A **motion tracker radar system** typically uses **Doppler radar** or **ultrasonic**/**LiDAR** technology to detect and track the movement of objects in a defined space. These systems are used in applications like:

* **Security & surveillance** (detecting intruders)
* **Vehicle tracking and collision avoidance**
* **Industrial automation**
* **Smart home presence detection**
* **Military & drone systems**

**History of radars:**

i. A British engineer **Robert Watson-Watt** gives an idea, to use radio waves to detect any enemy aircraft.

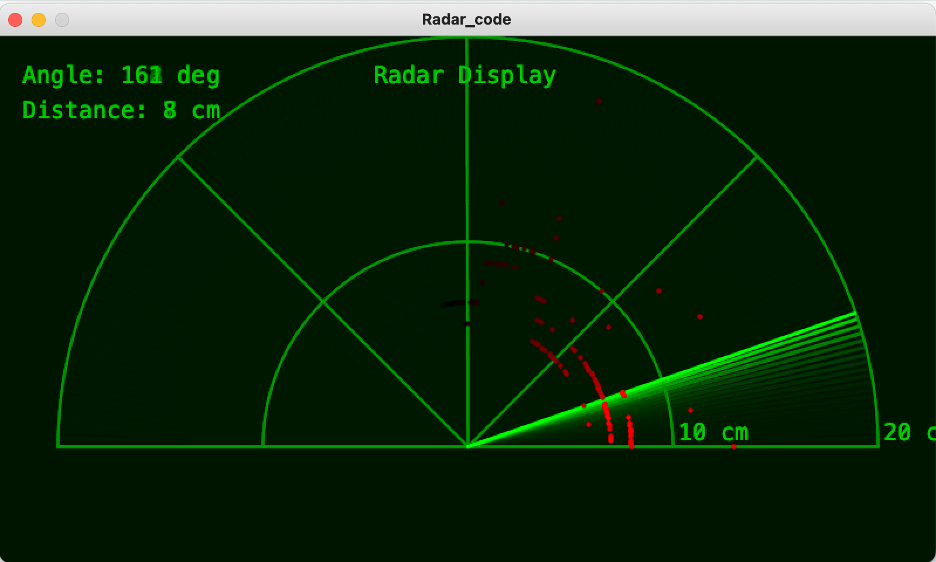
ii. **Chain home system** is the world’s first defense radar system

**How to hide from radar? (Stealth technology)**

1. Using chaff in aircraft.
2. Send your own, same radio waves.
3. Make a Plasma shield.
4. Make an angular shaped aircraft.
5. Using RAM(Radio waves Absorption Material)

Building an Ultrasonic Radar using Arduino and Processing

This project combines an Arduino-controlled ultrasonic sensor with a servo motor to scan for and measure the distance to nearby objects. The collected data is then visualized on your computer as a live, graphical radar screen using the Processing environment.



Components and supplies

**1**

Ultrasonic Sensor - HC-SR04

**1**

Arduino Uno Rev3

**1**

Servo Motor SG90 180 degree

Apps and platforms

**1**

Processing 3

**1**

Arduino IDE

Project description

This project details the construction of a simple yet effective 2D ultrasonic radar system. It serves as a classic introductory project that seamlessly bridges the gap between physical hardware and software-based data visualization. The system uses an Arduino microcontroller to control a sweeping ultrasonic sensor, which detects the presence and distance of objects in a 180-degree field of view. This spatial data is then transmitted to a computer and rendered in real-time as an intuitive, graphical radar display using the Processing development environment.

The final output is a visually satisfying and functional device that mimics the principles of real-world sonar and radar systems, providing a tangible representation of its surroundings.

1. To construct a physical scanning mechanism using a micro servo motor and an HC-SR04 ultrasonic sensor.
2. To program an Arduino board to control the servo motor's sweep and trigger the ultrasonic sensor for distance measurements.
3. To establish reliable serial communication between the Arduino and a host computer to transmit sensor data (angle and distance).
4. To develop a Graphical User Interface (GUI) in the Processing environment to visualize the incoming data as a radar screen.
5. To integrate the hardware and software components into a fully functional object detection and visualization system.

Hardware:

1. Arduino UNO (or compatible board): The central microcontroller that executes the control logic.
2. HC-SR04 Ultrasonic Distance Sensor: The "eyes" of the project, used to measure distances via sonar.
3. SG90 Micro Servo Motor: An actuator that physically rotates the ultrasonic sensor.
4. Breadboard and Jumper Wires: For creating and managing electrical connections without soldering.
5. USB A-to-B Cable: To power the Arduino and establish serial communication with the computer.

Software:

1. Arduino IDE: Used to write, compile, and upload the control code to the Arduino board.
2. Processing Development Environment: A flexible software sketchbook used to write the code for the radar visualization on the computer.

Methodology:

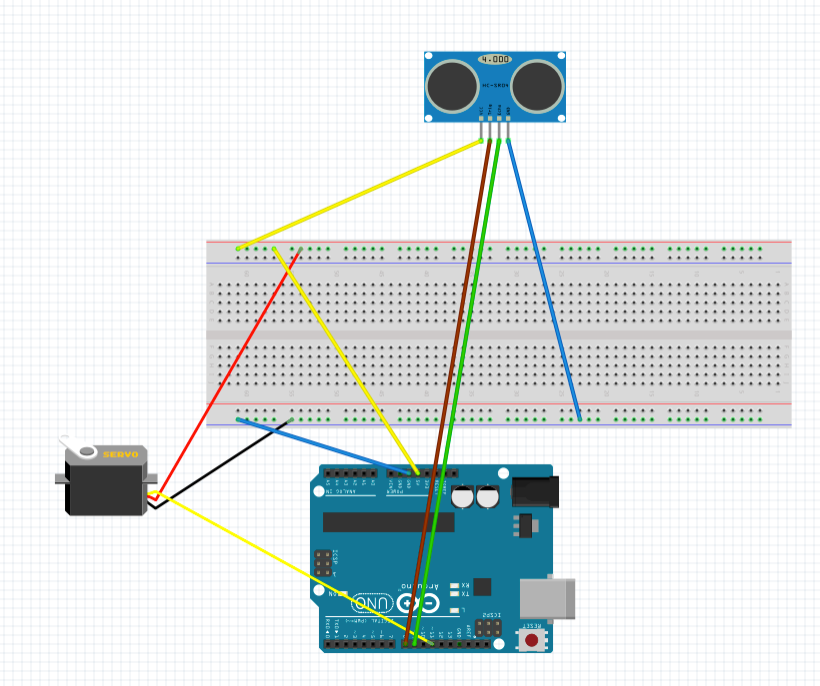
1. The operation of the radar is a coordinated effort between the physical components and the two software environments.
2. Scanning Motion: The Arduino board sends a precise signal to the servo motor, commanding it to rotate in small, incremental steps from 0 to 180 degrees and then back again.
3. Distance Measurement: At each angular step, the Arduino triggers the HC-SR04 ultrasonic sensor. The sensor emits a high-frequency sound pulse and measures the time it takes for the echo to bounce back from an object.
4. Data Calculation: The distance to the object is calculated within the Arduino sketch using the formula:

Distance=(Speed of Sound×Time)/2​

The result is a pair of data points: the current angle of the servo and the calculated distance.

Serial Communication: The Arduino formats this angle and distance data into a string and transmits it to the computer via the USB serial port.

Data Visualization: On the computer, a sketch running in the Processing environment continuously listens to the serial port. When it receives the data string from the Arduino, it parses the angle and distance values. Using trigonometry, it converts these polar coordinates into Cartesian (x, y) points to plot on the screen, creating a visual map of any detected objects relative to the sensor's position.



Simple diagram

This is just part of a bigger project, but I learned a lot doing it. Have a great day.

Code

arduino radar code

c

1#include <Servo.h>

2

3// --- Pin Definitions ---

4const int SERVO\_PIN = 11;

5const int TRIG\_PIN = 8;

6const int ECHO\_PIN = 9;

7

8// --- Constants for Servo ---

9const int MIN\_ANGLE = 0;

10const int MAX\_ANGLE = 180;

11const int ANGLE\_STEP = 1;

12const int SWEEP\_DELAY = 15; // Delay between servo movements in milliseconds

13

14// --- Constants for Ultrasonic Sensor ---

15// The speed of sound is 343 m/s or 29.1 microseconds per centimeter.

16// The pulseIn() function measures the round trip time.

17// So, the distance in cm is (duration / 2) / 29.1 = duration / 58.2

18const float SOUND\_SPEED\_FACTOR = 58.2;

19

20Servo myServo;

21

22void setup() {

23 pinMode(TRIG\_PIN, OUTPUT);

24 pinMode(ECHO\_PIN, INPUT);

25 myServo.attach(SERVO\_PIN);

26 Serial.begin(9600);

27}

28

29void loop() {

30 // Sweep from MIN\_ANGLE to MAX\_ANGLE

31 sweepAndMeasure(MIN\_ANGLE, MAX\_ANGLE, ANGLE\_STEP);

32

33 // Sweep back from MAX\_ANGLE to MIN\_ANGLE

34 sweepAndMeasure(MAX\_ANGLE, MIN\_ANGLE, -ANGLE\_STEP);

35}

36

37/\*\*

38 \* @brief Sweeps the servo motor and measures the distance at each step.

39 \* @param startAngle The starting angle of the sweep.

40 \* @param endAngle The ending angle of the sweep.

41 \* @param step The increment or decrement for the angle.

42 \*/

43void sweepAndMeasure(int startAngle, int endAngle, int step) {

44 for (int angle = startAngle; (step > 0) ? (angle <= endAngle) : (angle >= endAngle); angle += step) {

45 myServo.write(angle);

46 delay(SWEEP\_DELAY);

47 int distance = calculateDistance();

48 printData(angle, distance);

49 }

50}

51

52/\*\*

53 \* @brief Calculates the distance using the ultrasonic sensor.

54 \* @return The calculated distance in centimeters.

55 \*/

56int calculateDistance() {

57 // Trigger the ultrasonic sensor

58 digitalWrite(TRIG\_PIN, LOW);

59 delayMicroseconds(2);

60 digitalWrite(TRIG\_PIN, HIGH);

61 delayMicroseconds(10);

62 digitalWrite(TRIG\_PIN, LOW);

63

64 // Read the echo pulse

65 long duration = pulseIn(ECHO\_PIN, HIGH);

66

67 // Calculate the distance in centimeters

68 return static\_cast<int>(duration / SOUND\_SPEED\_FACTOR);

69}

70

71/\*\*

72 \* @brief Prints the angle and distance data to the Serial Monitor.

73 \* @param angle The current angle of the servo.

74 \* @param distance The measured distance.

75 \*/

76void printData(int angle, int distance) {

77 Serial.print(angle);

78 Serial.print(",");

79 Serial.print(distance);

80 Serial.print(".");

81}

Processing radar code

python

1# Processing (Python) code for a Radar Display

2# This script reads angle and distance data from an Arduino over the serial port

3# and visualizes it as a classic radar screen.

4#

5# MODIFIED to use a rectangular, half-screen layout.

6

7add\_library('serial')

8

9# --- Global Variables ---

10serial\_port = None # The serial port object

11font = None # Font for displaying text

12in\_string = "" # String to buffer data from serial

13

14# CHANGED: Use rectangular screen dimensions

15screen\_width = 800

16screen\_height = 450 # Half the original height with some padding

17

18# Radar screen properties

19radar\_radius = 350

20# CHANGED: Center coordinates are now relative to the new rectangular screen

21radar\_center\_x = screen\_width / 2

22# Position the radar's baseline near the bottom of the window

23radar\_center\_y = screen\_height - 100

24

25# Data from Arduino

26current\_angle = 0

27current\_distance = 0

28

29# Data storage for drawing points

30point\_history = []

31

32def setup():

33 """

34 This function runs once when the program starts.

35 It sets up the display window, initializes the serial communication,

36 and loads the font.

37 """

38 global serial\_port, font

39

40 # --- Window and Graphics Setup ---

41 # CHANGED: Use the new rectangular dimensions

42 size(screen\_width, screen\_height)

43 smooth()

44

45 # Create a font for text display.

46 font = createFont("Monospaced", 20)

47 textFont(font)

48

49 # --- Serial Communication Setup ---

50 print("Available Serial Ports:")

51 print(Serial.list())

52

53 # ---!!! IMPORTANT: YOU MUST USE THE CORRECT PORT NAME !!!---

54 port\_name = "/dev/cu.usbmodem11101"

55

56 try:

57 print("Connecting to port: {}".format(port\_name))

58 serial\_port = Serial(this, port\_name, 9600)

59 serial\_port.clear()

60

61 except Exception as e:

62 print("Error opening serial port: {}".format(e))

63 print("Please make sure the port name is correct and the Arduino is plugged in.")

64 print("Also ensure the Arduino IDE's Serial Monitor is closed.")

65 exit()

66

67

68def draw():

69 """

70 This function runs continuously in a loop, redrawing the screen

71 in each frame.

72 """

73 global current\_angle, current\_distance

74 background(0, 20, 0)

75 draw\_radar\_grid()

76 draw\_text\_labels()

77 draw\_sweep\_line(current\_angle)

78 draw\_detected\_points()

79 update\_and\_draw\_history()

80

81

82def draw\_radar\_grid():

83 """Draws the concentric semi-circles and lines of the radar."""

84 stroke(0, 150, 0)

85 noFill()

86 strokeWeight(2)

87

88 # Draw 2 concentric semi-circles for the top-half display

89 for i in range(1, 3):

90 radius = i \* (radar\_radius / 2.0)

91 # Use arc() to draw only the top half of the circle (from PI to TWO\_PI)

92 arc(radar\_center\_x, radar\_center\_y, radius \* 2, radius \* 2, PI, TWO\_PI)

93

94 # Draw the horizontal closing line for the semi-circle display

95 line(radar\_center\_x - radar\_radius, radar\_center\_y, radar\_center\_x + radar\_radius, radar\_center\_y)

96

97 # Draw 5 radial lines for the 180-degree sweep (left-to-right)

98 for i in range(5):

99 angle = radians((i \* 45) - 180)

100 x2 = radar\_center\_x + radar\_radius \* cos(angle)

101 y2 = radar\_center\_y + radar\_radius \* sin(angle)

102 line(radar\_center\_x, radar\_center\_y, x2, y2)

103

104

105def draw\_text\_labels():

106 """Displays text information on the screen."""

107 fill(0, 200, 0)

108 noStroke()

109 # Label the 2 rings along the horizontal axis

110 for i in range(1, 3):

111 radius\_text = i \* (radar\_radius / 2.0)

112 text(str(i \* 10) + " cm", radar\_center\_x + radius\_text + 5, radar\_center\_y - 5)

113

114 text("Angle: {} deg".format(current\_angle), 20, 40)

115 text("Distance: {} cm".format(current\_distance), 20, 70)

116 text("Radar Display", width / 2 - 80, 40)

117

118

119def draw\_sweep\_line(angle):

120 """Draws the moving line that sweeps across the radar."""

121 stroke(0, 255, 0, 150)

122 strokeWeight(3)

123 rad\_angle = radians(angle - 180)

124 end\_x = radar\_center\_x + radar\_radius \* cos(rad\_angle)

125 end\_y = radar\_center\_y + radar\_radius \* sin(rad\_angle)

126 line(radar\_center\_x, radar\_center\_y, end\_x, end\_y)

127

128

129def draw\_detected\_points():

130 """Draws a point on the radar for the current detection."""

131 max\_dist = 20.0

132 if current\_distance > 0 and current\_distance < max\_dist:

133 stroke(255, 0, 0)

134 strokeWeight(5)

135 rad\_angle = radians(current\_angle - 180)

136 mapped\_dist = map(current\_distance, 0, max\_dist, 0, radar\_radius)

137 point\_x = radar\_center\_x + mapped\_dist \* cos(rad\_angle)

138 point\_y = radar\_center\_y + mapped\_dist \* sin(rad\_angle)

139 point(point\_x, point\_y)

140 point\_history.append({'x': point\_x, 'y': point\_y, 'age': 255})

141

142def update\_and\_draw\_history():

143 """Draws and fades out old points to create a trail effect."""

144 global point\_history

145 new\_history = []

146 for p in point\_history:

147 stroke(0, p['age'], 0)

148 strokeWeight(4)

149 point(p['x'], p['y'])

150 p['age'] -= 2

151 if p['age'] > 0:

152 new\_history.append(p)

153 point\_history = new\_history

154

155

156def serialEvent(port):

157 """

158 This function is automatically called by Processing whenever new data is available.

159 """

160 global in\_string, current\_angle, current\_distance

161

162 while port.available() > 0:

163 in\_char = port.readChar()

164 if in\_char == '.':

165 values = in\_string.split(',')

166 if len(values) == 2:

167 try:

168 angle = int(values[0])

169 distance = int(values[1])

170 current\_angle = angle

171 current\_distance = distance

172 except ValueError:

173 pass

174 in\_string = ""

175 else:

176 if in\_char != '\n' and in\_char != '\r':

177 in\_string += in\_char

Arduino based radar system's monitor screen

For an Arduino-based radar system, you have several options for displaying the information collected by the sensor (typically an ultrasonic sensor)

.

1. Computer monitor with Processing software

* Functionality: This is a popular option that allows for a graphical, radar-like representation of the detected objects.
* Mechanism: The Arduino sends angle and distance data via serial communication to a computer running a Processing sketch.
* Display: The Processing sketch interprets the data and converts the polar coordinates (angle and distance) into Cartesian (x,y) points, plotting them on the screen. This creates a visual representation similar to a traditional radar screen, often with detected objects shown as fading "blips" and a green or red section indicating the presence of an object.
* Features: The display can show the object's angle and distance, and in some implementations, it can even indicate if an object is closer than a set threshold (e.g., triggering a red LED).

2. LCD display

* Functionality: A more compact and portable option, allowing for real-time display of distance readings and other relevant information directly on the device.
* Mechanism: The Arduino processes the sensor data and sends it to the LCD module for display.
* Features: Can display the distance to the detected object and can be programmed to show status messages or other critical information.

3. OLED display

* Functionality: Similar to LCDs, offering a real-time display but potentially with higher resolution and better contrast.
* Mechanism: The Arduino communicates with the OLED display, typically using the I2C interface, which reduces the number of pins required for communication.
* Features: Can be programmed to display detected objects as fading "blips" in a radar-style layout, similar to the Processing approach but on a smaller, embedded screen.

In summary

The choice of monitor screen depends on the specific needs of your Arduino-based radar system:

* For a graphical, interactive, and customizable radar display, a computer monitor with Processing software is the ideal choice.
* For a more compact and integrated system, an LCD or OLED display can provide essential information directly on the device.

Regardless of the chosen display, the Arduino acts as the central processor, gathering information from the ultrasonic sensor and controlling other components like the servo motor and buzzer to create a functional radar system.

Arduino based radar system's monitor screen

For an Arduino-based radar system, you have several options for displaying the information collected by the sensor (typically an ultrasonic sensor)

.

1. Computer monitor with Processing software

* Functionality: This is a popular option that allows for a graphical, radar-like representation of the detected objects.
* Mechanism: The Arduino sends angle and distance data via serial communication to a computer running a Processing sketch.
* Display: The Processing sketch interprets the data and converts the polar coordinates (angle and distance) into Cartesian (x,y) points, plotting them on the screen. This creates a visual representation similar to a traditional radar screen, often with detected objects shown as fading "blips" and a green or red section indicating the presence of an object.
* Features: The display can show the object's angle and distance, and in some implementations, it can even indicate if an object is closer than a set threshold (e.g., triggering a red LED).

2. LCD display

* Functionality: A more compact and portable option, allowing for real-time display of distance readings and other relevant information directly on the device.
* Mechanism: The Arduino processes the sensor data and sends it to the LCD module for display.
* Features: Can display the distance to the detected object and can be programmed to show status messages or other critical information.

3. OLED display

* Functionality: Similar to LCDs, offering a real-time display but potentially with higher resolution and better contrast.
* Mechanism: The Arduino communicates with the OLED display, typically using the I2C interface, which reduces the number of pins required for communication.
* Features: Can be programmed to display detected objects as fading "blips" in a radar-style layout, similar to the Processing approach but on a smaller, embedded screen.

In summary

The choice of monitor screen depends on the specific needs of your Arduino-based radar system:

* For a graphical, interactive, and customizable radar display, a computer monitor with Processing software is the ideal choice.
* For a more compact and integrated system, an LCD or OLED display can provide essential information directly on the device.

Regardless of the chosen display, the Arduino acts as the central processor, gathering information from the ultrasonic sensor and controlling other components like the servo motor and buzzer to create a functional radar system.