Aim of the Project:

To design and implement an advanced radar sensor integration system for industrial automation, ensuring enhanced safety, productivity, and operational efficiency through reliable object detection and tracking.

Problem Statement and Solution:

Problem Statement:

In industrial environments, traditional detection systems (e.g., proximity sensors) face limitations in range and adverse conditions such as dust, fog, or vibrations. These challenges hinder reliable object detection and tracking, reducing operational efficiency.

Proposed Solution:

Implement a system integrating radar sensors with a microcontroller to provide long-range, noise-filtered, and accurate real-time object detection and tracking. The system will include:

- **Signal Processing:** Efficient algorithms for noise reduction and movement tracking.
- **Web Interface:** A user-friendly platform for monitoring and controlling the system remotely.

Project Design Specification and Architecture:

Specifications:

- **Radar Sensors:** Long-range industrial-grade radar sensors (e.g., 24 GHz or 77 GHz FMCW radars).
- **Microcontroller Board:** Raspberry Pi Pico or ESP32 for computational efficiency.
- **Connectivity:** Wi-Fi-enabled for remote monitoring via a web interface.
- **Power Supply:** 5V DC power supply.

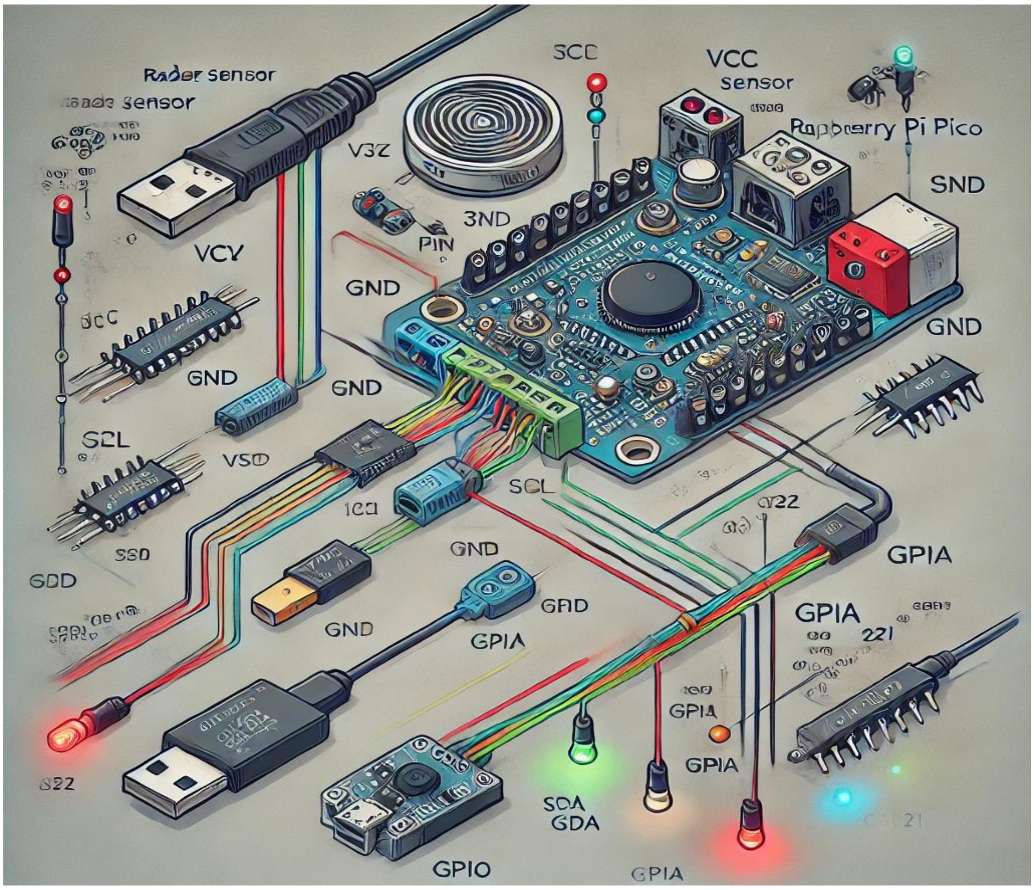
Architecture:

1. **Input:** Radar sensor detects objects and sends raw data.
2. **Processing:** The microcontroller processes the data, applies filters, and calculates object distance and movement.
3. **Output:** Data displayed via the web interface and used for automation triggers.

Wiring Diagram:

A clear wiring diagram illustrating:

- Radar sensor connections to the microcontroller (e.g., SPI/I2C protocol).

- Microcontroller to the computer (via USB).
- Power supply and any peripheral components (e.g., LEDs or alarms).
- 

KiCad PCB Design & Gerber File Submission:

KiCad Design:

- Schematic: Displays radar sensor, microcontroller, connectors, and power circuits.
- PCB Layout: Compact and industrial-grade.

Submission: Gerber files with complete layer information for manufacturing.

Components Working Principles/Functionality (20 Marks)

1. **Radar Sensors:**
    - **Working:** Emit electromagnetic waves and measure reflected signals to determine distance, speed, and position of objects.
    - **Advantages:** Effective in dust, fog, and low-visibility environments.
2. **Microcontroller (e.g., ESP32):**

    - Processes radar data using built-in computation units.
    - Enables connectivity to the web interface.
3. **Web Interface:**
    - Displays processed data (distance, speed, trajectory).
    - Interactive dashboard for manual controls.

Assembling Hardware Components & Coding:

Steps:

1. Mount radar sensors on an industrial frame.
2. Connect the sensors to the microcontroller using appropriate wiring.
3. Connect the microcontroller to a computer for coding.

Coding Workflow in Thonny IDE:

1. **Sensor Initialization:** Configure radar sensor parameters (e.g., frequency range).
2. **Data Filtering:** Implement signal processing techniques like Kalman filters.
3. **Output:** Transmit filtered data to the web interface using HTTP or WebSocket protocols.
4. **Coding:**

1. Setup and Sensor Initialization:

```
import machine
import time
import ujson
import network
from umqtt.simple import MQTTClient  # For sending data to a web interface


# Pin configuration for radar sensor (adjust according to your setup)
SENSOR_SCL = machine.Pin(22)  # Example GPIO pins for I2C
SENSOR_SDA = machine.Pin(21)


# Initialize I2C interface for the radar sensor
i2c = machine.I2C(0, scl=SENSOR_SCL, sda=SENSOR_SDA, freq=400000)
```

```
# Function to initialize the radar sensor
def initialize_sensor():
    print("Initializing radar sensor...")
    try:
        # Send initialization commands if required by the sensor
        i2c.writeto(0x68, b'\x00')  # Example sensor initialization
        print("Radar sensor initialized.")
    except Exception as e:
        print(f"Error initializing sensor: {e}")
        return False
    return True
```

2. Data Processing

```
# Function to read and process data from the radar sensor
def read_radar_data():
    try:
        # Read raw data from radar sensor (e.g., 4 bytes for distance and speed)
        raw_data = i2c.readfrom(0x68, 4)  # Replace 0x68 with your radar's I2C address
        distance = int.from_bytes(raw_data[:2], 'big') / 100.0  # Convert to meters
        speed = int.from_bytes(raw_data[2:], 'big') / 100.0  # Convert to m/s

        # Filter noise using a simple threshold (or Kalman filter for advanced processing)
        if distance > 0.1:  # Ignore readings below 10 cm
            return {"distance": distance, "speed": speed}
    except Exception as e:
        print(f"Error reading radar data: {e}")
        return None
```

Web Interface Integration:

```
# Configure Wi-Fi connection
SSID = "Your_SSID"
PASSWORD = "Your_PASSWORD"


def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(SSID, PASSWORD)
```

```
    while not wlan.isconnected():
        print("Connecting to Wi-Fi...")
        time.sleep(1)
    print("Wi-Fi connected:", wlan.ifconfig())

# MQTT setup for sending data to the web interface
MQTT_BROKER = "broker.hivemq.com"
CLIENT_ID = "RadarIntegrationSystem"
TOPIC = "industrial/radar/data"

client = MQTTClient(CLIENT_ID, MQTT_BROKER)

def send_data_to_web(data):
    try:
        client.connect()
        json_data = ujson.dumps(data)
        client.publish(TOPIC, json_data)
        client.disconnect()
        print(f"Data sent: {json_data}")
    except Exception as e:
        print(f"Error sending data: {e}")
```

5.Main Program:

```
        if __name__ == "__main__":
    connect_wifi()

    if initialize_sensor():
        while True:
            radar_data = read_radar_data()
            if radar_data:
                print(f"Distance: {radar_data['distance']} m, Speed: {radar_data['speed']} m/s")
```

```
    send_data_to_web(radar_data)
time.sleep(0.5)  # Delay for continuous readings
```

Project Output:

- Real-time object detection and movement tracking displayed on the web interface.
- Triggered actions for automation systems (e.g., stopping a conveyor belt if an object is detected).
