THE OMNIGUARD: LIDAR-RADAR SURVEILLANCE

CONTENT

CHAPTER **Page No.**

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to **Dr. K. Prakasan**, Principal, PSG College of Technology, for his kind patronage.

We wish to extend our sincere gratitude to **Dr. J. Arunshankar**, Professor and Head, Department of Instrumentation and Control Systems Engineering, for his moral support and valuable advice.

We extend our gratitude to **Dr. S. Esakkirajan**, Professor, UG Programme Coordinator, Department of Instrumentation and Control Systems Engineering, for his moral support and valuable advice.

We sincerely thank our faculty guide **Ms M Saranya**, Assistant Professor (Sr. Gr.), Department of Instrumentation and Control Systems Engineering, for giving us an opportunity to pursue our academic interests and for his constant support throughout the course of the project. We are grateful for his inspiring guidance, which motivated us from the inception of the project.

We wish to convey our regards to **Ms. Sonya**, Associate Professor,Department of Instrumentation and Control Systems Engineering, PSG College of Technology, for their support and valuable suggestions.

We are thankful to all our friends, faculty members, and non-teaching staff of the Department of Instrumentation and Control Systems Engineering for extending their cooperation toward the completion of this project work.

SYNOPSIS

OMNI GUARD: LIDAR-RADAR SURVEILLANCE is a cutting-edge 360-degree monitoring system designed to address the limitations of traditional CCTV surveillance, such as blind spots and poor visibility in low-light conditions. By integrating LiDAR for high-resolution 3D mapping and radar for detection in adverse environments like fog and darkness, this project enhances security and industrial safety. A Raspberry Pi serves as the processing unit, enabling real-time data analysis and remote monitoring, making the system efficient, scalable, and cost-effective.

The motivation behind this project arises from the growing need for high-accuracy, continuous surveillance in security enforcement, smart cities, and industrial applications. Unlike conventional cameras, LiDAR and radar provide superior object detection capabilities with minimal dependency on lighting conditions. By leveraging these advanced technologies, the project aims to develop a robust and reliable surveillance system suitable for various environments.

The methodology includes selecting and integrating LiDAR, radar, and Raspberry Pi, developing sensor fusion algorithms for enhanced detection, and implementing a ROS-based software framework for real-time data processing. The system undergoes rigorous testing under different conditions to ensure accuracy and efficiency before deployment. Potential applications include intrusion detection, traffic monitoring, and industrial hazard prevention.

The expected outcomes include a fully functional 360-degree surveillance system capable of detecting objects accurately even in challenging visibility conditions. The system also enables remote monitoring for enhanced security. However, challenges such as high sensor costs, data storage limitations, and the computational constraints of Raspberry Pi for complex AI-based analytics remain.

This project lays the foundation for a next-generation surveillance system, with future improvements including AI-driven object recognition and cloud-based monitoring. It represents a significant step toward improving security, automation, and smart city applications through advanced sensing technologies.

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

The advancement of sensor technologies has revolutionized surveillance systems, enhancing their ability to monitor environments in real time and detect potential threats with greater accuracy. Traditional surveillance systems, such as CCTV cameras, are often limited by factors like blind spots, poor visibility in low-light conditions, and the inability to detect objects through obstacles such as fog or smoke. The integration of LiDAR (Light Detection and Ranging) and radar technology overcomes these challenges by providing comprehensive spatial awareness and precise object detection. LiDAR uses laser pulses to create high-resolution 3D maps, while radar employs radio waves to detect the presence, speed, and movement of objects, making the combination of both technologies ideal for 360-degree surveillance applications.

The primary advantage of 360-degree surveillance lies in its ability to continuously monitor an area from all directions, minimizing blind spots and ensuring complete situational awareness. The integration of LiDAR and radar enhances this capability by providing a detailed representation of the surroundings and detecting objects even in adverse conditions such as heavy rain, fog, or darkness. LiDAR's high-resolution mapping ensures accurate distance measurement and object classification, while radar's ability to penetrate environmental obstructions ensures reliable detection even in the absence of visual clarity. This combination enables a robust surveillance system that can function effectively in critical applications, including perimeter security, traffic monitoring, and industrial hazard detection.

The Raspberry Pi plays a crucial role in this surveillance system by serving as the central processing unit responsible for collecting, analyzing, and transmitting data from LiDAR and radar sensors. As an affordable and compact computing platform, the Raspberry Pi allows for the integration of multiple sensors, real-time data processing, and wireless communication for remote monitoring. Its low power consumption and versatility make it suitable for both stationary and mobile surveillance applications, such as security drones, autonomous vehicles, and smart city infrastructure.

1.1 BACKGROUND OF THE PROJECT

The increasing demand for effective and intelligent surveillance systems has driven the need for innovative solutions that provide accurate and continuous monitoring. Traditional surveillance systems rely on cameras, which are limited by factors such as lighting conditions, blind spots, and difficulty in detecting objects in foggy or smoky environments. To overcome these limitations, integrating LiDAR and radar offers a more advanced approach, enabling 360-degree surveillance with enhanced object detection and environmental awareness.

1.2 FACTORS DRIVING LIDAR AND RADAR FOR OBJECT DETECTION

I . Enhanced Environmental Perception
- Provides accurate 3D spatial mapping with LiDAR.
- Enables reliable detection in low-visibility conditions using radar.
- Enhances perception through sensor fusion techniques.

II. Improved Object Tracking and Classification
- Increases detection accuracy with high-resolution LiDAR point clouds.
- Enables motion tracking using radar's Doppler shift capabilities.
- Facilitates AI-based classification by combining sensor data.

III. Robust Performance in Dynamic Environments
- Ensures real-time obstacle detection for autonomous navigation.
- Adapts to varying weather and lighting conditions.
- Reduces false positives through sensor redundancy and fusion.

IV. Economic and Safety Benefits
- Enhances safety in autonomous and assisted driving systems.
- Reduces operational costs by minimizing collision risks.
- Improves efficiency in industrial and security applications.

1.3 OBJECTIVE

- To integrate LiDAR and radar sensors with Raspberry Pi for enhanced surveillance and object detection.
- To develop a 360-degree monitoring system capable of real-time data acquisition and analysis.
- To implement sensor fusion techniques for improved accuracy and reliability in object detection.
- To design a communication system that enables remote monitoring and data transmission.
- To evaluate the system’s performance under different environmental conditions and improve detection capabilities.

1.4 GOAL OF THE PROJECT

This project aims to develop an advanced surveillance system integrating LiDAR and radar technology to enhance security, monitoring, and automation across various fields. Traditional surveillance methods like CCTV and motion sensors often struggle with blind spots, poor visibility in low-light conditions, and limited object detection. By leveraging LiDAR’s high-resolution 3D mapping and radar’s robust object detection, this system ensures real-time tracking, improved environmental awareness, and reliable threat detection, making it a valuable solution for modern security challenges. Its versatility allows applications in security enforcement, smart city infrastructure, and industrial safety. In security, it can be used for perimeter surveillance, intrusion detection, and threat assessment, while in smart cities, it aids in traffic management, pedestrian safety, and intelligent transportation. Industrial applications include monitoring safety conditions, detecting unauthorized access,

and providing early warnings of potential hazards. Additionally, cloud computing and IoT integration can enable remote monitoring and real-time data access, making the system more adaptable to dynamic environments. Through these innovations, this project will contribute to the development of next-generation surveillance systems, enhancing security, automation, and situational awareness across various industries.

1.5 OUTCOMES OF THE PROJECT

The project focuses on designing and implementing a 360-degree surveillance system using LiDAR and radar integration with Raspberry Pi. The system aims to provide real-time object detection, environmental mapping, and continuous monitoring in various applications, such as security enforcement, smart cities, and industrial safety. The scope of the project includes:

1. Integration of LiDAR and Radar Sensors: The system will use LiDAR for high-precision 3D mapping and radar for robust object detection in various environmental conditions.
2. 360-Degree Surveillance Capability: The system will be designed to provide complete coverage by either using multiple fixed sensors or a motorized rotating setup.
3. Real-Time Data Processing: Raspberry Pi will serve as the central processing unit to collect, analyze, and process sensor data in real-time.
4. Applications in Security, Smart Cities, and Industrial Safety: The system can be deployed in multiple environments, including restricted areas, traffic monitoring, and safety enforcement.

1.6 METHODOLOGY

The methodology of this project involves a structured approach to designing and implementing a 360-degree surveillance system using LiDAR and radar integration with Raspberry Pi. The key stages include system design, hardware selection, software development, sensor integration, data processing, and testing. Below is a detailed breakdown of the methodology:

1.6.1 SYSTEM DESIGN AND ARCHITECTURE DEVELOPMENT

The first step in the methodology involves defining the overall system architecture and functional requirements. This includes:

- Determining the hardware components, such as LiDAR, radar, Raspberry Pi, power supply, and communication modules.
- Selecting a 360-degree coverage approach, either using multiple fixed sensors or a motorized rotating setup.
- Designing the software architecture, including data processing algorithms and communication protocols.

1.6.2 HARDWARE SELECTION AND INTEGRATION

The hardware components are selected based on performance, cost, and compatibility with Raspberry Pi. The key components include:

- LiDAR Sensor: Provides high-resolution 3D mapping and object detection capabilities.
- Radar Sensor: Ensures object detection even in low-visibility conditions (fog, rain, darkness).
- Raspberry Pi: Serves as the central processing unit for data acquisition, fusion, and communication.
- Power Supply: Ensures uninterrupted operation, either through a battery or direct power source.
- Communication Module: Wi-Fi, Ethernet, or LoRa module for remote monitoring and data transmission.

After selection, these components are physically integrated, and wiring is done for sensor interfacing with Raspberry Pi.

1.6.3 SOFTWARE DEVELOPMENT AND SENSOR FUSION

- ROS Installation: ROS is installed on Raspberry Pi for managing LiDAR and radar data processing.
- LiDAR and Radar Drivers: ROS packages like rplidar_ros (for LiDAR) and mmWave radar drivers are installed and configured.
- VNC Viewer Setup: Enables remote access for monitoring, debugging, and real-time visualization using ROS tools (RViz).
- Sensor Communication: LiDAR interfaces via USB/UART, while radar connects via SPI/I2C for seamless data acquisition.
- Sensor Fusion Algorithm: Combines LiDAR's 3D mapping with radar's motion detection to enhance object detection accuracy.

This setup ensures real-time object tracking, efficient data processing, and remote system control.

1.6.4 SYSTEM TESTING AND CALIBRATION

Once the hardware and software are integrated, the system undergoes rigorous testing to ensure accuracy and reliability. The testing includes:

- Calibration of Sensors: Adjusting LiDAR and radar sensitivity for accurate object detection.
- Performance Testing: Checking response time, range accuracy, and real-time monitoring capabilities.
- Environmental Testing: Evaluating system performance in different conditions such as low light, fog, and high traffic areas.
- Error Handling: Implementing fail-safe mechanisms to handle sensor malfunctions or data inconsistencies.

1.6.5 DEPLOYMENT AND APPLICATION TESTING

- System Deployment: The integrated system is installed in real-world environments such as security zones, traffic monitoring areas, and industrial sites.
- Field Testing: Performance is evaluated under various conditions (day/night, fog, rain, and high-traffic scenarios).
- Calibration & Optimization: LiDAR and radar sensors are fine-tuned for improved accuracy and detection range.
- Error Handling & Debugging: System logs are analyzed to identify and resolve errors in data processing and communication.
- Performance Evaluation: Metrics such as object detection accuracy, response time, and system reliability are assessed.

This phase ensures the system functions effectively in real-world applications and is optimized for future advancements.

1.7 FLOW OF THE PROJECT

Fig 1.1 depicts the sequential process of system development, starting from the initial design phase to deployment and testing. The flowchart begins with the "START" node, followed by "System Design and Architecture Development," where the fundamental framework of the system is established. It then proceeds to "Hardware Selection and Integration," ensuring that appropriate components are chosen and connected. The next phase, "Software Development and Sensor Fusion," involves coding and combining sensor data for optimal performance. Afterward, "System Testing and Calibration" ensures accuracy and reliability before moving to "Deployment and Application Testing," where the system undergoes real-world evaluation. The process concludes at the "END" node, marking the completion of the development cycle.

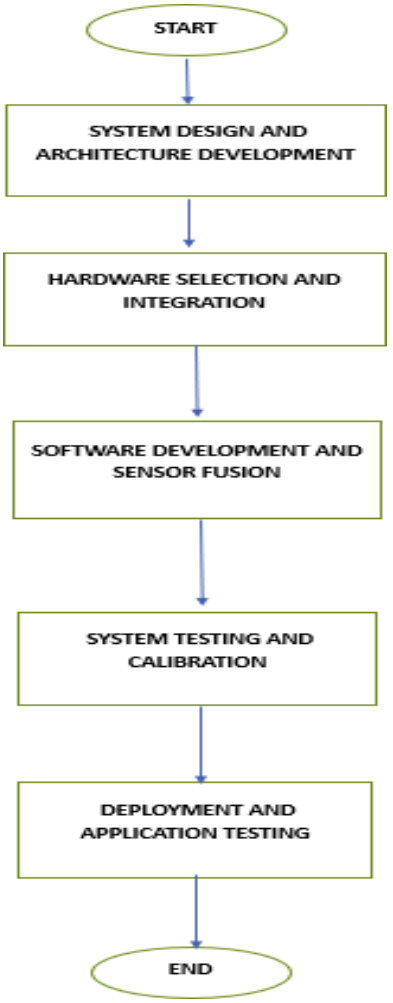

Fig 1.1 FLOWCHART DESCRIBING THE FLOW OF THE PROJECT

CHAPTER 2
LITERATURE SURVEY

The development of 360-degree surveillance systems has been extensively studied in various fields, including security, autonomous navigation, industrial monitoring, and smart city applications. Traditional surveillance methods, primarily relying on fixed and pan-tilt cameras, face limitations such as blind spots, high dependency on lighting conditions, and difficulty in detecting objects in real-time. As a result, researchers have explored multi-sensor integration, real-time processing algorithms, and automated threat detection systems to enhance surveillance capabilities.

Existing literature covers different approaches to 360-degree surveillance, including image stitching from multiple cameras, AI-powered anomaly detection, and sensor fusion techniques. Studies have highlighted the importance of computational efficiency in embedded systems, emphasizing the role of lightweight processing units such as Raspberry Pi in enabling cost-effective, scalable surveillance solutions. Additionally, research on robotic surveillance and autonomous monitoring has explored how ROS (Robot Operating System) can facilitate sensor data management, object tracking, and real-time decision-making.

This literature survey reviews key research papers, case studies, and technological advancements in automated surveillance. It focuses on existing methodologies, challenges in implementation, and the role of embedded computing platforms. By analyzing previous work, this survey provides insights into best practices, potential improvements, and the feasibility of integrating LiDAR, radar, and AI-driven analytics for a more robust and intelligent 360-degree surveillance system.

2.1 MOVING OBJECT DETECTION USING ULTRASONIC RADAR [1]

Table 2.1 ANALYSIS 1

Section	Summary
Objective	Develop a cost-effective ultrasonic radar system to detect moving objects, measure distance, speed, and direction, and improve detection accuracy.

System Components	- HC-SR04 Ultrasonic Sensor: Measures distance via sound waves. - Servo Motor:Rotates sensor (0°–180°) for wider scanning. - Arduino UNO:Processes data and runs detection algorithms. - Data Processing Algorithm:Calculates velocity based on displacement over time.
Key Findings	- Detects both moving and stationary objects. - Measures speed and direction based on displacement. - Differentiates metal vs. non-metal objects (higher accuracy for metals). - Provides real-time data but with precision and range limitations.
Comparison with 360° Surveillance	Similarities: - Uses a rotating sensor, similar to LiDAR/radar. - Captures and updates object positions in real-time. Limitations: - Short detection range (3.6m indoors, 2.5m outdoors) vs. LiDAR (200m) & radar (500m+). - Limited object classification (only metal vs. non-metal). - Affected by environmental factors (temperature, humidity). - Lacks AI-based classification and threat detection.
Recommendations	- LiDAR & Radar Integration:Improves detection range & accuracy. - AI-based Classification:Enables identification of humans, vehicles, and threats. - Raspberry Pi with ROS: Enhances real-time processing and remote monitoring. - Multiple Sensors & Faster Rotation: Ensures seamless 360° coverage. -Cloud Storage & Secure Data

	Transmission:Allows centralized monitoring and data security.
Conclusion	The ultrasonic radar system is a low-cost, real-time object detection solution but has limitations in range, accuracy, and classification. Integrating LiDAR, radar, AI, and ROS would enhance its capabilities, making it more suitable for modern surveillance and smart city applications.

The table 2.1 describes the study of Moving object detection using ultrasonic radar involves using ultrasonic sensors to emit sound waves and analyze their reflections to detect and track objects in real-time. This system is widely used in security, automation, and robotic applications for obstacle avoidance and motion sensing.

2.2 ENHANCED OBJECT DETECTION IN AUTONOMOUS VEHICLE THROUGH LIDAR- CAMERA SENSOR FUSION [2]

Table 2.2 ANALYSIS 2

Objective	Improve object detection and tracking in autonomous vehicles using LiDAR-camera sensor fusion.
Challenges	- LiDAR provides precise 3D data but lacks color/texture. - Cameras offer high-resolution images but struggle with depth and lighting.
Solution	- Combining LiDAR and camera for enhanced object detection and tracking. - YOLO for image detection and PointPillars for LiDAR point cloud processing. - Improved DeepSORT tracking algorithm with Unscented Kalman Filtering.

Methodology	1. LiDAR-Camera Calibration: Aligning LiDAR 3D data with camera 2D images. 2. Object Detection & Fusion: YOLO for camera, PointPillars for LiDAR, IoU matching for alignment, D-S Theory for confidence fusion. 3. Object Tracking: DeepSORT algorithm enhanced with motion data for stable tracking.
Experimental Setup	- Dataset: KITTI - Hardware: RoboSense LiDAR, USB Monocular Camera, Intel i5-12400, RTX 3060 GPU - Software: PyTorch
Results	1. Object Detection: - Daytime: 97.3% (cars), 95.4% (pedestrians) - Nighttime: 94.1% (cars), 92.5% (pedestrians) 2. Object Tracking: - MOTA: 66% (+10% improvement) - MOTP: 79% (+5% improvement) - Reduced identity switching
Effectiveness	- Improved classification confidence with D-S fusion. - IoU matching improved localization accuracy. - Enhanced tracking stability in occlusion scenarios.
Conclusion	- Multi-sensor fusion improves real-time surveillance. - Applicable for security, traffic monitoring, and automated threat detection. - Future work: AI-driven anomaly detection, cloud-based real-time processing, and radar integration.

Table 2.2 describes Enhanced object detection in autonomous vehicles through LiDAR-camera sensor fusion that combines LiDAR's precise depth sensing with a camera's detailed visual data for improved accuracy and environmental perception. This integration enhances obstacle detection, lane recognition, and real-time navigation in diverse driving conditions.

2.3 AUTOMATIC 360◦ MONO-STEREO PANORAMA GENERATION USING A COST-EFFECTIVE MULTI-CAMERA SYSTEM [3]

Table 2.3 ANALYSIS 3

Objective	Develop a cost-effective 360-degree panorama system for mono and stereo panoramas.
Challenges	- Existing systems rely on expensive hardware and manual image stitching. - Need for seamless image alignment and blending to avoid artifacts.
Solution	- Multi-camera setup automating panorama generation with minimal user input. - Uses six cameras for mono panoramas and three for stereo panoramas. - Feature-based image stitching ensures artifact-free results.
Methodology	1. Data Acquisition: - Six fisheye cameras (mono), three cameras (stereo). - Optimized placement for seamless coverage. 2. Camera Calibration: - ORB feature extraction, RANSAC feature matching, and bundle adjustment. 3. Image Stitching: - Homography-based alignment, multi-band blending, and panorama straightening.
Experimental Setup	- Implemented in C++ with Nvidia Stitching SDK. - Hardware: GeForce-Titan-X 1060, 3.3 GHz CPU, 8GB RAM. - Evaluation: Qualitative (visual quality) and quantitative (image quality metrics).
Mono Panorama Results	- Automatic parallax-free image stitching. - Outperformed Autostitch, Panoweaver, and Kolor Autopano. - No visible seams or distortions.

Stereo Panorama Results	- Left-right stereo views blended for depth perception. - Seamless 3D effect when viewed on an HMD. - Accurate depth-aware representation.
Performance Metrics	- Peak Signal-to-Noise Ratio (PSNR). - Structural Similarity Index (SSIM). - Root Mean Square Error (RMSE). - Higher accuracy and speed compared to existing methods.
Conclusion	- Efficient, low-cost 360° surveillance system. - Ideal for security, urban monitoring, and law enforcement. - Future work: AI-driven behavior analysis, LiDAR augmentation, cloud-based monitoring.

Table 2.3 describes the automatic 360° mono-stereo panorama generation using a cost-effective multi-camera system, which captures and stitches images from multiple cameras to create seamless panoramic views. This approach enhances immersive imaging for applications in virtual reality, surveillance, and autonomous navigation.

2.4 A ROBUST SINGLE AND MULTIPLE MOVING OBJECT DETECTION, TRACKING, AND CLASSIFICATION [4]

Table 2.4 ANALYSIS 4

Section	Details
Introduction	The paper presents an advanced method for single and multiple moving object detection, tracking, and classification. The system improves real-time surveillance by addressing challenges like occlusion handling and computational efficiency.
Detection Phase	Uses Mixture of Adaptive Gaussian (MoAG) for foreground segmentation and Fuzzy Morphological Filtering for noise reduction.

Tracking Phase	Employs Blob Detection and Analysis for tracking multiple moving objects in dynamic environments.
Evaluation Phase	Utilizes J48 decision tree classifier for object classification, distinguishing between pedestrians, vehicles, and stationary objects.
Dataset	Tested on datasets including MOT17, PETS2009, and football surveillance videos.
Performance Metrics	Evaluated using Precision, Recall, F-measure, Peak Signal-to-Noise Ratio (PSNR), and Mean Squared Error (MSE).
Comparison with Existing Methods	Outperforms traditional methods like Gaussian Mixture Models (GMM), Optical Flow Tracking, and Kalman Filtering in detection accuracy and tracking robustness.
Processing Speed	Achieves real-time performance with an average processing speed of 0.025 seconds per frame.
Conclusion	Provides an efficient and scalable solution for security surveillance, traffic monitoring, and autonomous surveillance drones. Future improvements may include AI-driven behavior analysis and deep learning-based object recognition.

Table 2.4 describes a robust system for single and multiple moving object detection, tracking, and classification, utilizing advanced algorithms to analyze motion patterns and distinguish objects in real time. This system enhances accuracy in surveillance, autonomous navigation, and intelligent monitoring applications.

The proposed system efficiently detects, tracks, and classifies moving objects in real-time using Mixture of Adaptive Gaussian (MoAG) segmentation and Fuzzy Morphological Filtering to reduce noise. Blob Detection ensures accurate tracking, while the J48 Decision Tree enhances classification. Compared to traditional methods, it improves accuracy, reduces false detections, and ensures stability. With a 0.025-second processing time, it excels in surveillance, security, and traffic monitoring under dynamic conditions.

CHAPTER 3
DESIGN AND DEVELOPMENT

The rapid advancements in sensor technology, artificial intelligence (AI), and embedded systems have significantly transformed modern surveillance, enabling real-time object detection, tracking, and analysis. A 360-degree surveillance system integrates multiple sensing modalities, such as LiDAR, radar, ultrasonic sensors, and cameras, to provide comprehensive monitoring coverage. Unlike conventional fixed-angle camera systems that suffer from blind spots and environmental limitations, this multi-sensor approach enhances accuracy, reliability, and adaptability in complex environments. Such systems are essential for applications in security enforcement, autonomous navigation, industrial safety, and smart city management, where continuous and precise situational awareness is crucial.

To achieve effective 360-degree surveillance, a structured sensor fusion framework is implemented, ensuring efficient data acquisition, real-time processing, and intelligent decision-making. The system follows a layered architecture, beginning with data collection from various sensors, followed by pre-processing to filter noise and enhance useful features. The data fusion layer merges sensor inputs to improve detection accuracy, while the decision layer employs AI-driven algorithms and security protocols to analyze the processed data and trigger appropriate actions.

3.1 BLOCK DIAGRAM

The block diagram in Fig 3.1 represents a structured approach to designing a 360-degree surveillance system, ensuring efficient data handling and decision-making. The system processes information in a stepwise manner, allowing better object detection, classification, and tracking.

1. Data Collection Layer serves as the foundation, gathering raw data from multiple sensors, including LiDAR, radar, ultrasonic sensors, and cameras. This layer ensures wide-area coverage and multi-sensor redundancy, allowing the system to function under varying conditions.
2. Pre-Processing Layer enhances the quality of raw sensor data by removing noise, filtering distortions, and calibrating sensor readings. This step is crucial in improving object detection accuracy before data is analyzed.
3. Data Fusion Layer combines multiple sensor outputs to create a comprehensive environmental model. It eliminates inconsistencies and ensures high-precision object recognition and tracking by merging LiDAR depth maps, radar signals, and camera feeds.

4. Decision Layer takes the fused data and applies AI-based analysis, machine learning models, and predefined security protocols to classify objects, detect threats, and issue necessary alerts or responses.

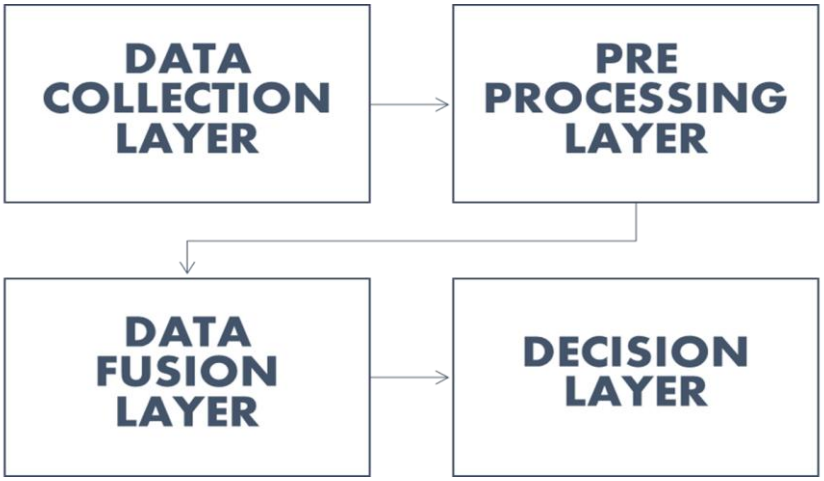

Fig 3.1 DESIGN BLOCK DIAGRAM

3.2 DESCRIPTION OF EACH LAYER

3.2.1 DATA COLLECTION LAYER

- This layer acts as the sensory input system of the surveillance setup.
- It includes multiple sensors such as LiDAR (for depth perception), radar (for object detection in adverse conditions), ultrasonic sensors (for short-range proximity detection), and cameras (for visual tracking and identification).
- The main objective is to capture a complete 360-degree view of the surroundings by gathering real-time raw data.
- Ensures redundancy and reliability by using multiple sensor modalities for robust surveillance.

3.2.2. PRE-PROCESSING LAYER

- The raw data collected by sensors often contains noise, distortions, and irrelevant background details.
- This layer applies signal filtering, noise reduction, and feature extraction to refine the data.
- Algorithms such as Kalman filtering, morphological filtering, and edge enhancement techniques improve the clarity and quality of the collected information.
- Sensor data is also calibrated and synchronized to ensure accurate real-time tracking.

3.2.3. DATA FUSION LAYER

- The most crucial layer where data from multiple sensors is merged to create a unified situational awareness model.
- It eliminates data inconsistencies, ensuring accurate object detection, classification, and tracking.
- Techniques such as Bayesian inference, sensor fusion algorithms, and deep learning-based fusion methods are used to merge multiple inputs into a single decision-making framework.
- This integration significantly improves the system's ability to track multiple objects, even in complex and dynamic environments.

3.2.4. DECISION LAYER

- The final stage where the processed and fused data is analyzed using machine learning algorithms, AI-based decision models, and predefined rule-based logic.
- This layer enables intelligent threat detection, anomaly recognition, and real-time security actions such as issuing alerts, activating security responses, or triggering automated defensive mechanisms.
- It can be further enhanced by cloud-based analytics, predictive modeling, and AI-driven behavior recognition for advanced surveillance applications.

3.3 HARDWARE COMPONENTS

A 360-degree surveillance system ensures continuous situational awareness by integrating LiDAR and radar with Raspberry Pi 4 for high-precision monitoring. Unlike traditional camera-based surveillance, it eliminates blind spots and adapts to varying environmental conditions. The RPLIDAR A1M8 generates a 360-degree environmental map, enabling accurate object detection and tracking. The RCWL-0516 radar enhances motion detection, even in low visibility conditions like fog or darkness. Real-time data processing on Raspberry Pi 4 ensures rapid analysis and decision-making for security, automation, and navigation applications.

3.3.1 Raspberry Pi 4

The Raspberry Pi 4 in Fig 3.2 serves as the core processing unit for the 360-degree surveillance system, handling data acquisition, sensor fusion, and real-time decision-making. With a quad-core ARM Cortex-A72 processor (1.5 GHz) and multiple RAM options, it ensures fast, low-latency processing for object detection and tracking. Its support for GPIO, I2C, SPI, and UART enables seamless integration with LiDAR and radar sensors. Low power consumption makes it ideal for 24/7 surveillance, while connectivity options like Wi-Fi and USB 3.0 enhance remote

monitoring.Raspberry Pi 4 is chosen for object detection due to its affordability, compact size, and ability to run AI-based analytics on a Linux-based OS. It efficiently processes data from the RPLIDAR A1M8 and RCWL-0516 sensors, applying sensor fusion algorithms to enhance detection accuracy. Additionally, its support for remote access via VNC Viewer, SSH, and cloud-based platforms makes real-time monitoring more accessible. The combination of computational power, energy efficiency, and flexible software support makes Raspberry Pi 4 an ideal choice for advanced surveillance applications.

Fig 3.2 RASPBERRY PI 4 [5]

3.3.2 RPLIDAR A1M8

The RPLIDAR A1M8 shown in Fig 3.3 is a high-precision 2D LiDAR sensor used for environmental mapping and object detection. It scans the surroundings using laser pulses, measuring distances with high accuracy to generate a 360-degree view. With a range of up to 12 meters and a rotational speed of 5–10 Hz, it provides real-time point cloud data for detailed mapping and motion tracking. The Raspberry Pi 4 processes this data, integrating it with radar motion detection for enhanced surveillance.We chose the RPLIDAR A1M8 for its ability to provide continuous 360-degree scanning, unaffected by lighting conditions, ensuring reliable object detection in low-visibility environments. Unlike cameras, LiDAR offers precise spatial mapping, improving intrusion detection and motion tracking without blind spots.

Fig 3.3 RPLIDAR A1M8 SETUP [6]

3.3.3 RCWL-0516:

The RCWL-0516 depicted in Fig 3.4 is a microwave radar motion sensor that detects movement using Doppler radar principles. Unlike PIR sensors, it works through obstacles like walls and fog, ensuring reliable all-weather surveillance. Operating at 3.18 GHz, it detects motion within 5–7 meters by analyzing reflected microwave signals. Its high sensitivity allows for instant detection, even in dark or obstructed environments.

We chose the RCWL-0516 for its ability to enhance motion detection beyond LiDAR and cameras, ensuring precise object tracking in all conditions. By integrating radar with LiDAR data, the system improves accuracy, reduces false alarms, and enables adaptive surveillance for efficient monitoring.

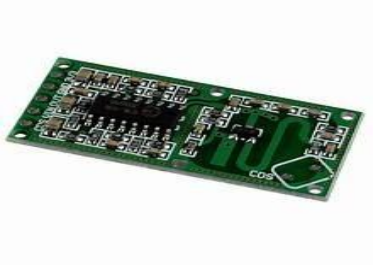

Fig 3.4 RADAR BASE- RCWL 0516 [7]

3.4 SOFTWARE

The effectiveness of a 360-degree surveillance system depends not only on its hardware components but also on its software framework, which ensures seamless communication, real-time data processing, remote access, and system security. The integration of various software tools allows for the efficient handling of sensor data, multi-threaded processing, and real-time monitoring. The primary software components utilized in this surveillance system include Raspberry Pi Imager, ROS (Robot Operating System), VNC Viewer, PuTTY, Advanced IP Scanner, and SSH (Secure Shell). Each software plays a significant role in different aspects of system functionality, from initial setup and configuration to real-time operation and remote management.

Table 3.1 Key Software Components in 360-Degree Surveillance System

Component	Description & Role in System
Raspberry Pi Imager	Official tool for installing OS onto microSD cards for Raspberry Pi. Simplifies setup and allows configuration of network and SSH settings. Supports multiple OS versions optimized for real-time processing.

ROS (Robot Operating System)	Framework for managing sensor data and real-time object detection. Uses a node-based architecture for modular processing, integrates object tracking algorithms, and supports visualization and mapping libraries.
VNC Viewer	Enables remote desktop access to Raspberry Pi GUI, allowing real-time monitoring of LiDAR and radar data. Facilitates wireless system control and troubleshooting.
PuTTY	Terminal emulator and SSH client for secure remote command execution. Supports file transfer, debugging, and automated system management via scripting.
Advanced IP Scanner	Network scanning tool for detecting Raspberry Pi's IP address, monitoring security vulnerabilities, and ensuring proper connectivity.
SSH (Secure Shell)	Secure remote access protocol enabling command execution, system configuration, and maintenance automation while ensuring encrypted communication.

Table 3.1 outlines the key software components used in a 360-degree surveillance system, detailing their primary functions and roles. These components include data acquisition, processing, and visualization tools that enable real-time monitoring and efficient management of the surveillance data captured by the system.

The successful implementation of the 360-degree surveillance system depends on the seamless integration of hardware and software components. The Raspberry Pi 4 serves as the central processor, managing data from the RPLIDAR A1M8 and RCWL-0516 radar sensor for precise object detection and environmental mapping. Supporting hardware, including a power supply and communication modules, ensures reliable operation. On the software side, tools like Raspberry Pi Imager, ROS, VNC Viewer, PuTTY, SSH, and Advanced IP Scanner facilitate setup, sensor fusion, remote access, and secure communication. This integration enables high-accuracy monitoring, adaptability, and scalability for security, industrial, and autonomous applications. Future enhancements could include AI-driven threat detection, machine learning-based object classification, and IoT-based cloud integration.

CHAPTER 4

INSTALLATION SETUP

This chapter gives a detail explanation about the installation setup that was used to interface the sensors

4.1 ROS INSTALLATION IN UBUNTU 18.4 (WSL)

Windows Subsystem for Linux (WSL) allows users to run a full Linux environment directly on Windows without requiring a virtual machine or dual-boot setup. This guide will walk you through the installation of Ubuntu 18.04 on WSL, covering everything from enabling WSL to configuring Ubuntu for first-time use.

STEP 1: INSTALLATION OF UBUNTU (18.4)

1.1 ENABLE WSL ON WINDOWS

1. Open powershell as administrator
   - Press Win + X, then select Windows Terminal (Admin) or PowerShell (Admin).
   - Alternatively, search for PowerShell in the Start Menu, right-click, and choose Run as administrator.

2. Enable WSL
   Run the following command in the PowerShell terminal: wsl --install
   - This command installs the latest version of WSL (including WSL 2) and sets it up automatically.
   - If WSL is already installed, you will see a message confirming it.

3. Enable virtual Machine Platform
   To use WSL 2, you must enable the Virtual Machine Platform feature.
   Run this command in PowerShell:
   dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

4. Set WSL2 as default
   For better performance and compatibility, set WSL 2 as the default version:
   Command : wsl --set-default-version 2

5. Restart your computer
   Once done, restart your computer to apply changes.

1.2 INSTALL UBUNTU 18.04 ON WSL

Once WSL is enabled, you can install Ubuntu 18.04 from the Microsoft Store.
Open Microsoft Store

- Press Win + S and search for Microsoft Store.
- Open the store and search for Ubuntu 18.04 LTS.
- Click Get → then click Install.

Launch Ubuntu 18.04

- After installation, open Start Menu → search for Ubuntu 18.04 → click to launch.
- The first-time setup may take a few minutes as it initializes the Linux file system.

1.3 SET UP UBUNTU 18.04

Create a New User

- When prompted, enter a username and password.
- This user will have sudo (admin) privileges, allowing you to install software and configure the system.

Update and Upgrade System Packages;

Once inside Ubuntu, update package lists and install the latest updates using

Command: sudo apt update && sudo apt upgrade -y

This ensures you have the latest security patches and system improvements.

1.4 VERIFY INSTALLATION

To confirm that Ubuntu 18.04 is properly installed and running, check the system details: lsb_release -a

Expected Output:

Distributor ID: Ubuntu
Description: Ubuntu 18.04.x LTS
Release: 18.04
Codename: bionic

STEP 2: ADD ROS PACKAGE REPOSITORY

To install ROS, we need to add the official ROS package repository to our system.

2.1 SETUP SOURCE LIST

Open a terminal and enter the following command to add the ROS package source:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

This command tells Ubuntu to fetch ROS packages from the official ROS repository whenever we install or update software.

2.2 ADD ROS GPG KEYS

Ubuntu requires a GPG key to verify the authenticity of packages before installing them.

```
sudo apt install curl  # Ensure curl is installed
curl -sSL 'http://repo.ros2.org/repos.key' | sudo apt-key add -
```

This command downloads and adds the required security key for ROS.

STEP 3: INSTALL ROS MELODIC

ROS (Melodic) is required for handling robotic applications, including LiDAR integration and sensor management

Steps to Install ROS Melodic:

3.1 Update the Package List: sudo apt update

This ensures that your package manager retrieves the latest versions of all dependencies.

3.2 Install ROS Melodic Desktop Full:

sudo apt install ros-melodic-desktop-full

This installs the complete ROS package, including essential libraries, tools, and simulation environments.

Output: If installed successfully, ROS should be available without errors.

STEP 4: INITIALIZE rosdep (DEPENDENCY MANAGER)

The rosdep tool is essential for resolving dependencies for ROS packages, ensuring that all required system libraries, Python modules, and ROS utilities are installed.

- Initialize rosdep: sudo rosdep init
- Update rosdep: rosdep update
  - Output: The dependency manager initializes successfully.

STEP 5: Set Up ROS Environment Variables

Setting up environment variables ensures that ROS commands like roscore, rosrun , and roslaunch work properly in every new terminal session.

Steps:

- Add the ROS environment setup to the .bashrc file:
  echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
- Apply changes to the current sessions: source ~/.bashrc
- Verify the setup by running: echo $ROS_PACKAGE_PATH

Output :The command should return /home/pi/catkin_ws/src:/opt/ros/melodic/share, confirming the workspace setup.

STEP 6: Run ROS Core to Verify

The roscore process manages communication between ROS nodes: roscore

Expected Output:

started core service [/rosout]

This confirms that ROS is running correctly.

STEP 7: Run a Sample ROS Node (Turtlesim)

To verify ROS functionality, run a sample node.

- Start the Turtlesim GUI: rosrun turtlesim turtlesim_node
- Control the turtle using keyboard command: rosrun turtlesim turtle_teleop_key

Output: A GUI opens showing a turtle that moves using arrow key inputs, confirming that ROS nodes and topics are working.

The below fig 4.1 shows the final completion of ros installation using GUI keys and it depicts the movement of turtle using the arrow keys present in the keyboard by making it visual approach on command prompt.

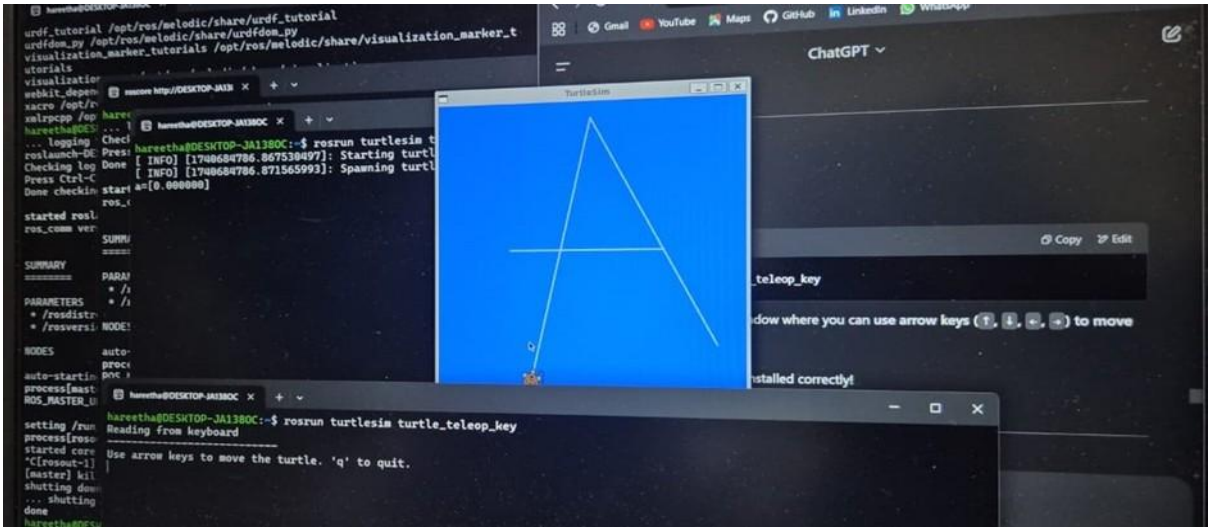

Fig 4.1 EXPECTED OUTPUT IN TURTLESIM

STEP 8: Check Installed ROS Packages

To list all installed ROS packages: rospack list

Output: The list should include packages such as turtlesim, rplidar_ros, usb_cam, and rospy.

STEP 9: Test LIDAR Detection

To check if RPLIDAR A1M8 is detected:

- Check available serial ports: ls /dev | grep ttyS*
  If found on /dev/ttyS0 , set permissions: sudo chmod 666 /dev/ttyS0

- Run LIDAR in ROS: roslaunch rplidar_ros rplidar.launch
- Visualize in RViz: rosrun rviz

Output: Running ls /dev/ttyS0 should return /dev/ttyS0. The LIDAR should start scanning when roslaunch rplidar_ros rplidar.launch is executed.

STEP 10: Debugging ROS Issues

If something isn't working, use ROS debugging tools.

- Run roswtf to diagnose issues: roswtf
- Restart ROS service: sudo systemctl restart ros
- Install missing packages: sudo apt install ros-melodic-<package-name>
- Check logs if roscore fails: cat ~/.ros/log/latest/roslaunch-*.log

Expected Output: If no issues are found, roswtf should return

4.2 SSH SERVER SETUP

STEP 1: Install SSH on Raspberry Pi

SSH allows remote access to the Raspberry Pi without needing a monitor, keyboard, or mouse. It enables commands to be executed from another computer, which is crucial for managing LIDAR and Radar remotely.

- Open a terminal on Raspberry Pi (or connect via a temporary monitor).
- Install the SSH server:

```
sudo apt update
sudo apt install openssh-server -y
```

- Enable and start the SSH service:

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

Output: Running sudo systemctl status ssh should return:

Active: active (running)

SSH is now enabled, allowing remote login.

STEP 2: Find the Raspberry Pi's IP Address

To connect remotely, first find the IP address, which is done using the Advanced IP Scanner (Windows)

- Download and install Advanced IP Scanner.
- Open the software and click Scan.
- Look for a device named raspberrypi.local
- Note the assigned IP address (e.g.,192.168.1.00)

STEP 3: Connecting using PuTTY (Windows)

- Open PuTTY (download from PuTTY.org if not installed).
- Enter Hostname or IP Address: raspberrypi.local OR 192.168.xx.xx
- Set Port to 22 and Click Open.
- When prompted, enter:
  - Username: pi
  - Password: raspberry (default, unless changed).
  - Alternatively, use SSH in a terminal: ssh pi@raspberrypi.local
  - If prompted, type yes and press Enter.

STEP 4: Enabling VNC Viewer (Remote Desktop Access)

- Enable VNC Server on Raspberrypi: sudo raspi-config
- Navigate to Interfacing Options → VNC → Enable.
- Install VNC Viewer on Windows/macOS
- Download VNC Viewer.
- Open VNC Viewer and type Raspberry Pi's address: raspberrypi.local
- Log in using Pi credentials.
- You should now see the Raspberry Pi's desktop interface.

OUTPUT:

Fig. 4.2 illustrates the Raspberry Pi's desktop interface, showcasing its graphical user interface (GUI) and available tools for user interaction. This interface provides access to the system's functionalities, allowing users to manage and control various processes running on the Raspberry Pi.

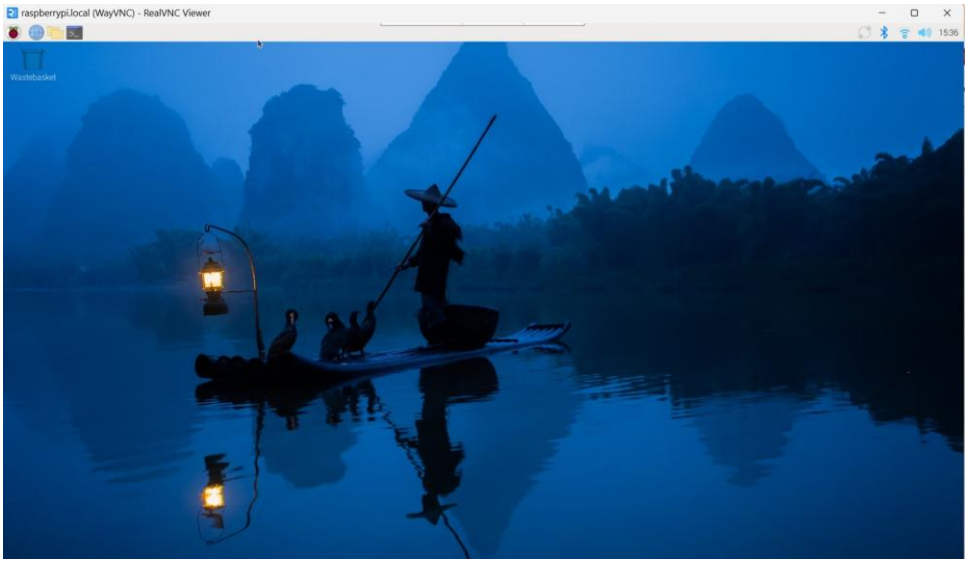

Fig 4.2 RASPBERRY PI'S DESKTOP INTERFACE

4.3 PYTHON INSTALLATION

STEP 1: Connect RPLIDAR A1M8 to Raspberry Pi

- A USB-to-Serial adapter is used to connect the Raspberry Pi to RPLIDAR A1M8.

STEP 2: Install Required Software Packages

- To communicate with the RPLIDAR A1M8 and visualize its data, we need to install some essential software packages on the Raspberry Pi.
- Update Package List
- Before installing new packages, always update the system's package list to ensure that you get the latest versions:

  sudo apt update

  - This updates the repository index, allowing the latest software versions to be installed.
  - Install Python and PIP

Since we will use Python for interfacing with the LIDAR, we need to ensure Python and PIP (Python package manager) are installed:

sudo apt install python3 python3-pip -y

- python3 → Installs Python 3 (if not already installed).
- python3-pip → Installs PIP, which is used to install Python packages.

4.4 PYTHON REQUIRED LIBRARIES

STEP 1: INSTALL REQUIRED PYTHON LIBRARIES
Done using the command: pip3 install numpy matplotlib rplidar

- numpy → Used for numerical calculations and data processing.
- matplotlib → Helps in plotting LIDAR scan data.
- rplidar → A dedicated Python library to communicate with and process data from the RPLIDAR.
- Verify installation by running: python3 -c "import numpy; import matplotlib; import rplidar; print('Libraries installed successfully!')"

If there are no errors, the setup is correct.

STEP 2: ENABLE SERIAL PORT

Since RPLIDAR communicates via a serial interface, we need to enable the serial port on the Raspberry Pi.

- Open Raspberry Pi Configuration Tool
- Run the following command to open the Raspberry Pi configuration menu: sudo raspi-config
- Navigate to Serial Port Settings
- Use arrow keys to navigate to Interfacing Options and press Enter.
- Scroll down to Serial Port and press Enter.
  It will ask:
  - Would you like a login shell to be accessible over serial?
    → Select NO (because we don't want the Pi's terminal to interfere with the LIDAR's serial communication).
  - Would you like the serial port hardware to be enabled?
    → Select YES (to enable communication with the RPLIDAR).
- Press Enter and then select Finish.

Reboot the Raspberry Pi if changes to take effect, reboot the Raspberry Pi,
sudo reboot

STEP 3: VERIFY THE SERIAL CONNECTION

- After rebooting, check if the RPLIDAR is detected: ls /dev | grep ttyS0
  If the output shows: ttyS0
  It means the Raspberry Pi successfully detected the RPLIDAR on /dev/ttyS0.
- Set Permissions

  Ensure that the Raspberry Pi has the correct permissions to access the LIDAR: sudo chmod 666 /dev/ttyS0

This grants read and write access to the serial port.

STEP 4: RUN TO CHECK RPLIDAR OUTPUT

- To ensure, LIDAR is working, run: python3 -m rplidar --port /dev/ttyUSB0.
- If the LIDAR is working properly, you should see raw scan data being printed in the terminal.

This project demonstrates the installation and configuration of essential software packages to integrate ROS Melodic and radar technologies on a Raspberry Pi. By setting up ROS, Python libraries, and radar tools, the system ensures optimal performance for sensor management, including LiDAR and radar. SSH and VNC enable remote access for efficient monitoring and control. The installation of radar packages supports advanced signal processing, providing a strong foundation for robotics applications. Verifications like running ROS nodes and visualizing radar outputs confirm the setup's effectiveness, offering a scalable solution for autonomous navigation and environmental mapping.

CHAPTER 5
RASPBERRY PI 4

The Raspberry Pi 4 Model B is a powerful single-board computer (SBC) used extensively in robotics, IoT, and automation. Powered by a 1.5 GHz quad-core ARM Cortex-A72 processor and available with 2GB, 4GB, or 8GB LPDDR4 RAM, it efficiently handles tasks such as AI-based applications, object detection, and real-time data processing. Its VideoCore VI GPU enables dual 4K display support via two micro-HDMI ports, making it ideal for graphical data visualization.

In this project, the Raspberry Pi 4 Model B as shown in Fig 5.1 functions as the central processing unit for a 360-degree object detection system using LiDAR and radar sensors. Its compact size, efficient thermal management, and processing power make it a cost-effective and reliable solution for sensor-based real-world applications.

Fig 5.1 RASPBERRY PI 4 MODEL B [5]

5.1 HARDWARE FEATURES

The Raspberry Pi 4 features a quad-core 64-bit ARM Cortex-A72 CPU (1.5GHz) for high-performance multitasking and computation. It offers RAM options of 1GB, 2GB, or 4GB LPDDR4 for efficient memory management. With H.265 (4Kp60) and H.264 (1080p60) decode support, it enables smooth 4K streaming and full-HD playback. The VideoCore VI GPU ensures advanced 3D rendering for gaming and multimedia. Additionally, dual HDMI outputs (4Kp60) allow for seamless multi-monitor setups, enhancing productivity.

5.2 PERIPHERALS

The Raspberry Pi 4 Model B features 28 GPIO pins on a 40-pin header as shown in fig 5.2 , enabling flexible interfacing with electronic components. These pins support digital I/O for controlling LEDs and relays, as well as alternate functions like I²C, SPI, UART, and PWM for communication with sensors and actuators. Operating at a 3.3V logic level, they require caution when interfacing with different voltages. The BCM2711 processor enhances GPIO capabilities with additional I²C, UART, and SPI peripherals, improving hardware integration, efficiency, and expandability. GPIO multiplexing allows pins to switch between functions, ensuring seamless connectivity while maintaining backward compatibility with earlier Raspberry Pi models.

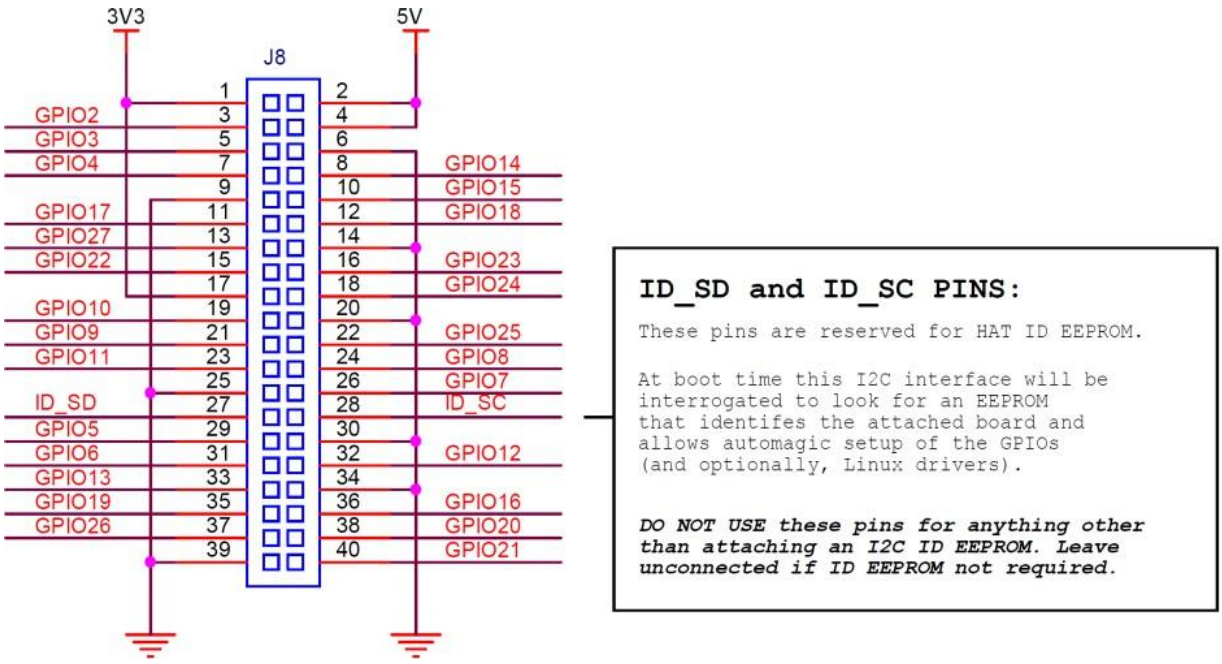

Fig 5.2 GPIO CONNECTOR PINOUT [8]

5.3 RASPBERRY PI IMAGER INSTALLATION

Objective: The purpose of this section is to explain the step-by-step process of preparing a Raspberry Pi 4 Model B by installing the Raspberry Pi OS on a microSD card using the Raspberry Pi Imager.

Step 1: Downloading and Installing Raspberry Pi Imager

- Navigate to the Official Website: Visit the official Raspberry Pi website to download the Raspberry Pi Imager software. The software is available for

multiple operating systems, including Windows, macOS, and Ubuntu. The download link is: https://www.raspberrypi.com/software/.

- Download the Software: Click on the download button corresponding to your computer's operating system. Once the download is complete, locate the installer file (e.g., rpi-imager-exe on Windows).
- Install the Raspberry Pi Imager: Double-click the downloaded file to start the installation process. Follow the on-screen instructions to complete the setup. After successful installation, launch the Raspberry Pi Imager application.

Step 2: Preparing the microSD Card

- Insert the microSD Card: Use a card reader to connect a 16GB microSD card to your computer. Ensure that the card is securely inserted and detected by the system.
- Formatting the Card (Optional): If the card contains previous data, it is recommended to format it using a standard formatting tool (e.g., SD Card Formatter) to ensure compatibility.

Step 3: Configuring the Raspberry Pi Imager

- Open Raspberry Pi Imager: Launch the Raspberry Pi Imager application from your computer.
- Select the Operating System (OS):
  - Click the "Choose OS" button.
  - From the list of available options, select Raspberry Pi OS (64-bit). This operating system is optimized for Raspberry Pi and provides the necessary environment for your project.
- Select the Storage Device:
  - Click the "Choose Storage" button.
  - From the list of connected devices, select the microSD card that you inserted earlier. Ensure the correct card is chosen to avoid overwriting other storage devices.

Step 4: Writing the Image to the microSD Card

- Start the Writing Process: After selecting the OS and storage, click the "Write" button. A confirmation dialog will appear, warning that the data on the microSD card will be erased. Confirm to proceed.

- Wait for the Process to Complete: The Raspberry Pi Imager will now download the selected operating system image and flash it onto the microSD card. This process might take a few minutes, depending on your internet speed and system performance.
- Verification: Once the writing process is complete, the software will verify the integrity of the flashed image to ensure there are no errors.
- Eject the microSD Card: After the verification is successful, safely eject the microSD card from the card reader.

Step 5: Booting the Raspberry Pi

- Insert the microSD Card: Place the prepared microSD card into the designated slot on the Raspberry Pi 4 Model B.
- Power On the Raspberry Pi: Connect a compatible power supply (5V, 3A USB-C adapter) to the Raspberry Pi and turn it on.
- First Boot: On powering up, the Raspberry Pi will boot into the operating system that was flashed onto the microSD card. If successful, the default Raspberry Pi desktop or command-line interface will appear.

Output: Successful Boot

- The below given fig 5.3 describes about the completion of the Raspberry pi imager installation. The successful completion of this process ensures that the Raspberry Pi is properly set up and ready for further configuration.

Fig 5.3 RASPBERRY PI IMAGER

The Raspberry Pi 4 Model B stands out as a robust and versatile platform, making it an ideal choice for integrating lidar and radar technologies for 360-degree object detection and tracking systems. Throughout this exploration, we have examined its hardware specifications, capabilities, and setup processes, all of which showcase its adaptability and performance in high-demand applications. Key takeaways from the analysis include:

1. GPIO Interface and Peripheral Support: The GPIO pins provide extensive connectivity options and support multiple protocols, such as I²C, UART, and SPI. This flexibility, along with multiplexing capabilities, allows efficient communication with external sensors and peripherals, critical for data processing in object detection systems.

2. Enhanced Features: Advanced interfaces such as the Display Parallel Interface (DPI), SD/SDIO support, camera and display connectors, and USB ports contribute to seamless integration and expanded functionality. These features ensure compatibility with a wide range of devices and accessories, aiding complex project requirements.

3. Thermal Management and Reliability: The Pi4B's dynamic thermal management system ensures optimal performance across varied workloads while maintaining safe operating temperatures. Its capability to deliver "sprint performance" and scale efficiently supports high-intensity tasks in real-time detection and tracking.

4. Ease of Setup with Raspberry Pi Imager: The detailed setup process, including the installation of Raspberry Pi OS via Raspberry Pi Imager, highlights the simplicity and user-friendliness of the platform. This ensures quick deployment and readiness for development, even for novice users.

5. Long-Term Availability and Support: Raspberry Pi guarantees availability of the Pi4B until at least 2031, coupled with comprehensive documentation and community forums. These assurances offer stability and ample resources for long-term project planning and execution.

CHAPTER 6
RPLIDAR A1M8

Fig 6.1 depicts the RPLIDAR A1M8 which is a compact and cost-effective 360-degree laser range scanner developed by Slamtec. It is widely used in applications such as robotics, mapping, and navigation due to its ability to generate high-resolution 2D point cloud data of the surrounding environment. The device operates on a laser triangulation measurement principle and offers reliable performance with a scanning frequency of up to 10 Hz. Its lightweight design and compatibility with various platforms make it an ideal choice for projects requiring precise environmental scanning and spatial awareness.

Fig 6.1 RPLIDAR A1M8 [6]

6.1 LIDAR IN 360 DEGREE SURVEILLANCE SYSTEM

The RPLIDAR A1M8 serves as a crucial component in a 360-degree surveillance system by providing continuous, accurate, and comprehensive environmental scanning. This lidar operates by emitting laser pulses and analyzing their reflections to measure distances to surrounding objects. Using its 360-degree rotational mechanism, it captures a wide field of view and generates detailed 2D point cloud data representing the spatial layout of the environment.

In the context of a 360-degree surveillance system, the RPLIDAR A1M8 is instrumental in several ways

I. Real-Time Environmental Monitoring: The device continuously scans the surrounding area and updates the point cloud data in real time. This ensures that the system remains aware of any changes in the environment, such as the movement of objects or the appearance of new obstacles.

II. Obstacle Detection and Avoidance: The high-resolution data produced by the lidar enables the system to detect obstacles with precision. This is particularly useful for applications involving autonomous robots or drones, where obstacle avoidance is critical to smooth operation.

III. Mapping and Localization: The RPLIDAR A1M8 can be used to create detailed maps of the environment. This mapping capability is valuable in static surveillance setups to monitor predefined areas or in dynamic environments where localization and path planning are required.

IV. Enhanced Security Monitoring: By integrating the lidar with advanced algorithms, the surveillance system can identify unusual or suspicious activities. For instance, it can detect unauthorized intrusions or the presence of unfamiliar objects, triggering alarms or responses as necessary.

V. Cost-Effective and Compact Solution: The compact size and affordability of the RPLIDAR A1M8 make it suitable for a variety of applications, ranging from indoor surveillance to outdoor perimeter monitoring. Its versatility and ease of integration with existing systems add to its appeal.

Overall, the RPLIDAR A1M8 enhances the functionality, accuracy, and reliability of a 360-degree surveillance system, making it an indispensable tool for ensuring safety and monitoring in diverse settings.

6.2 WORKING SCHEMATIC

The RPLIDAR A1M8 shown in Fig 6.2 is a 360-degree 2D laser scanner that utilizes Time-of-Flight (TOF) and triangulation principles for accurate distance measurement. It consists of a rotating head with an infrared laser emitter and a light-sensitive receiver. When the laser beam is emitted, it travels outward, hits an object, and reflects back to the receiver. The system calculates the distance based on the time delay and angle of return of the reflected light. A built-in motor enables continuous rotation at speeds of 5–10 Hz (300–600 RPM), allowing the scanner to capture real-time spatial data in all directions. The collected data points form a 2D map of the surroundings, making it useful for precise environment mapping.

Due to its high precision and real-time scanning capabilities, the RPLIDAR A1M8 is widely used in Simultaneous Localization and Mapping (SLAM) applications. It is commonly integrated into robotics, autonomous navigation, and obstacle detection systems, where it helps in path planning and real-time environment perception. The scanner communicates via a serial interface, making it easy to integrate with microcontrollers, embedded systems, and computing platforms like Raspberry Pi and Arduino.

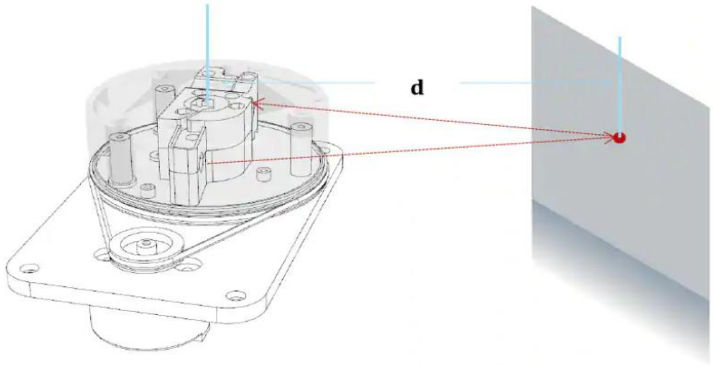

Fig 6.2 RPLIDAR A1M8 WORKING SCHEMATIC [6]

6.3 TECHNICAL SPECIFICATIONS

Table 6.1 TECHNICAL SPECIFICATIONS OF RPLIDAR A1M8

Specification	Details
Scanning Range	360°
Distance Range	0.15 m to 12 m
Sample Frequency	4000 Hz to 8000 Hz
Scan Rate	5.5 Hz (typical), up to 10 Hz
Angular Resolution	≤1°
Ranging Accuracy	1% (≤3 m), 2% (3-5 m), 2.5% (5-12 m)
Communication Interface	UART @ 115200 bps
Power Supply	5V
Power Consumption	0.5 W
Working Current	100 mA
Working Temperature	0°C to 40°C
Laser Safety	Class 1 Standards
Dimensions	96.8 mm x 70.3 mm x 55 mm

The table 6.1 describes the technical specification of RPLIDAR A1M8 which can be used for further analysis and interfacing purposes.

6.4 INTEGRATION OF LIDAR USING RASPBERRY PI

6.4.1 TEST FOR LIDAR DETECTION

To integrate the RPLIDAR A1M8 into a 360-degree surveillance system using ROS (Robot Operating System), the following steps are essential. These steps outline how to detect the device, set permissions, initiate the lidar operation, and visualize its data.

I. Check for Lidar Detection on the Serial Port

Before the system can process data from the RPLIDAR A1M8, it must confirm that the lidar device is connected and detected on the serial port. This can be verified using the following command in command prompt: ls /dev | grep ttyUSB

This command lists all the devices connected to serial ports and filters results to show entries related to `ttyUSB`. If the lidar is correctly detected, it will be listed under a specific port (commonly `/dev/ttyUSB0`).

II. Set Permissions for the Detected Serial Port

Once the lidar is detected (e.g., `/dev/ttyUSB0`), appropriate permissions need to be assigned to allow ROS to access the port. This can be achieved by running: sudo chmod 666 /dev/ttyS0

The `chmod` command modifies the access permissions, enabling read and write operations on the specified device.

III. Run the RPLIDAR Node in ROS

With the lidar detected and permissions granted, the next step is to launch the RPLIDAR node in ROS. The following command initiates the node: roslaunch rplidar_ros rplidar.launch

This command starts the `rplidar_ros` package, which acts as the driver to interface with the RPLIDAR A1M8. The lidar begins scanning and collecting environment data, ready to be utilized by other ROS nodes.

IV. Visualize Data Using RViz

To visualize the scanning data produced by the RPLIDAR A1M8, the RViz (ROS Visualization) tool is used.

Execute the following command to open RViz: rosrun rviz rviz

RViz provides a graphical interface where the lidar’s output (a 2D point cloud) can be observed in real time. This visualization helps validate the lidar’s functionality and enables monitoring of the surrounding environment.

V. Verify Lidar Operation

After running `ls /dev/ttyUSB*`, the expected output should display the device connected on `/dev/ttyUSB0`, confirming that the system has detected the lidar: /dev/ttyS0

Running the `roslaunch` command should successfully initiate the scanning process, with the lidar rotating and collecting data. Any errors at this stage indicate a configuration issue that needs troubleshooting.

By following these steps, the RPLIDAR A1M8 can be seamlessly integrated into the ROS ecosystem, enabling real-time scanning and visualization. This process forms the foundation for building sophisticated 360-degree surveillance systems or other robotics applications.

6.4.2 INSTALLING AND CONFIGURING RPLIDAR A1M8

I. Pin Configuration for Microcontroller and Raspberry Pi (RPi)

The document outlines the pin connections between a microcontroller and the Raspberry Pi (RPi) for serial communication and motor control. The pin configuration is as follows:

- Pin 1 (GND) → Pin 6 (RPi): Ground connection for common reference.
- RX (Receive) → Pin 8 (RPi): Receives data from the microcontroller.
- TX (Transmit) → Pin 10 (RPi): Transmits data to the microcontroller.
- V5.0 (Power) → Pin 2 (RPi): Supplies 5V power to the connected device.
- GND → Pin 9 (RPi): Additional ground connection.
- VMOTO → Pin 4 (RPi): Provides motor voltage supply.

II. Enabling and Configuring the Serial Port

- To establish communication between the Raspberry Pi and the microcontroller, the serial port must be enabled:
  Run the command: sudo raspi-config
- This opens the Raspberry Pi configuration menu, where serial communication can be enabled. Verify if the serial connection is established using: ls /dev/ttyS0
- This checks if the serial port is available for communication.

III. GPIO17 Activation for Motor Control

GPIO17 is used to control the motor. The following steps enable and configure it

- Install GPIO Driver
  If encountering an "invalid argument" error when accessing GPIO, a new driver may be required. Install it using: sudo apt install gpiod

- Check GPIO Pin Status
  Use the following command to display all GPIO pin information

IV. Configuring GPIO17

- Set GPIO17 as an input pin (inactive state by default).
- Convert GPIO17 to an output pin using the following command:

```
gpioset gpiochip0 17=1
```

This enables the motor by setting GPIO17 to a high state (1).

This configuration allows the Raspberry Pi to communicate with a microcontroller via serial communication while controlling a motor using GPIO17. The instructions ensure proper setup, debugging, and activation of the required components.

6.4.3 ALGORITHM FOR VISUALIZING LIDAR DATA IN A POLAR PLOT

I. Initialize LIDAR and Plot

- Import the required libraries: matplotlib for plotting, RPLidar for lidar communication, and numpy for numerical calculations.
- Define the LIDAR's connection port (/dev/ttyS0) and initialize the RPLidar object.
- Configure the plot for polar coordinates using matplotlib.

II. Define the process_scan Function

- Accept a single scan of LIDAR data as input.
- Initialize empty lists to store angles (in radians) and distances (in millimeters).
- Iterate through the scan data:
  - Extract the angle and distance for each detected point.
  - Convert the angle to radians using numpy.radians() and add it to the angles list.Add the distance value to the distances list.
- Update the polar plot:
  - Clear previous plot data using ax.clear().
  - Set the radial limit to a maximum of 4000 millimeters (4 meters).
  - Plot the detected points as blue scatter dots with specific size and transparency.
  - Render the updated plot using plt.draw().

III. Start Continuous Scanning and Visualization

- Print a message indicating that the scanning process has started.
- Enter a loop that continuously retrieves scans from the LIDAR using lidar.iter_scans().
- For each scan, pass the data to the process_scan function to update the polar plot in real time.

IV. Handle Interruption
- Detect a keyboard interruption (Ctrl+C) to allow the user to stop the scanning process gracefully.
- Catch any exceptions during execution and print error details.

V. Shutdown and Cleanup
- Once the scanning process is stopped (or an exception occurs):
  - Stop the LIDAR operation using lidar.stop().
  - Disconnect the LIDAR device using lidar.disconnect().
  - Disable the interactive plotting mode (plt.ioff()).
  - Display the final plot using plt.show().

This algorithm enables the real-time visualization of RPLIDAR A1M8 data in a polar plot, showcasing detected obstacles in a 360-degree field of view.

6.4.4 PYTHON CODE LIBRARIES USED EXPLANATION

I. Matplotlib

- Purpose: Matplotlib is a popular Python library used for creating static, interactive, and animated visualizations.
- Usage: In this code, Matplotlib is used to generate a polar plot that displays the LIDAR's scanned data.
- Functionality:
  - plt.ion(): Enables interactive mode for real-time updates.
  - fig, ax= plt.subplots(subplot_kw={'projection':'polar'}): Configures the plot with polar coordinates.

II. RPLidar
- Purpose: The RPLidar Python library allows communication with RPLIDAR sensors, enabling data acquisition and processing.
- Usage: This library is used to connect to the RPLIDAR device, retrieve scan data, and manage the device's operations.
- Functionality:
  - RPLidar(PORT_NAME): Initializes the lidar using the specified serial port.
  - lidar.iter_scans(): Retrieves data points from successive scans.

- lidar.stop() and lidar.disconnect(): Gracefully shuts down the LIDAR sensor.

III. Numpy

- Purpose: Numpy is a library for numerical computations and efficient array manipulation.
- Usage: In this code, Numpy is used to convert angles from degrees to radians for plotting.
- Functionality:
  - np.radians(angle): Converts the angular data (in degrees) into radians, suitable for polar plot representation.

6.4.5 FLOWCHART

The workflow depicted in Fig. 6.3 provides a comprehensive overview of the operational process of a LiDAR-based scanning system, detailing each step from initialization to data visualization and termination. The process begins with the initialization of the LiDAR sensor, ensuring that it is properly powered on and ready for communication. Establishing a stable connection between the LiDAR sensor and the control system is crucial, as it allows seamless data transmission for accurate real-time processing. Once the connection is successfully established, the system proceeds to configure a polar plot, which serves as the primary medium for visualizing the collected data. This polar plot allows for an intuitive graphical representation of distance measurements, providing users with a clear understanding of the scanned environment.

After setting up the visualization interface, the scanning function is defined, specifying how the system will acquire, process, and display data points. This function determines the scanning parameters, including the frequency of data acquisition, the range of angles to be covered, and the method for processing raw sensor data. With these configurations in place, the system enters a continuous scanning loop, where it repeatedly collects LiDAR data at various angles. The collected angles are then converted into radians to ensure accurate plotting within the polar coordinate system. This conversion is essential for maintaining precise alignment between the scanned data points and their corresponding locations in the visualization interface. Throughout the process, the system dynamically updates the polar plot, enabling real-time observation of the scanned environment and ensuring that users can monitor changes as they occur.

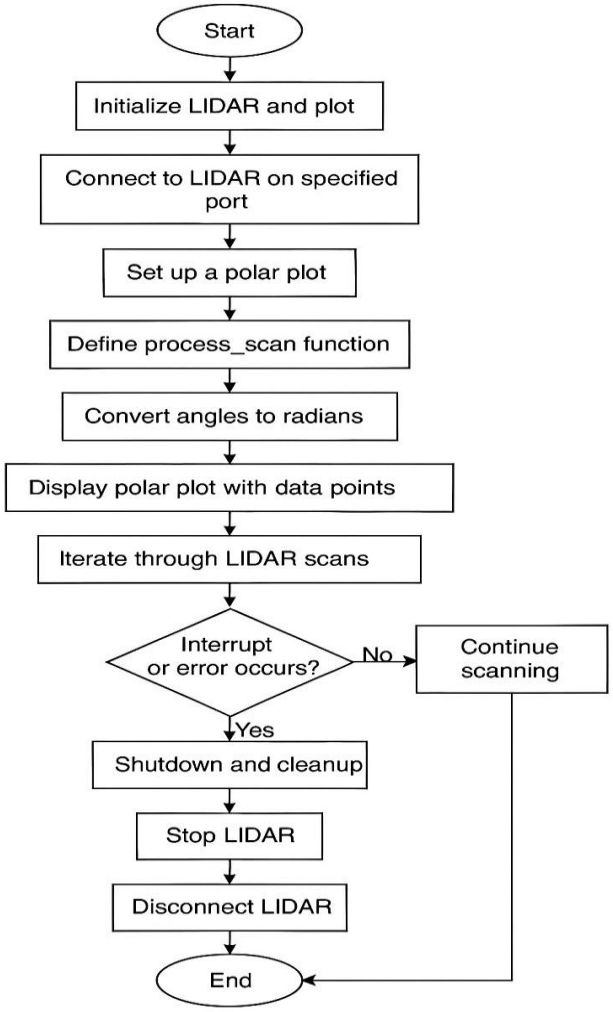

Fig 6.3 FLOWCHART FOR VISUALIZING LIDAR DISPLAY

6.4.6 LIDAR OUTPUT DISPLAY

Fig.6.4 and Fig 6.5 describe a polar plot visualization used for displaying LiDAR scan data in a circular coordinate system. The plot consists of concentric circles representing distance measurements and angular labels in degrees, enabling the graphical representation of detected objects. This visualization is essential for analyzing LiDAR data, as it helps interpret spatial information, detect obstacles, and monitor environmental changes in real time. Such a system is commonly used in robotics, autonomous navigation, and surveillance applications, ensuring precise object detection and mapping. The interface also includes zoom, pan, and save options, enhancing user interaction and analysis capabilities.

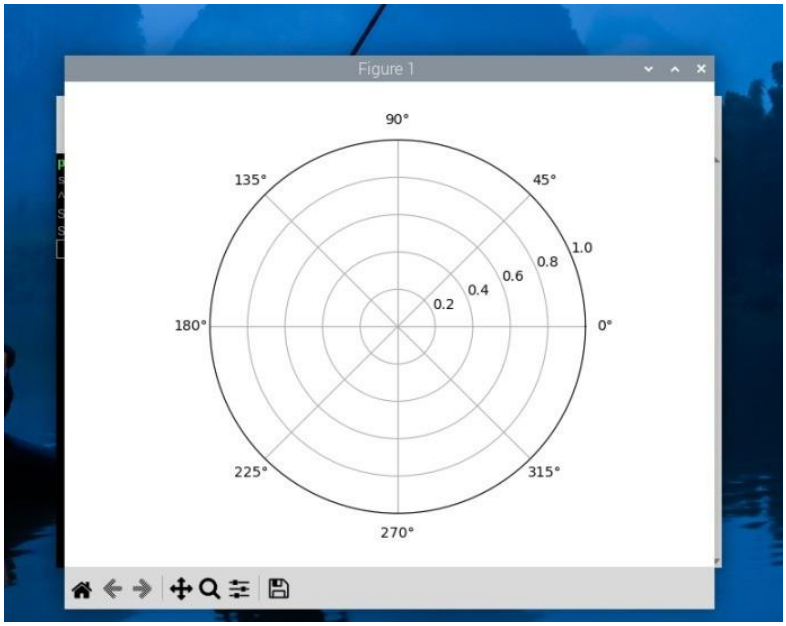

Fig 6.4 POLAR PLOT REPRESENTATION OF LIDAR

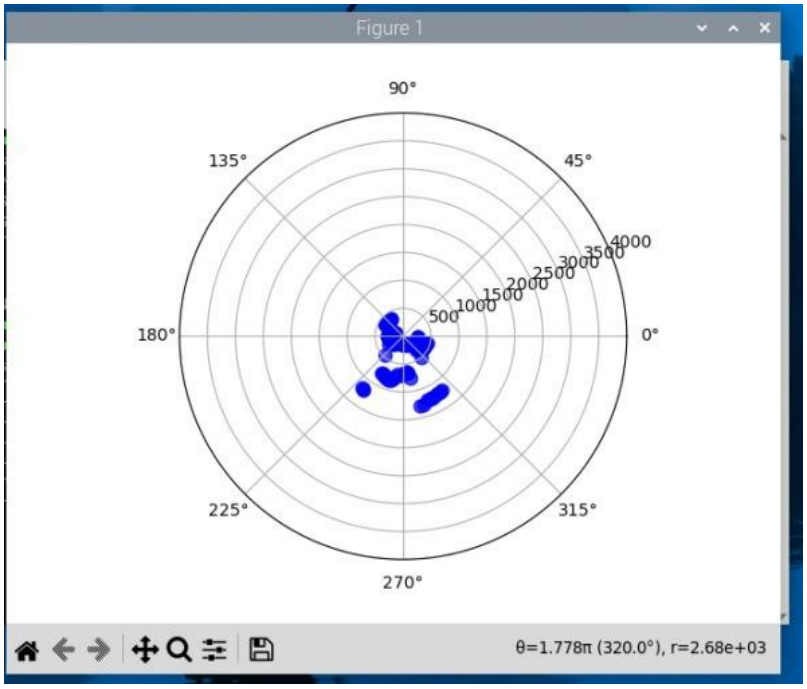

Fig 6.5 LIDAR DISPLAY WITH OUTPUT

6.4.6.1 Polar Plot Representation from RPLIDAR A1M8

The image represents a polar plot generated using data from an RPLIDAR A1M8 sensor. This visualization displays the scanned environment in a 360-degree field of view, with detected objects or obstacles represented as blue data points.

I. Polar Coordinate System

- The circular plot represents the LIDAR's scanning range in polar coordinates.
- The angles (θ) are labeled from 0° to 360°, indicating the direction of measurements.
- The radial distance (r) represents the measured distance of obstacles in millimeters.

II. Data Representation

- Each blue dot corresponds to a detected object within the LIDAR's scanning range.
- The closer a dot is to the center, the nearer the obstacle is to the LIDAR sensor.
- A cluster of points in the bottom-left section suggests obstacles positioned within a short range.

III. Bottom Display Values

- The θ (angle) and r (range) values displayed at the bottom indicate a specific detected point:
  - θ = 320.0° (angle of detection).
  - r = 2.68e+03 mm (2.68 meters) (distance from LIDAR).

This LIDAR output is essential for applications such as mapping, navigation, and obstacle detection in robotics and autonomous systems. The data generated can be further processed for tasks like Simultaneous Localization and Mapping (SLAM), object avoidance, and path planning.

The generated polar plots offer a clear representation of the scanned environment, highlighting detected objects and their positions relative to the sensor. This visualization is not only vital for debugging and validation but also serves as a foundation for advanced tasks like SLAM (Simultaneous Localization and Mapping) and autonomous navigation.In summary, the RPLIDAR A1M8 is an essential component for robotics and surveillance systems, offering precision, efficiency, and adaptability.

CHAPTER 7
RADAR RCWL-0516

A RADAR sensor (Radio Detection and Ranging) uses electromagnetic waves to detect and track objects, providing real-time data on their position, speed, and movement, even in poor visibility conditions like darkness or fog. Initially developed for military purposes, RADAR has evolved with advancements in artificial intelligence and signal processing, finding applications in aviation, automotive safety, weather forecasting, and modern security systems.

7.1 RADAR IN 360-DEGREE SURVEILLANCE SYSTEM

RADAR sensors are essential in 360-degree surveillance systems, offering continuous monitoring and object tracking across all directions. They use electromagnetic waves to operate effectively in conditions like darkness, fog, or rain, where optical sensors may fail. This makes them crucial for applications like perimeter security, traffic management, and autonomous navigation.

Key techniques include:

- Signal Modulation: Structures radio waves for better detection and range resolution.
- Doppler Frequency Shift: Measures object velocity via frequency changes in reflected waves.
- Digital Signal Processing (DSP): Enhances data by filtering noise, identifying multiple targets, and extracting parameters like speed and range.
- Phased-Array Antennas: Electronically steer beams for rapid and precise scanning.

These features ensure RADAR sensors provide reliable situational awareness in diverse environments.

7.2 RADAR IN MOTION DETECTION AND TRACKING

RADAR sensors excel in motion detection and tracking by emitting electromagnetic waves and analyzing reflected signals to detect movement, positions, and trajectories, even in challenging conditions like darkness or adverse weather. Their ability to distinguish stationary from moving objects makes them essential for surveillance, security, and automation.

Key Techniques in Motion Detection and Tracking

- Range-Doppler Processing: Measures distance using signal delay and velocity using Doppler shifts, enabling object tracking and trajectory prediction.
- CFAR (Constant False Alarm Rate): Reduces false alarms by adapting thresholds to filter out noise, maintaining accuracy in diverse environments.
- Kalman and Particle Filters: Enhance tracking precision by refining position estimates and handling predictable or erratic motion.

These advanced techniques ensure RADAR systems provide reliable and accurate motion detection for applications such as security and automation.

7.3 RCWL- 0516 RADAR SENSOR - OVERVIEW

The RCWL-0516 is a Doppler radar-based motion detection module designed for sensing human motion and other object movements. It operates on the principle of microwave Doppler effect, allowing it to detect movement with high sensitivity, even through non-metallic materials such as plastic, glass, and wood. This makes it a robust alternative to Passive Infrared (PIR) sensors, which are affected by temperature variations and line-of-sight restrictions.

This sensor finds applications in smart lighting systems, security alarm systems, and automation projects, where reliable motion detection is necessary.

→KEY FEATURES:

- Advanced Processing Chip: The RCWL-0516 module is powered by the RCWL-9196 chip, which handles signal transmission and processing with precision.
- Wide Operating Voltage: It supports a voltage range of 4.0V to 28.0V, making it adaptable to various power supply conditions.
- Penetrative Detection: Unlike PIR sensors, this module can detect motion through certain materials like glass, wood, and thin walls, enhancing its usability in concealed installations.
- Adjustable Parameters: The module allows customization of block time and detection distance, providing flexibility for specific applications.
- Output Power Supply: It offers a 3.3V output, suitable for driving microcontrollers or other connected devices.

→TECHNICAL SPECIFICATIONS:

- Operating Voltage: 4V to 28V
- Detection Range: 5 to 9 meters
- Output Voltage: 3.3V (regulated)
- Operating Frequency: Approximately 3.181 GHz
- Power Consumption: Typical transmitting power is 20mW, a maximum of 30mW
- Operating Temperature: -20°C to 80°C
- Storage Temperature: -40°C to 100°C

Fig. 7.1 illustrates the RCWL-0516 radar sensor module, a highly efficient component for motion detection. The figure offers two perspectives of the module: the top side, showcasing electronic components like the integrated circuit, resistors, and capacitors, and the bottom side, where connection pins (3V3, GND, OUT, VIN, and CDS) are clearly labeled.

The RCWL-0516's ability to sense motion through walls and non-metallic barriers makes it a preferred choice for applications such as security systems and automation projects. Its compact design and versatile functionality ensure reliable performance for various innovative uses.

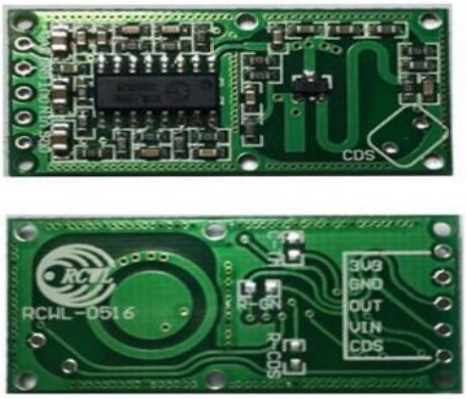

Fig 7.1 RCWL-0516 Radar Sensor [7]

→ WORKING PRINCIPLE OF RCWL-0516:

The RCWL-0516 sensor uses Doppler radar technology, based on the Doppler effect where wave frequency changes with relative motion between the wave source and the object. Here's how it functions:

- Microwave Emission: Generates microwaves at 3.181 GHz, emitting signals into the surroundings.
- Wave Reflection: Detects reflected signals that shift in frequency depending on the object's motion—frequency increases if approaching and decreases if moving away.
- Signal Mixing: Combines transmitted and reflected signals to produce a beat frequency, representing the difference between them.
- Signal Processing: The RCWL-9196 chip processes the Doppler signal, amplifies it, and determines if motion has occurred.
- Triggering Output: Activates the output pin (3.3V) when motion is detected, suitable for triggering connected devices like alarms or microcontrollers.

→ DETECTION CHARACTERISTICS OF RCWL-0516:

The RCWL-0516 radar sensor is designed for efficient motion detection and adaptability. Here are its key features:

- Sensing Face Orientation: The front side (component side) is the main sensing surface for optimal detection, while the rear side is less sensitive, reducing unnecessary detections.

- Detection Range: Default range is 7 meters, adjustable to 5 meters using a 1MΩ resistor on the R-GN pad.
- Blocking Time: Default blocking time is 2 seconds, extendable with an external capacitor on the C-TM pad for longer detection delays.
- Interference Management: Ensure 1 cm clearance from metallic surfaces and keep sensors at least 1 meter apart to avoid mutual interference.

7.4 ALGORITHM FOR RADAR-MOTION DETECTION

The motion detection process using the RCWL-0516 radar sensor follows a structured approach to ensure real-time detection of movement. The algorithm for this system is outlined below:

- ➔ Initialize the System: Begin by setting up the Raspberry Pi and installing the necessary libraries, including RPi.GPIO for interfacing with the GPIO pins.
- ➔ Configure GPIO Pins: Define the GPIO pin connected to the sensor's OUT pin and set it up as an input.
- ➔ Start Monitoring: Enter a continuous loop to monitor the state of the sensor.
- ➔ Detect Motion: If the sensor detects movement (i.e., the GPIO pin goes HIGH), print "Motion Detected!" to indicate detection.
- ➔ Introduce Delay: A small delay is introduced between detections to manage the system's response time.
- ➔ Handle User Interruption: Allow the user to stop the program using a keyboard interrupt (Ctrl+C).
- ➔ Cleanup and Exit: Ensure that the GPIO pins are properly reset before terminating the program.

7.5 FLOWCHART

This flowchart (Fig 7.2) provides a good and systematic representation of a radar motion detection program using GPIO pins. It begins by initializing the libraries to be used and setting the sensor pin to input mode. The program goes into a loop to continuously monitor motion detection, printing "Motion Detected!" when there is movement. It also has a feature of graceful program termination, freeing the GPIO pins upon program termination.
The flowchart is of very significant importance in depicting the program logic underlying, particularly in cases such as security systems or automation works where motion detection is of prime importance. The linear flowchart makes it easy to implement and the ability to adjust to different real-life scenarios.

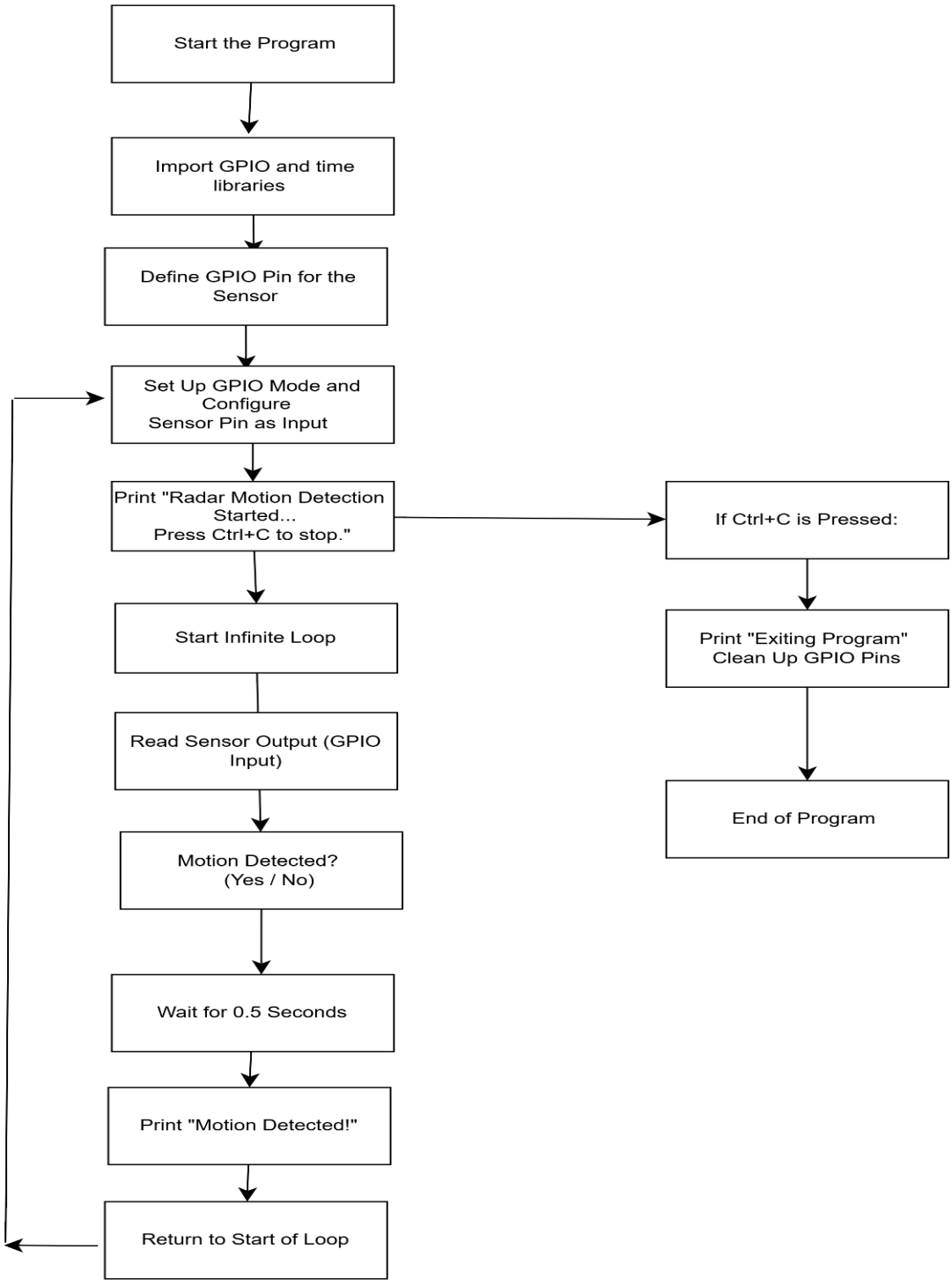

Fig 7.2 FLOWCHART

7.6 PYTHON CODE EXPLANATION

The provided Python script enables motion detection using the RCWL-0516 RADAR sensor with a Raspberry Pi. The code is structured to continuously monitor the sensor's output and identify movement in real time. Below is a detailed explanation of its key components and functionality:

→ Importing Required Libraries

The script begins by importing the necessary libraries:

```
import RPi.GPIO as GPIO
import time
```

- RPi.GPIO: This library allows the Raspberry Pi to interact with the General Purpose Input/Output (GPIO) pins, enabling communication with external sensors like the RCWL-0516.

- time: This module is used to introduce delays in the execution of the program, preventing excessive CPU usage and allowing for controlled polling of the sensor data.

→ Defining the GPIO Pin

```
RADAR_PIN = 17  (Connect OUT pin of RCWL-0516 to GPIO 17)
```

- The RADAR_PIN variable is assigned the GPIO pin number (17 in this case) to which the sensor’s output pin is connected. This helps the program monitor motion signals effectively.

→ GPIO Setup

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(RADAR_PIN, GPIO.IN)
```

- GPIO.setmode(GPIO.BCM): This command sets the Raspberry Pi's GPIO numbering scheme to BCM mode, which refers to the Broadcom chip-specific pin numbers.
- GPIO.setup(RADAR_PIN, GPIO.IN): Configures the defined pin (GPIO 17) as an input so that the Raspberry Pi can read signals from the RADAR sensor.

→ Initiating Motion Detection

```
print("Radar Motion Detection Started... Press Ctrl+C to stop.")
```

A simple message is printed to indicate that the motion detection system is running.

→ Detecting Motion Using an Infinite Loop

```
try:
  while True:
    if GPIO.input(RADAR_PIN):  # If motion is detected
      print("Motion Detected!")
    time.sleep(0.5)  # Adjust delay as needed
```

- The program runs an infinite loop (while True), continuously checking the sensor's output.
- GPIO.input(RADAR_PIN): Reads the state of the connected sensor pin. If the sensor detects motion, the OUT pin goes HIGH (1), and the program prints "Motion Detected!".
- time.sleep(0.5): Introduces a short delay (0.5 seconds) between readings to optimize CPU usage and improve response time.

→ Handling Program Termination Gracefully

```
except KeyboardInterrupt:
  print("\nExiting Program")
finally:
  GPIO.cleanup()
```

- The except KeyboardInterrupt block ensures that if the user presses Ctrl+C to stop the program, it prints "Exiting Program" instead of terminating abruptly.
- The finally block executes GPIO.cleanup(), resetting the GPIO pins to a safe state and preventing errors in future executions.

7.7 CHALLENGES AND LIMITATIONS

1. Environmental Interference: Adverse weather (rain, fog, snow) reduces signal strength and accuracy.
   - Solution: Advanced signal processing, noise filtering, and sensor fusion with LiDAR.
2. Multipath Reflections: Signal bouncing off surfaces causes ghost targets and localization errors.
   - Solution: Adaptive filtering, beamforming, and machine learning for accurate detection.
3. High Power Consumption: Long-range and high-resolution RADARs consume significant energy.
   - Solution: Solid-state RADARs and optimized power management systems like duty cycling.
4. Size and Cost Constraints: Traditional RADAR systems are bulky and expensive.
   - Solution: Miniaturized solid-state RADARs and cost-effective manufacturing techniques.

5. Overall Solutions: Integrating RADAR with other sensors, using AI for real-time processing, and adopting energy-efficient hardware designs ensures better performance and reliability in diverse applications.

7.8 RESULTS

Fig. 7.3 displays the radar output, illustrating detected objects and distance measurements in real time. The radar system processes reflected signals to identify obstacles, movement patterns, and environmental features. Data visualization highlights object positions and relative distances, aiding in situational awareness. This output is crucial for applications such as autonomous navigation and object tracking. The integration of radar enhances detection capabilities, especially in low-visibility conditions.

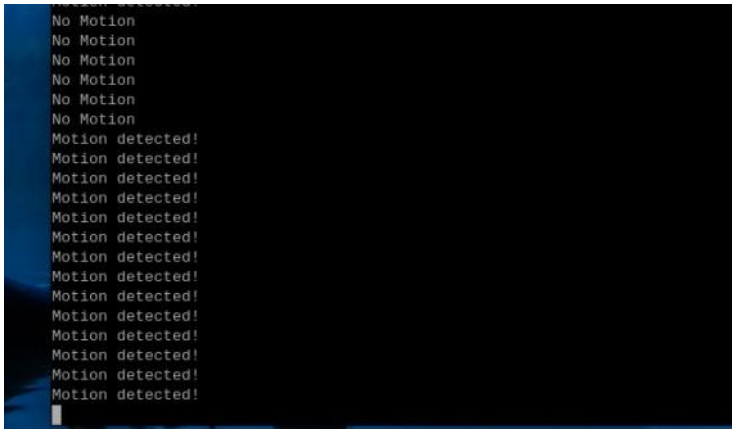

Fig 7.3 RADAR OUTPUT

The RCWL-0516 RADAR sensor is a compact yet powerful motion detection module that operates efficiently under various conditions. Unlike infrared sensors, it is immune to temperature variations and can detect movement through non-metallic obstacles, making it ideal for security, automation, and robotics applications. Despite its advantages, challenges like interference, false triggering, and lack of precise range measurement limit its standalone usability. However, integrating it with LiDAR and cameras enhances detection accuracy. Overall, the RCWL-0516 is a cost-effective and reliable RADAR module, demonstrating the potential of RADAR technology in modern sensing and automation systems.

CHAPTER 8

FUSION OF SENSORS

The integration of LiDAR (Light Detection and Ranging) and RADAR (Radio Detection and Ranging) is transforming real-time object detection, tracking, and spatial mapping across various domains such as autonomous vehicles, industrial automation, and security surveillance. LiDAR offers precise distance measurements and detailed 3D mapping, while RADAR excels in motion detection, even in low-visibility conditions. Combining these technologies enhances detection accuracy and situational awareness. This chapter explores the integration of RPLIDAR A1M8 and RCWL-0516 RADAR using Raspberry Pi 4, which processes real-time data and visualizes it through Tkinter. The system employs polar plots and heatmaps for intuitive spatial representation, while multithreading ensures non-blocking data acquisition, preventing interface lag.

The implementation focuses on seamless sensor interfacing, real-time data visualization, and computational efficiency. Using Python's Tkinter after() method, the GUI dynamically updates without excessive CPU usage. The system finds applications in autonomous navigation, industrial robotics, security monitoring, smart cities, and assistive technologies. By the end of this chapter, readers will understand how to interface these sensors, develop a responsive GUI, implement non-blocking data acquisition, and optimize real-time visualization using Matplotlib and NumPy. The following sections detail the system architecture, algorithm design, and step-by-step implementation of this integrated LiDAR-RADAR framework.

8.1 ALGORITHM FOR SENSOR DATA INTEGRATION

Step 1: Initialize Raspberry Pi and configure GPIO for RADAR sensor input.
Step 2: Establish a serial connection with RPLIDAR and initialize scanning.
Step 3: Set up a Tkinter GUI with a table, polar plot, and heatmap.
Step 4: Start a background thread to acquire real-time LiDAR and RADAR data.
Step 5: Process LiDAR scan data, converting polar coordinates into Cartesian for visualization.
Step 6: Read RADAR motion data and synchronize it with corresponding LiDAR distance readings.
Step 7: Update the Tkinter GUI periodically to refresh LiDAR plots and motion heatmaps.
Step 8: Continuously check for new data and refresh the display at defined intervals.
Step 9: If the system stops, safely disconnect LiDAR and clean up GPIO pins.

8.2 FLOWCHART OF SENSOR DATA INTEGRATION

Fig 8.1 depicts the flow of sensor data integration by obtaining the datas from LIDAR and RADAR sensors.

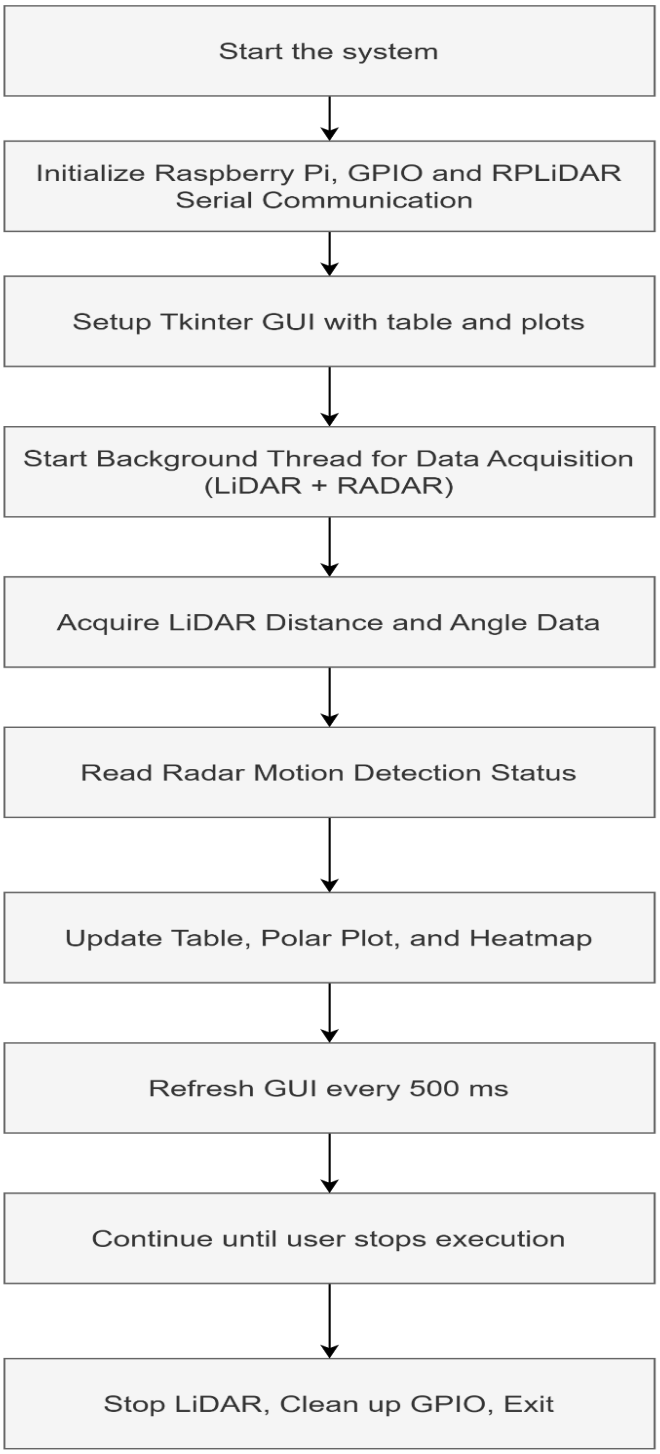

Fig 8.1: FLOWCHART

8.3 EXPLANATION FOR PYTHON CODE OF SENSOR DATA INTEGRATION

→ SENSOR INITIALIZATION

```
LIDAR_PORT = "/dev/ttyS"
lidar = RPLidar(LIDAR_PORT)
RADAR_PIN = 16
GPIO.setmode(GPIO.BCM)
GPIO.setup(RADAR_PIN, GPIO.IN)
```

- RPLIDAR A1M8 is initialized using /dev/ttyS as its communication port.
- RCWL-0516 RADAR sensor is connected to GPIO 16, using BCM mode for pin referencing.

→Tkinter GUI SETUP

```
root = tk.Tk()
root.title("Real-time LIDAR & RADAR Data")
root.geometry("1200x600")
```

- A Tkinter window is created with a title and a defined resolution of 1200x600 pixels.

→ Data Table for Real-Time Readings

```
frame = tk.Frame(root)
frame.pack(side=tk.LEFT, padx=10, pady=10)
columns = ("Distance (m)", "Angle(°)", "Motion")
tree = ttk.Treeview(frame, columns=columns, show="headings")
for col in columns:
    tree.heading(col, text=col)
    tree.column(col, width=100)
tree.pack()
```

- A Treeview table displays distance, angle, and motion detection values.
- The tree.heading() and tree.column() functions define table headers and column widths.

→ Matplotlib Visualization Setup

```
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
heatmap_fig, heatmap_ax = plt.subplots()
heatmap_canvas = FigureCanvasTkAgg(heatmap_fig, master=root)
heatmap_canvas.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
```

- Two visualization panels are created
  - Polar scatter plot for LiDAR data (ax).
  - Heatmap plot for object density visualization (heatmap_ax)

→ SENSOR DATA ACQUISITION (Running in Background Thread)

```
def get_sensor_data():
    """ Continuously fetch LIDAR & Radar data in background """
    global distances, angles, motion_detected
    for scan in lidar.iter_scans():
        distances, angles, motion_detected = [], [], []
        for (_, angle, distance) in scan:
            if distance > 0:
                distances.append(distance / 1000)  # Convert mm to meters
                angles.append(np.radians(angle))  # Convert degrees to radians
                motion_detected.append("Motion" if GPIO.input(RADAR_PIN) == 1 else
    "No Motion")
        time.sleep(0.1)  # Prevent excessive CPU usage
```

- LiDAR data is continuously fetched in a separate thread.
- Distance (converted to meters) and angle (converted to radians) are extracted.
- Motion detection is performed by reading the GPIO input from the RADAR sensor.
- A 0.1-second delay prevents excessive CPU usage.

→ REAL TIME DATA UPDATE IN GUI

```
def update_ui():
    """ Update Tkinter Table and Plots """
    tree.delete(*tree.get_children())  # Clear table
    if distances and angles:
        for i in range(len(distances)):
            tree.insert("", "end", values=(f"{distances[i]:.2f}", f"{np.degrees(angles[i]):.2f}",
    motion_detected[i]))
        ax.clear()
        ax.set_ylim(0, 6)
        colors = ['red' if m == "Motion" else 'blue' for m in motion_detected]
        ax.scatter(angles, distances, c=colors, alpha=0.75)
        canvas.draw_idle()
        heatmap_ax.clear()
        heatmap, _, _ = np.histogram2d(distances, angles, bins=(50, 50))
        heatmap_ax.imshow(heatmap.T, origin='lower', cmap='hot',
    interpolation='nearest')
        heatmap_canvas.draw_idle()
    root.after(500, update_ui)  # Refresh every 500ms
```

- Clears old data from the Tkinter table.
- Updates treeview table with the latest distance, angle, and motion readings.
- Polar scatter plot is updated, with:
- Red points indicating motion detected by RADAR.
- Blue points for static objects.
- Heatmap updates to visualize object density.

- Refreshes every 500ms using root.after(500, update_ui).

→ MULTITHREADING FOR NON BLOCKING EXECUTION

```
threading.Thread(target=get_sensor_data, daemon=True).start()
root.after(500, update_ui)
root.mainloop()
```

- Sensor data acquisition runs in a separate thread (daemon=True ensures the thread closes with the program).
- Tkinter event loop (mainloop()) starts, handling real-time updates.

→ CLEANUP AND SAFE SHUTDOWN

```
lidar.stop()
lidar.disconnect()
GPIO.cleanup()
```

- Stops LiDAR scanning to prevent hardware conflicts.
- Disconnects the LiDAR sensor properly.
- Resets GPIO settings to avoid errors in subsequent executions.

This chapter discussed the integration of LiDAR and radar for enhanced surveillance, highlighting their individual strengths and the benefits of sensor fusion. LiDAR provides high-resolution 3D mapping, while radar ensures reliable detection even in low-visibility conditions. By combining these technologies, the system achieves improved object tracking, environmental awareness, and real-time monitoring. The use of Raspberry Pi as the processing unit enhances cost-effectiveness and scalability. The concepts explored in this chapter lay the groundwork for implementing a robust surveillance system, with future advancements focusing on AI-driven analytics, cloud integration, and automated decision-making for improved efficiency and adaptability.

CHAPTER 9
CHALLENGES

Developing a robust and efficient system requires careful planning, execution, and troubleshooting at multiple stages. Throughout the development of our project, we encountered several hardware and software challenges that tested the reliability and functionality of our setup. These issues ranged from component failures and integration difficulties to software bugs and network instability. Addressing these challenges required a combination of technical expertise, analytical troubleshooting, and adaptive problem-solving.

This chapter provides a detailed account of the major hardware and software challenges faced during the project's development. The discussions highlight the technical hurdles, diagnostic steps, and corrective measures undertaken to ensure system reliability and performance.

9.1 INITIAL SETUP AND HARDWARE COMMUNICATION ISSUES

One of the first challenges encountered was setting up the Raspberry Pi 4 as the central processing unit without access to a dedicated monitor. The system had to be remotely accessed using a laptop, which introduced multiple communication issues.

Initially, a WiFi-based SSH connection was established, but it proved unreliable due to frequent disconnections and network instability. The inability to maintain a stable connection resulted in interruptions during file transfers, remote command execution, and debugging processes.

Further complications arose while configuring VNC Viewer for remote desktop access. Establishing a graphical interface required additional settings and modifications to ensure smooth operation. The dependency on a stable communication link created delays in the initial configuration and setup process.

9.2 RPLIDAR A1M8 SETUP AND DATA VISUALISATION

Integrating the RPLIDAR A1M8-R5 sensor with the Raspberry Pi 4 posed multiple challenges, particularly in terms of software compatibility and data processing. The initial plan involved using FrameGrabber for data visualization, but this solution was not compatible with the Raspberry Pi environment, necessitating an alternative approach. Another significant challenge was installing and configuring the RPLIDAR A1M8 libraries for Python. Several issues were encountered, including missing dependencies, incorrect baud rate configurations, and conflicts between different Python versions. Ensuring proper serial communication between the Raspberry Pi and the LiDAR sensor required deep troubleshooting.

Additionally, difficulties arose in interpreting and visualizing the LiDAR scan data. Understanding how to process distance and angle measurements accurately was a key challenge. The visualization tools initially used did not provide the necessary level of clarity, making it difficult to assess the sensor's output effectively.

9.3 ADAPTER MALFUNCTION AND RADAR SOLDERING ISSUES

One of the earliest and most critical challenges involved the USB to serial adapter of the RPLiDAR A1M8. This adapter plays a crucial role in establishing a communication link between the RPLiDAR and the Raspberry Pi. However, it experienced technical failures that severely disrupted system operations. Data transmission became unreliable, resulting in incomplete or corrupted data readings.

Resolution Process**:**

- Extensive troubleshooting was conducted, which included measuring continuity with a multimeter to identify broken connections.
- Precision soldering was applied to reinforce weak joints, ensuring consistent electrical contact.
- Improved quality control measures were implemented to prevent similar soldering issues in the future.

This experience underscored the importance of rigorous hardware testing and meticulous assembly processes to prevent hardware malfunctions that could compromise system functionality.

9.4 TRANSITION FROM RASPBERRY PI 3 TO RASPBERRY PI 4

The initial setup was designed around the Raspberry Pi 3, which provided a compact and efficient platform for handling computational tasks. Given the constraints of diagnosing and repairing the hardware failure, upgrading to a Raspberry Pi 4 was deemed a viable solution.

The Raspberry Pi 4 offers improved processing power, enhanced memory, and better hardware support. However, transitioning to the new hardware presented unforeseen complications:

- Power management differences: The Raspberry Pi 4 required higher power input, necessitating changes in the power supply design.
- Inconsistent ACT indicator behavior: The anticipated green glow from the ACT indicator was inconsistent, leading to initial confusion regarding system boot status.
- Software compatibility challenges: The upgraded hardware required new drivers and updated libraries, demanding significant modifications to the software stack.

Corrective Measures

- Updated the system firmware and boot configurations to ensure compatibility with the Raspberry Pi 4.
- Conducted extensive testing to validate software functionality and peripheral integration.
- Implemented power supply adjustments to match the new energy requirements of the Raspberry Pi 4.

Though the transition was challenging, the upgraded hardware significantly improved overall system performance and efficiency.

9.5 DIFFICULTIES WITH RASPBERRY PI MONITORING DISPLAY

Monitoring real-time operations on the Raspberry Pi was crucial for debugging and system diagnostics. However, the default display interface was cumbersome, making navigation time-consuming and inefficient. The difficulty in quickly accessing system metrics, logs, and performance indicators hampered the ability to respond promptly to issues.

Proposed Solutions

- Developed a custom monitoring interface that provided streamlined real-time data visualization.
- Integrated logging mechanisms to store system performance data for later analysis.
- Optimized the UI layout to improve usability and ease of access to critical information.

This challenge highlighted the importance of designing an intuitive monitoring interface that allows for seamless interaction with the hardware.

9.6 IP ADDRESS AND SSH TERMINAL DISCREPANCIES

During deployment, inconsistencies were observed between the Raspberry Pi's IP address and the SSH terminal access setup. This resulted in unexpected failures when attempting to establish remote connections, disrupting system control and maintenance operations.

Key Issues Identified

- Dynamic IP allocation by the router led to frequent address changes.
- SSH access configurations were not consistently synchronized with network changes.

Steps Taken to Resolve the Issue

- Configured a static IP address for the Raspberry Pi to ensure consistent remote access.
- Adjusted SSH settings to automatically reconnect upon network fluctuations.
- Implemented network monitoring tools to detect and log IP changes in real time.

9.7 PERFORMANCE ISSUES WITH Tkinter

Tkinter was selected as the GUI framework due to its simplicity and ease of integration. While it functioned well in isolated test scenarios, its performance deteriorated during real-time monitoring. The system exhibited noticeable latency, making the GUI sluggish and unresponsive.

- Introduced multithreading to separate GUI operations from data processing.
- Reduced unnecessary UI refresh rates to improve response times.
- Considered alternative lightweight GUI frameworks better suited for real-time applications.

Improving Tkinter's efficiency was crucial to ensuring a seamless user experience during system monitoring and control.

9.8 ENHANCING VISUALIZATION WITH Tkinter AND MATPLOTLIB

The initial visualization method relied on Matplotlib, which provided static plots of LiDAR and radar data. However, this approach lacked real-time interactivity, leading to the need for a GUI-based solution using Tkinter.

Integrating Tkinter with Matplotlib introduced several complications

- Synchronization Issues: Matplotlib's real-time plotting mechanisms conflicted with Tkinter's event-driven framework, resulting in execution delays and display errors.
- Recursion Errors: The system encountered repeated recursion errors when attempting to dynamically update the visualization, leading to crashes.
- Incorrect Data Display: The GUI initially displayed random test values instead of real sensor data, making it difficult to assess sensor performance.

Furthermore, Tkinter required additional programming to incorporate interactive buttons for starting scans and updating the display. Managing these functions while maintaining real-time data processing created additional challenges.

9.9 RPLiDAR A1M8 AND ITS OBJECT DETECTION LIMITATIONS

The RPLiDAR A1M8 is designed for 180-degree scanning, but it exhibited inconsistencies in detecting stationary versus moving objects. The sensor often misclassified static objects as moving entities, leading to errors in mapping and tracking.

Solutions Explored

- Fine-tuned software algorithms to differentiate between object types based on historical data analysis.
- Adjusted calibration settings to improve object detection accuracy.
- Conducted extensive testing under various environmental conditions to refine sensor behavior.

9.10 NETWORK SWITCHING FROM WiFi TO ETHERNET

Initially, the system relied on a WiFi connection, but frequent disruptions caused by interference and fluctuating signal strength led to unreliable operations. To mitigate these issues, an Ethernet-based connection was adopted.

Challenges Faced During the Transition

- Required modifications to hardware layout to accommodate wired connections.
- Introduced cable management complexities in a compact workspace.
- Needed to reconfigure network settings to prioritize Ethernet connectivity.

Despite these challenges, the switch to Ethernet significantly improved network stability and operational reliability.

CHAPTER 10

RESULT AND OBSERVATION

Integrating LiDAR and radar technology enhances real-time object detection and motion tracking by leveraging their complementary strengths. LiDAR provides high-resolution, short-range spatial mapping, while radar extends detection capabilities to longer distances with 360° coverage. This fusion creates a robust system for surveillance and autonomous navigation.

This section presents the integration results using three key visual components: a data table, a LiDAR-based heatmap, and a radar-based polar plot. The data table lists detected objects with attributes such as distance, angle, and motion status, aiding in numerical analysis. The LiDAR heatmap represents close-range object intensities using color variations, highlighting areas of strong reflections. Meanwhile, the radar polar plot maps object positions over a full 360° range, tracking motion at greater distances.

Key observations include object clustering, movement patterns, and sensor limitations. The LiDAR heat map indicates directional motion, while the radar plot detects objects beyond LiDAR's range, albeit with potential weak reflections. The analysis helps identify sensor blind spots and occlusions.

By combining LiDAR's precision with radar's wide coverage, this integration improves real-time situational awareness. Such a system is valuable for applications in autonomous vehicles, security surveillance, and obstacle detection in dynamic environments.

10.1 INTEGRATION RESULT

Fig. 10.1 illustrates the output of the integrated LiDAR and radar system, demonstrating how data from both sensors are collected, processed, and visualized in a unified framework. This integration plays a crucial role in enhancing environmental perception, as each sensor contributes unique capabilities. LiDAR provides high-resolution distance measurements by emitting laser pulses and calculating the time taken for their reflections to return. This enables precise mapping of static objects, terrain, and structural features. Meanwhile, radar complements LiDAR by detecting moving objects, identifying their speed, and performing well in adverse weather conditions such as fog, rain, or low-light environments where LiDAR performance may be limited.

The combined LiDAR-radar output creates a more robust situational awareness system, offering both high-resolution spatial data and dynamic object tracking. This synergy supports real-time monitoring and analysis, making it valuable for applications such as autonomous navigation, obstacle detection, and environmental sensing. The system can effectively differentiate between stationary structures and moving entities, improving decision-making in robotics, vehicular automation, and security surveillance. By leveraging

both LiDAR and radar technologies, the integrated output enhances the reliability and adaptability of sensor-based systems, ensuring accurate perception and improved operational efficiency in complex environments.

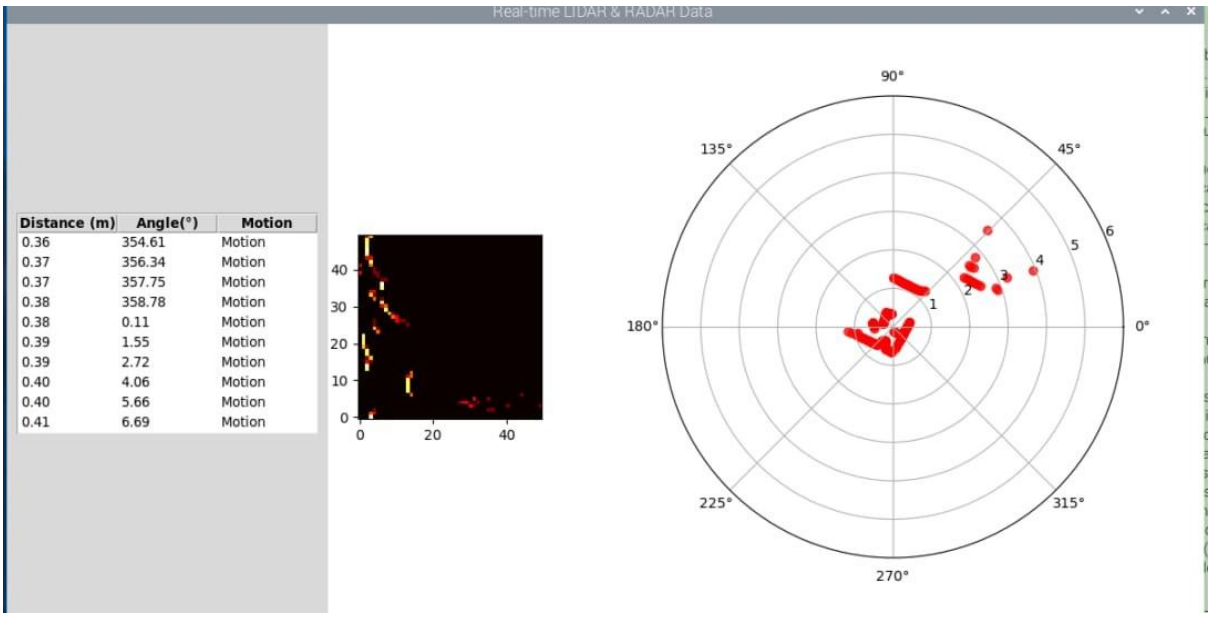

FIG 10.1 LIDAR-RADAR INTEGRATION OUTPUT

10.2 OBSERVATION

The provided image displays real-time LiDAR and radar data integration for surveillance and object detection. The output consists of three primary visual elements

1. A Data Table (Left Side) – Listing Object Properties.
2. A LiDAR-Based Heatmap (Middle) – Close-Range Object Detection.
3. A Radar-Based Polar Plot (Right) – 360° Object Tracking.

Let's analyze each component in depth

1. Object Detection Table (Left Side) – Raw Data Interpretation

The table provides numerical details of detected objects with three parameters:

- Distance (m): How far the object is from the sensor.
- Angle (°): The direction of the object in polar coordinates.
- Motion Status: Whether the object is moving or stationary (all detected as "Motion" in this case).

Observations from Table

- Objects are detected within 0.36m to 0.41m, indicating nearby motion within half a meter of the sensor.
- The angle ranges from 354.61° to 6.69°, meaning objects are mainly detected in the front-right region (near 0°/360°).
- Since all objects are labeled "Motion", the system effectively differentiates between static and dynamic objects, ensuring real-time motion tracking.

2. LiDAR-Based Heatmap (Middle) – Short-Range Object Representation
   - The heatmap represents object intensity using color variations (red/yellow spots).
   - Red and yellow indicate strong reflections, suggesting close-range moving objects.
   - Most detections appear on the left portion of the heatmap, meaning motion is concentrated in that area.
   - Sparse points on the right suggest:
     - Sensor occlusion (objects blocked by other objects).
     - Environmental factors (low reflectivity surfaces).

   Key Inferences
   - LiDAR captures short-range object details accurately.
   - Clustering in a specific region suggests directional movement rather than random motion.
   - Lack of detections in some areas could indicate blind spots or limited LiDAR range.

3. Radar-Based Polar Plot (Right) – 360° Motion Tracking
   - The polar plot provides a full-circle view (360°) of object positions.
   - Objects are plotted based on distance and angle, giving a spatial map of movement.
   - Detections are heavily concentrated in the 0°–45° sector, meaning most objects are moving within that field of view.
   - Some objects are spread out up to 5m from the center, meaning radar detects objects farther than LiDAR.

   Key Inferences
   - Radar effectively tracks objects in a wider area than LiDAR.
   - Objects clustered near the center mean movement close to the sensor, while outliers represent farther motion.
   - Sparse detections beyond 5m suggest:
     - Radar range limitations.
     - Potential weak reflections from distant objects.
     - Low radar cross-section (some objects reflect poorly).

The results and observations demonstrate the effectiveness of integrating LiDAR and radar for real-time surveillance, ensuring accurate object detection even in challenging environments. The system successfully provides 360-degree monitoring with improved tracking and reduced false detections. Sensor fusion enhances reliability by combining LiDAR's precision with radar's ability to operate in low-visibility conditions. The Raspberry Pi efficiently processes the data, enabling cost-effective and scalable deployment. These findings validate the system's potential for applications in security, smart cities, and industrial safety, with future improvements focusing on AI-driven analytics and cloud-based monitoring for enhanced performance.

CHAPTER 11

FUTURE TRENDS

This chapter explains the modifications that could be done to the project for future development using better advancements and techniques.

11.1 ADVANCEMENT IN SENSOR FUSION TECHNIQUES

Sensor fusion integrates data from multiple sources for accurate object detection. Future enhancements will refine tracking and object differentiation in complex environments.

- AI-Enhanced Data Fusion: AI and deep learning will improve the integration of RADAR, LiDAR, and other sensors, reducing background noise and enhancing detection.
- Advanced Filtering Techniques: Improved Kalman and Particle Filtering will enhance trajectory prediction and tracking efficiency.
- Adaptive Cross-Sensor Learning: Sensors will dynamically adjust based on environmental conditions, such as using thermal imaging in foggy settings.
- 3D Multi-Layer Mapping: Real-time 3D mapping will improve spatial awareness in autonomous navigation and security applications.

11.2 MINIATURIZATION AND COST REDUCTION

Technological advancements will make 360-degree surveillance systems smaller, more affordable, and more accessible.

- Compact LiDAR & Mini RADAR Modules: Transitioning to solid-state designs will reduce size and power consumption.
- MEMS-Based Cost Efficiency: Micro-Electro-Mechanical Systems (MEMS) will drive low-cost manufacturing of LiDAR and RADAR modules.
- Portable Surveillance Systems: Miniaturized sensors will enable integration into UAVs, robots, and wearable devices.
- Low-Power Processors: Energy-efficient AI chips will extend battery-powered surveillance system runtimes.

11.3 ENHANCED COMMUNICATION SYSTEMS

Reliable communication is essential for real-time monitoring and data security.

- 5G & Beyond: High-speed, low-latency networks will enhance real-time surveillance.
- Edge Computing: Local data processing on AI-enabled devices will reduce cloud dependency and improve response times.

- Secure Wireless Protocols: Advanced networks like LoRaWAN, Wi-Fi 6, and UWB will ensure seamless connectivity.
- Blockchain for Security: Decentralized encryption will protect surveillance data from unauthorized access.

11.4 AUTONOMOUS AND INTELLIGENT SYSTEMS

AI-driven automation will redefine surveillance by enabling self-learning systems capable of predictive analysis and independent decision-making.

- Predictive Analytics and Threat Detection: AI models trained on historical data will predict security threats before they occur, helping prevent incidents through preemptive alerts.
- Adaptive Object Recognition: Advanced neural networks will allow surveillance systems to distinguish between humans, animals, and vehicles, reducing false alarms and improving response accuracy.
- Autonomous Monitoring and Response Systems: Future surveillance systems will automatically trigger alerts, send drones to investigate disturbances, and notify security personnel without human intervention.
- Swarm Intelligence for Multi-Agent Surveillance: Multiple UAVs and ground-based robots will communicate in real time, coordinating their movements for efficient surveillance coverage.

11.5 ENVIRONMENTAL ADAPTABILITY

Future surveillance systems will feature enhanced adaptability to function effectively under diverse environmental conditions.

- Weather-Resistant Sensors: LiDAR and RADAR modules will be designed to withstand extreme temperatures, heavy rain, and high humidity without performance degradation.
- Low-Light and Night Vision Enhancement: Infrared and near-infrared imaging will be integrated with LiDAR to improve object detection in low-visibility conditions such as fog or total darkness.
- Electromagnetic Interference Mitigation: Future RADAR systems will employ advanced filtering techniques to minimize the effects of electromagnetic interference in industrial environments.
- Self-Calibrating Sensors: Surveillance systems will autonomously adjust their sensor parameters based on environmental feedback, ensuring continuous and optimal performance.

11.6 APPLICATIONS IN EMERGING INDUSTRIES

The advancements in 360-degree surveillance systems will extend their applications into various industries beyond security.

- Smart Cities: AI-powered surveillance will optimize traffic flow, detect unauthorized activity, and enhance urban safety through real-time monitoring of public spaces.
- Autonomous Vehicles: LiDAR and RADAR integration will enable self-driving cars to navigate safely by detecting obstacles, pedestrians, and road conditions in real time.
- Industrial Automation: In manufacturing and logistics, surveillance systems will ensure operational safety by monitoring worker activities, identifying machinery failures, and preventing hazardous incidents.
- Wildlife Conservation and Environmental Monitoring: AI-enhanced surveillance systems will track endangered species, monitor deforestation, and analyze climate change impacts in remote regions.

11.7 ETHICAL AND PRIVACY CONSIDERATIONS

As surveillance technology advances, ethical concerns regarding privacy and data security must be addressed to ensure responsible use.

- Data Protection and Encryption: Future systems will employ end-to-end encryption and secure storage solutions to safeguard sensitive surveillance data from cyber threats.
- Regulatory Compliance and Legal Frameworks: Governments and regulatory bodies will establish laws governing the ethical use of surveillance, ensuring transparency and accountability.
- Privacy-Preserving AI: AI models will be designed to anonymize personal data, reducing concerns related to mass surveillance and protecting individual identities.
- Ethical AI Deployment: AI-driven decision-making in surveillance will require ethical oversight to prevent bias, discrimination, and misuse of technology.

CHAPTER 12

CONCLUSION

The development of a 360-degree surveillance system utilizing Raspberry Pi 4, RPLIDAR A1M8, and RCWL-0516 successfully integrates advanced object detection, motion tracking, and environmental awareness. By leveraging LIDAR and radar technologies, the system achieves real-time monitoring with enhanced accuracy and reliability. The fusion of these technologies ensures uninterrupted surveillance, overcoming the limitations of traditional CCTV systems, which are often hindered by blind spots and poor lighting conditions. The implementation of sensor fusion techniques further improves object detection and tracking, making the system suitable for security, industrial monitoring, and autonomous navigation applications.

The hardware and software components work in synchronization to provide a scalable and efficient surveillance solution. Raspberry Pi 4 serves as the central processing unit, managing data collection and analysis, while software tools such as ROS OS, VNC Viewer, PuTTY, SSH, and Advanced IP Scanner enhance the system's remote accessibility and control. The combination of edge computing and wireless communication ensures real-time decision-making and monitoring, making the system adaptable to smart city applications, security enforcement, and automated industrial environments. Future improvements could include the integration of AI-driven analytics, cloud connectivity, and machine learning algorithms for enhanced object classification, automated threat detection, and predictive surveillance.

In summary, the proposed surveillance system offers a robust, cost-effective, and scalable solution to modern security challenges. Its ability to operate efficiently under various environmental conditions, combined with real-time data analysis and remote accessibility, makes it a promising advancement in surveillance technology. With future enhancements, this system has the potential to revolutionize security monitoring, industrial automation, and smart infrastructure development, ensuring a safer and more secure environment.

BIBLIOGRAPHY

[1] A. Biswas, S. Abedin, and M. A. Kabir, "Moving object detection using ultrasonic radar with proper distance, direction, and object shape analysis," *J. Inf. Syst. Eng. Bus. Intell.*, vol. 6, no. 2, pp. 99–111, Oct. 2020, doi: 10.20473/jisebi.6.2.99-111.

[2] H. Ullah, O. Zia, J. H. Kim, K. Han, and J. W. Lee, "Automatic 360° mono-stereo panorama generation using a cost-effective multi-camera system," *Sensors*, vol. 20, no. 11, Art. no. 3097, May 2020, doi: 10.3390/s20113097.

[3] S. Mahalingam and S. Subramoniam, "A robust single and multiple moving object detection, tracking, and classification," *Appl. Comput. Inf.*, Jan. 2018, doi: 10.1016/j.aci.2018.01.001.

[4] X. Li, J. Wang, and Y. Zhang, "Enhanced object detection in autonomous vehicles through LiDAR-camera sensor fusion," *World Electr. Veh. J.*, vol. 15, no. 7, Art. no. 297, Jul. 2024, doi: 10.3390/wevj15070297.

[5] Raspberry Pi Foundation, "Raspberry Pi 4 Model B image," [Online]. Available: https://cdn-reichelt.de/bilder/web/xxl_ws/A300/RASP_PI_4_B_01_ANW.png.

[6] Amazon, "Raspberry Pi 4 board image," [Online]. Available: https://images-na.ssl-images-amazon.com/images/I/61bAjpaoaTL.jpg.

[7] J. Desbonnet, "RCWL-0516 microwave motion sensor board image," [Online]. Available:https://raw.githubusercontent.com/jdesbonnet/RCWL-0516/refs/heads/master/images/RCWL-0516-board.jpg.

[8] Raspberry Pi Ltd., *Raspberry Pi 4 Model B Datasheet*, Revision 1.0, Jun. 2019. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf.

[9] TDK InvenSense, *A1M8 Datasheet*,2023. [Online]. Available: https://www.digikey.dk/htmldatasheets/production/3265529/0/0/1/a1m8.html.

[10] Guangxi WIT Electron Co., Ltd., *RCWL-0516 Microwave Proximity Sensor Module Datasheet*,2016.[Online]. Available:https://www.makerhero.com/img/files/download/RCWL0516-Datasheet.pdf?srsltid=AfmBOorcE3Iw14Bh1jb_fH38rS_RcWdQxSOn_2ZU5T7NPJy4PvrHVws1.

[11] S. Hameed, N. J. D. Rashid, and F. Shoaib, "Arduino-based radar system," *3C Tecnología. Glosas de innovación aplicadas a la pyme*, Special Issue on Recent Trends in Computer Science and Electronics, pp. 157–166, 2019.

[12] H. H. Raheem, A. I. Abdalla, and Z. N. A. Al-Rawi, "Design and implementation of ultrasonic radar system for distance measurements using Arduino," *Int. J. Eng. Technol.*, vol. 7, no. 4, pp. 3115–3118, 2018.

[13] A. A. Shah, G. Mustafa, Z. Ali, and T. Anees, "Video stitching with localized 360° model for intelligent car parking monitoring and assistance system," *Int. J. Comput. Sci. Netw. Secur. (IJCSNS)*, vol. 19, pp. 43, 2019.

[14] K. O. Demiralp, E. S. Kurşun-Çakmak, S. Bayrak, N. Akbulut, C. Atakan, and K. Orhan, "Trabecular structure designation using fractal analysis technique on panoramic radiographs of patients with bisphosphonate intake: A preliminary study," *Oral Radiol.*, vol. 35, pp. 23–28, 2019.

[15] A. Mehra, M. Mandal, P. Narang, and V. Chamola, "ReViewNet: A fast and resource optimized network for enabling safe autonomous driving in hazy weather conditions," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, pp. 4256–4266, 2020, doi: 10.1109/TITS.2020.3013099.

[16] X. Liu and O. Baiocchi, "A comparison of the definitions for smart sensors, smart objects and things in IoT," in *Proc. IEEE 7th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Vancouver, BC, Canada, Oct. 13–15, 2016.

[17] A. K. Shrivastava, A. Verma, and S. P. Singh, "Distance measurement of an object or obstacle by ultrasound sensors using P89C51RD2," *Int. J. Comput. Theory Eng.*, vol. 2, no. 1, 2010.

[18] G. Deshmukh, D. N. Nalavade, H. Salve, and D. A. Shaikh, "Ultrasonic radar for object detection, distance, and speed measurement," *Int. J. Res. Advent Technol. (IJRAT)*, vol. 5, Special Issue, 2017.

APPENDICES

A.1 PYTHON CODE FOR VISUALIZING LIDAR DISPLAY IN POLAR PLOT

```
import matplotlib.pyplot as plt
from rplidar import RPLidar
import numpy as np

PORT_NAME = '/dev/ttyS0'
lidar = RPLidar(PORT_NAME)
plt.ion()
fig, ax= plt.subplots(subplot_kw={'projection':'polar'})

def process_scan(scan):
  angles = []
  distances = []
  for (_, angle, distance) in scan:
    angles.append(np.radians(angle))
    distances.append(distance)

  ax.clear()
  ax.set_ylim(0, 4000)
  ax.scatter(angles,distances,s=60,c='blue',alpha=0.75)
  plt.draw()
try:
  print("starting continuous lidar scanning.. press ctrl+c to stop")
  for scan in lidar.iter_scans():
    process_scan(scan)
except KeyboardInterrupt:
print("\nStopping...")
except Exception as e:
print(f"\nError: {e}")

finally:
  print("Shutting down LIDAR...")
  lidar.stop()
  lidar.disconnect()
  plt.ioff()
  plt.show()
```

A.2 PYTHON CODE FOR PERFORMING MOTION DETECTION USING RCWL 0516 RADAR SENSOR

```
import RPi.GPIO as GPIO
import time

# Define the GPIO pin for the sensor output
RADAR_PIN = 17  # Connect OUT pin of RCWL-0516 to GPIO 17

# Setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(RADAR_PIN, GPIO.IN)
print("Radar Motion Detection Started... Press Ctrl+C to stop.")
try:
    while True:
        if GPIO.input(RADAR_PIN):  # If motion is detected
            print("Motion Detected!")
        time.sleep(0.5)  # Adjust delay as needed
except KeyboardInterrupt:
    print("\nExiting Program")
finally:
    GPIO.cleanup()
```

A.3 PYTHON CODE FOR SENSOR DATA INTEGRATION

```
import tkinter as tk
from tkinter import ttk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from rplidar import RPLidar
import RPi.GPIO as GPIO
import threading
import time
# LIDAR & Radar Setup
LIDAR_PORT = "/dev/ttyUSB0"
lidar = RPLidar(LIDAR_PORT)
RADAR_PIN = 17
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(RADAR_PIN, GPIO.IN)
# Initialize Tkinter
root = tk.Tk()
root.title("Real-time LIDAR & RADAR Data")
root.geometry("1200x600")
# Table for data
frame = tk.Frame(root)
frame.pack(side=tk.LEFT, padx=10, pady=10)
columns = ("Distance (m)", "Angle(°)", "Motion")
tree = ttk.Treeview(frame, columns=columns, show="headings")
for col in columns:
    tree.heading(col, text=col)
    tree.column(col, width=100)
tree.pack()
# Matplotlib Figures (Polar + Heatmap)
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
heatmap_fig, heatmap_ax = plt.subplots()
heatmap_canvas = FigureCanvasTkAgg(heatmap_fig, master=root)
heatmap_canvas.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
# Data Storage
distances, angles, motion_detected = [], [], []
def get_sensor_data():
    """ Continuously fetch LIDAR & Radar data in background """
    global distances, angles, motion_detected
    for scan in lidar.iter_scans():
        distances, angles, motion_detected = [], [], []
        for (_, angle, distance) in scan:
            if distance > 0:
                distances.append(distance / 1000)
                angles.append(np.radians(angle))
                motion_detected.append("Motion" if GPIO.input(RADAR_PIN) == 1 else "No
Motion")
        time.sleep(0.1)  # Prevent excessive CPU usage
def update_ui():
    """ Update Tkinter Table and Plots """
    tree.delete(*tree.get_children())  # Clear table
    if distances and angles:
        for i in range(len(distances)):
            tree.insert("", "end", values=(f"{distances[i]:.2f}", f"{np.degrees(angles[i]):.2f}",
motion_detected[i]))
        ax.clear()
```

```
        ax.set_ylim(0, 6)
        colors = ['red' if m == "Motion" else 'blue' for m in motion_detected]
        ax.scatter(angles, distances, c=colors, alpha=0.75)
        canvas.draw_idle()
        heatmap_ax.clear()
        heatmap, _, _ = np.histogram2d(distances, angles, bins=(50, 50))
        heatmap_ax.imshow(heatmap.T, origin='lower', cmap='hot', interpolation='nearest')
        heatmap_canvas.draw_idle()

    root.after(500, update_ui)  # Refresh every 500ms
# Start background thread
threading.Thread(target=get_sensor_data, daemon=True).start()
# Start UI update loop
root.after(500, update_ui)
root.mainloop()
# Cleanup on exit
lidar.stop()
lidar.disconnect()
GPIO.cleanup()
```