11;

// Variables for the duration and the distance

long duration;

int distance;
```

```
Servo myServo; // Creates a servo object for controlling the servo motor
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600);
  myServo.attach(12); // Defines on which pin is the servo motor attached
}
void loop() {
  // rotates the servo motor from 15 to 165 degrees
  for(int i=15;i<=165;i++){
  myServo.write(i);
  delay(30);
  distance = calculateDistance();// 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
  }
  // Repeats the previous lines from 165 to 15 degrees
  for(int i=165;i>15;i--){
  myServo.write(i);
  delay(30);
  distance = calculateDistance();
  Serial.print(i);
  Serial.print(",");
  Serial.print(distance);
  Serial.print(".");
  }
}
// Function for calculating the distance measured by the Ultrasonic sensor
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); // Reads the echoPin, returns the sound wave travel time in microseconds
  distance= duration*0.034/2;
  return distance;
}
```

PROCESSING CODE:

```
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;
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;
void setup() {

 size (1200, 700); // ***CHANGE THIS TO YOUR SCREEN RESOLUTION***
 smooth();
 myPort = new Serial(this,"COM5", 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.
}
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();
}
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);
```

```
}
void drawRadar() {
  pushMatrix();
  translate(width/2,height-height*0.074); // moves the starting coordinats to new location
  noFill();
  strokeWeight(2);
  stroke(98,245,31);
  // draws the arc lines
  arc(0,0,(width-width*0.0625),(width-width*0.0625),PI,TWO_PI);
  arc(0,0,(width-width*0.27),(width-width*0.27),PI,TWO_PI);
  arc(0,0,(width-width*0.479),(width-width*0.479),PI,TWO_PI);
  arc(0,0,(width-width*0.687),(width-width*0.687),PI,TWO_PI);
  // draws the angle lines
  line(-width/2,0,width/2,0);
  line(0,0,(-width/2)*cos(radians(30)),(-width/2)*sin(radians(30)));
  line(0,0,(-width/2)*cos(radians(60)),(-width/2)*sin(radians(60)));
  line(0,0,(-width/2)*cos(radians(90)),(-width/2)*sin(radians(90)));
  line(0,0,(-width/2)*cos(radians(120)),(-width/2)*sin(radians(120)));
  line(0,0,(-width/2)*cos(radians(150)),(-width/2)*sin(radians(150)));
  line((-width/2)*cos(radians(30)),0,width/2,0);
  popMatrix();
}
void drawObject() {
  pushMatrix();
  translate(width/2,height-height*0.074); // moves the starting coordinats to new location
  strokeWeight(9);
  stroke(255,10,10); // red color
  pixsDistance = iDistance*((height-height*0.1666)*0.025); // 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)),(width-
width*0.505)*cos(radians(iAngle)),-(width-width*0.505)*sin(radians(iAngle)));
  }
  popMatrix();
}
void drawLine() {
  pushMatrix();
  strokeWeight(9);
  stroke(30,250,60);
  translate(width/2,height-height*0.074); // moves the starting coordinats to new location
  line(0,0,(height-height*0.12)*cos(radians(iAngle)),-(height-
height*0.12)*sin(radians(iAngle))); // draws the line according to the angle
  popMatrix();
}
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, height-height*0.0648, width, height);
  fill(98,245,31);
  textSize(25);

  text("10cm",width-width*0.3854,height-height*0.0833);
  text("20cm",width-width*0.281,height-height*0.0833);
  text("30cm",width-width*0.177,height-height*0.0833);
  text("40cm",width-width*0.0729,height-height*0.0833);
```

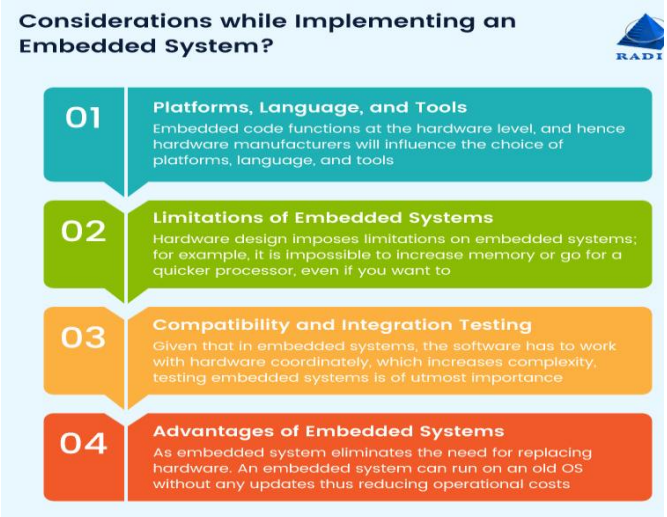
```
  textSize(40);
  text("ELECTRICAL MANDIR ", width-width*0.875, height-height*0.0277);
  text("Angle: " + iAngle +"  ", width-width*0.48, height-height*0.0277);
  text("Distance: ", width-width*0.26, height-height*0.0277);
  if(iDistance<40) {
  text("        " + iDistance +" cm", width-width*0.225, 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));
  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: CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

This project successfully demonstrates a **low-cost radar-based embedded system** that detects motion using the RCWL-0516 microwave radar sensor and responds with a **servo motor** action. The **Arduino Uno** processes the signal in real time and communicates with **Processing IDE** for visualization.

The system shows how embedded technologies can be effectively used in automation and surveillance applications, where real-time detection and actuation are crucial.



5.2 Applications

- **Smart security systems** (motion-activated doors, lights, or alarms)
- **Automotive safety** (collision detection, blind spot warning)
- **Military/defense surveillance** (border motion tracking)
- **Industrial automation** (robot arm trigger, conveyor alerts)
- **Assistive tech** (door opening for elderly when motion is detected)

5.3 Future Scope

- **Wireless Communication**: Add Bluetooth, Wi-Fi (ESP8266), or LoRa for remote alerts.
- **Distance Measurement**: Replace with HB100 or mmWave radar to estimate distance or speed.
- **Object Classification**: Use AI/ML models to identify human vs object.
- **IoT Integration**: Send alerts to cloud/dashboard via MQTT or Blynk.
- **Multi-Sensor Setup**: Add IR or ultrasonic for redundancy or better direction sensing.