

Engineering and Implementation of a Real-Time Image Processing System for Navigational Assistance to Visually Impaired Individuals

A report submitted in partial fulfillment of the requirements of degree of

Bachelor of Technology

in

Electronics and Communication Engineering

By

Amit Kumar Singh (D/21/EC/025)

Diana Khwairakpam (D/21/EC/026)

Kiranmayee Sharma (D/21/EC/028)

Ngasepam Victoria Devi (D/21/EC/031)

Progyan Jyoti Thakur (D/21/EC/040)

Under Guidance of

Dr. Rajesh Kumar

and

Dr. Joyatri Bora Hazarika

Professor, Dept of ECE

Professor and HoD, Dept of ECE



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
NORTH-EASTERN REGIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED-TO-BE-UNIVERSITY)
UNDER MINISTRY OF EDUCATION, GOVT OF INDIA
NIRJUI – 791109, ARUNACHAL PRADESH**

JUNE 2025

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to the **Northeastern Regional Institute of Science and Technology (NERIST)**, and the **Department of Electronics and Communication Engineering**, for providing the opportunity, infrastructure, and academic environment to carry out this project successfully.

I am deeply thankful to my project guides, **Dr. Rajesh Kumar** and **Dr. Joyatri Bora Hazarika**, for their invaluable guidance, constant encouragement, and insightful suggestions throughout the course of this work. Their expertise and mentorship were crucial in shaping this project.

I would also like to sincerely thank **Dr. Ashok Kumar Ray** for his external support and valuable technical input that greatly enhanced the depth and quality of the project.

My deepest appreciation goes to the team members — **Progyan Jyoti Thakur, Kiranmayee Sharma, Ngasepam Victoria Devi, Diana Khwairakpam, and Amit Kumar Singh** — for their dedication, collaborative spirit, and unwavering support during every stage of this work.

Lastly, I extend my profound gratitude to my family and close friends for their moral support, patience, and encouragement, which kept me motivated throughout this journey.

List of Figure

Figure 1: Raspberry pi.....	13
Figure 2: Pi Camera	14
Figure 3: Ultrasonic Sensor	15
Figure 4: System Architecture Diagram	18
Figure 5: Flowchart of the System	41
Figure 6: Hardware Connections	42
Figure 7: System	42
Figure 8: Workplace	43
Figure 9: Raspberry pi in use	43
Figure 10: Taking image for Database	44
Figure 11: Model Training	45
Figure 12: Face Recognition Output	45
Figure 13: Face Recognition with Object Recognition Output	46
Figure 14: Graph of impact of cv_scaler on FPS and Face Recognition efficiency	46
Figure 15: Graph of Model Training and Testing Loss Curve	47
Figure 16: Efficiency of YOLO on Different Objects	47

TABLE OF CONTENT

Chapter / Section	Page No.
Acknowledgment	ii
List of Figures	iii
Abstract	iv
Chapter 1: Introduction	1
1.1 Background & Motivation.....	1
1.2 Problem Statement.....	2
1.3 Objective.....	3
1.4 Scope of the Project.....	4
Chapter 2: Literature Review	5
2.1 Computer Vision in Assistive and Embedded Systems.....	5
2.2 Face Recognition Algorithms.....	8
2.3 YOLO Object Detection.....	11
2.4 Raspberry Pi Architecture and Microprocessor.....	13
Chapter 3: System Architecture	16
3.1 Methodology.....	16
3.2 System Architecture.....	19
3.3 Component Description.....	21
3.4 Additional Tools Used.....	23

Chapter / Section	Page No.
3.5 Algorithm.....	24
3.6 Flowchart.....	27
3.7 Hardware Setup.....	28
3.8 Workplace & Hardware.....	29
Chapter 4: Implementation & Result.....	30
4.1 Step-by-Step Implementation.....	30
4.2 Code Explanation.....	33
4.3 Modules Description.....	45
4.4 Graphs and Screenshots.....	49
Chapter 5: Conclusion and Future Scope.....	52
5.1 Summary of Achievement.....	52
5.2 Limitations.....	53
5.3 Future Scope.....	54
5.4 Conclusion.....	56
Chapter 6: References.....	57

ABSTRACT

This project presents the design, development, and implementation of a smart assistive system tailored to improve the mobility, situational awareness, and social interaction of visually impaired individuals. The system aims to address critical challenges faced by users with partial or complete vision loss, particularly in recognizing people, identifying nearby objects, and interpreting written text. To meet these goals, the solution integrates real-time facial recognition, object detection, and optical character recognition (OCR) with audio feedback capabilities, forming a comprehensive aid for everyday navigation and interaction.

At the heart of the device lies the Raspberry Pi, a compact and cost-effective computing platform well-suited for portable applications. It is paired with a Pi Camera module for visual input and headphones for delivering verbal cues to the user. The entire system runs on Python and leverages state-of-the-art computer vision and machine learning libraries. Facial recognition is performed using the `face_recognition` library, which compares real-time camera input with a custom-trained face database. Object detection is handled by a lightweight and efficient YOLOv5 model, capable of recognizing a wide range of common objects in real-time. For reading textual content from the environment, the system incorporates Tesseract OCR, allowing it to capture and vocalize printed or handwritten text.

Audio feedback is a central feature of the system, designed to keep users informed without relying on a visual display. Text-to-speech (TTS) conversion is implemented using tools such as `espeak` and `pyttsx3`, which announce recognized faces, detected objects, and any readable text. This enables users to remain aware of their surroundings in both indoor and outdoor environments. The system also includes performance optimizations such as frame resizing and efficient face matching to maintain smooth operation and acceptable frame rates on limited hardware.

During testing, the prototype demonstrated consistent performance in recognizing known individuals and detecting everyday objects under varying lighting and environmental conditions. The system's modular design and low power consumption make it suitable for wearable deployment, such as mounting on eyeglasses or headgear, without compromising

mobility. Though the current version does not include obstacle detection or GPS-based navigation, these features are part of the future development roadmap.

In conclusion, this project contributes significantly to the field of assistive technology by delivering a functional, scalable, and user-friendly system that empowers visually impaired individuals to navigate and interact with their environment more confidently and independently.

CHAPTER 1

INTRODUCTION

1.1 Background & Motivation

Technology has the potential to be a great equalizer in society, especially when it is used to improve the quality of life for those with disabilities. As engineering students, we constantly seek opportunities to not just build innovative solutions, but to make those solutions meaningful and impactful. One such opportunity presented itself not through research papers or classroom discussions, but through real life a simple, yet profound encounter that shifted our perspective.

In our regular day, we frequently came across a visually impaired man navigating the same footpath as us. Over time, we observed how dependent he was on his cane and on strangers' help to move around safely. We experienced and empathized with them that how hard is it for them and discussed among us.

It was this incident that inspired us to pursue a project that could provide practical assistance to visually impaired individuals using the tools we are learning to master. We envisioned a system that could act like a "digital companion" one that could observe the environment, recognize people, detect obstacles, and verbally communicate with the user in real-time.

With recent advancements in computer vision, artificial intelligence, and affordable embedded platforms like the Raspberry Pi, such an idea was no longer confined to theoretical exploration. Technologies like facial recognition, object detection through YOLOv5, and audio feedback systems such as text-to-speech engines have become not only accessible but also highly customizable. These technologies empowered us to bring our vision to life in the form of a real-time assistive device tailored to the needs of the visually impaired.

This project is not just a final-year academic requirement for us — it is a mission to use our knowledge to create a tool that can bring safety, confidence, and social engagement back into the lives of those who have been underserved by mainstream technological

progress. It is our way of giving back to the community and making our engineering education count in a socially impactful way.

1.2 Problem Statement

Visually impaired individuals face significant challenges in navigating their surroundings safely and independently. Unlike those with normal vision, they are unable to detect obstacles, identify people in their vicinity, or interact easily with dynamic environments such as crowded streets, public transportation, or even indoor spaces. This lack of situational awareness can lead to anxiety, disorientation, and a greater risk of accidents, especially in unfamiliar areas.

Traditional assistive tools such as white canes and guide dogs have served this community for decades. While these tools are essential and effective to a certain extent, they have inherent limitations. White canes only detect obstacles that are in immediate contact range, and guide dogs require extensive training and come with high costs and maintenance. Additionally, these aids do not provide information about people in the user's vicinity or enable verbal interaction with the environment.

With the increasing availability of affordable embedded hardware and powerful machine learning models, there exists an opportunity to bridge this gap by developing an intelligent, wearable device that combines face recognition, object detection, and audio feedback into a single system. Such a system could identify known individuals, detect obstacles at a distance, and communicate this information to the user through spoken language, enabling more confident and informed mobility.

However, the real-world deployment of such a system introduces challenges in terms of hardware optimization, processing speed, real-time performance, power efficiency, and reliability in various lighting or environmental conditions. Therefore, the core problem this project addresses is:

How can we design a compact, cost-effective, and real-time assistive system that empowers visually impaired users to navigate safely and socially interact through automated facial recognition, object detection, and voice feedback?

Our project seeks to provide a practical, scalable, and user-friendly solution to this problem using open-source technologies and readily available hardware, thus making a significant contribution to inclusive and assistive technological development.

1.3 Objective

The primary objective of our project is to develop an intelligent assistive device that enhances the situational awareness of visually impaired individuals by providing real-time facial recognition and object detection, along with audio feedback. The system aims to improve safety, mobility, and social interaction.

The specific objectives of the project are as follows:

1. **To design and implement a vision-based system** capable of detecting and recognizing human faces from a live camera feed.
2. **To integrate object detection using machine learning models** to identify obstacles in the surrounding environment.
3. **To develop an audio feedback system** that communicates facial recognition and object detection results to the user in real time.
4. **To ensure real-time performance** on a compact, power-efficient embedded platform (Raspberry Pi).
5. **To maintain a local database** of known faces that can be updated or modified as needed by the user or caregiver.
6. **To build a user-friendly system** that is easily deployable and requires minimal technical knowledge for setup or use.

1.4 Scope of the Project

This project is focused on the development of a prototype assistive device that can serve as a supportive tool for visually impaired individuals. The device leverages computer vision, machine learning, and speech synthesis technologies to offer real-time interaction with the environment.

The scope includes:

- **Hardware Integration:** Using Raspberry Pi, Pi Camera, and ultrasonic sensors to collect live video and distance data.
- **Face Recognition:** Identifying known individuals by comparing captured face encodings to a stored dataset.
- **Object Detection:** Employing YOLOv5 to detect and label objects that could act as obstacles.
- **Audio Feedback:** Converting recognized visual information into speech using text-to-speech (TTS) tools like espeak or pyttsx3.
- **Offline Operation:** Ensuring the system works without internet dependency by using locally stored models and data.
- **Real-time Performance:** Optimizing for low-latency processing to provide immediate feedback.

The scope does not include:

- GPS-based navigation (reserved for future enhancement)
- Multi-language support for audio feedback
- Cloud-based facial recognition or remote-control functionality

CHAPTER 2

LITERATURE REVIEW

2.1 Computer Vision in Assistive and Embedded Systems

Computer Vision (CV) has emerged as a transformative field within artificial intelligence, enabling machines to interpret and understand visual information from the world. Its integration into assistive technologies and embedded systems has significantly advanced over the last decade, particularly in applications aimed at enhancing the autonomy of individuals with visual impairments.

2.1.1. Computer Vision in Assistive Technologies

Recent advancements in computer vision have enabled the development of real-time assistive systems that interpret environmental cues and communicate them to visually impaired users. These systems often incorporate facial recognition, object detection, scene understanding, and text reading functionalities.

For example, wearable smart glasses like the **OrCam MyEye** and **Envision Glasses** use onboard CV modules to read text, recognize faces, and identify objects. Research by **Lourdes Santhosh et al. (2020)** implemented a YOLO-based object detection model on a Raspberry Pi for obstacle avoidance and speech feedback, validating the practicality of using computer vision on low-cost embedded platforms for real-world assistance.

Scene text recognition using OCR techniques (like Tesseract) integrated with cameras and audio output has enabled real-time reading of signs, bills, and labels — critical for navigation and daily activities. Moreover, **facial recognition** helps users identify acquaintances, fostering social interaction.

However, challenges such as inconsistent lighting, occlusions, and computational limitations still affect the performance of these systems, especially when deployed on embedded devices without GPU support.

2.1.2. Real-Time Image Processing for Embedded Vision

Real-time processing is essential for user safety and responsiveness in assistive applications. Techniques like image downscaling, frame skipping, and use of lightweight neural networks (e.g., MobileNet, YOLOv5n) are commonly employed to meet latency constraints.

OpenCV is a widely used library in embedded CV systems due to its efficiency and ease of integration. When combined with Python, it allows for rapid prototyping of face detection (Haar cascades), object tracking, and basic image transformations.

For learning-based models, **TensorFlow Lite** and **PyTorch Mobile** are the leading frameworks enabling deep learning inference on edge devices. Researchers often convert pre-trained CNNs to these formats and optimize them using quantization or pruning to reduce resource consumption.

2.1.3. Hardware and Deployment Considerations

Embedded systems like Raspberry **Pi**, **Jetson Nano**, and **Google Coral** are popular platforms for deploying CV-based assistive systems. While the Raspberry Pi provides broad community support and ease of use, it is limited in terms of processing power and lacks a built-in neural processing unit (NPU). Jetson Nano and Coral offer better inference speed due to their GPU/TPU accelerators but at higher cost and complexity.

Power efficiency, thermal control, and camera integration are crucial factors in designing such systems for continuous outdoor or wearable use. Studies by **Zhang et al. (2023)** and others demonstrate successful real-time deployment by optimizing inference pipelines and minimizing redundant computation.

2.1.4. Key Challenges and Research Gaps

Despite progress, current CV-based assistive technologies face several challenges:

- **Lighting Variability:** Outdoor and nighttime environments drastically affect detection accuracy.
- **Occlusion and Clutter:** Crowded scenes hinder recognition and tracking.
- **Real-Time Constraints:** Maintaining <100ms response time is difficult on edge devices without acceleration.
- **User Interface and Feedback:** Conveying useful information via audio without overwhelming the user remains an open design problem.

Future research is moving toward multi-modal systems that combine visual, spatial, and auditory data for enhanced contextual awareness.

2.2 Face Recognition Algorithms

Classical face recognition methods rely on a two-stage pipeline of detection followed by identification. Early approaches used hand-crafted features: for example, Haar cascade classifiers (Viola–Jones) quickly scan images for faces. Haar cascades use simple rectangular features in a cascade of boosted classifiers. They are extremely fast and effective for real-time face *detection* on CPUs. In fact, Viola and Jones’ cascade is praised for “rapid processing speed and high detection accuracy, making it an ideal choice for real-time face detection”. However, Haar detectors can suffer many false positives and are less robust to large pose or illumination changes than modern methods.

Once a face is detected, traditional face recognition methods extract features to identify the person. A widely used classical method is Local Binary Patterns Histograms (LBPH). LBPH encodes each pixel by thresholding its 3×3 neighborhood into a binary code and then pools the codes into a histogram. This yields a simple descriptor that is *computationally efficient* and robust to lighting and expression changes. For example, Nuety *et al.* note that LBPH is “simple and effective in handling changes in facial expressions, lighting, occlusions, distance, and camera resolution”. In practical terms,

LBPH recognition is often implemented in OpenCV and can run in real-time on modest hardware, making it suitable for embedded systems. Its accuracy on standard tasks is moderate (often 80–90% on controlled datasets), significantly lower than deep models, but sufficient for constrained scenarios (e.g. attendance checklists).

More recent deep learning methods have dramatically improved face recognition accuracy. Convolutional Neural Networks (CNNs) like FaceNet or ArcFace learn rich embeddings from large face datasets. State-of-the-art CNN models routinely achieve >99% accuracy on public benchmarks (e.g. LFW) by projecting faces to a 128D embedding space where distance reflects identity. These methods require substantial computation: training often uses GPUs for days, and even inference is heavier than classical methods. For instance, Dürr *et al.* demonstrated a CNN pipeline on a Raspberry Pi 3 (after offline training) yielding ~97% recognition accuracy but only ~2 frames per second. Their CNN-based system “outperformed all of OpenCV’s algorithms with respect to both accuracy and speed” on that platform, indicating that deep models can exceed classical methods even on limited hardware – but at a cost of higher latency. In general, deep CNNs offer far better recognition accuracy (especially under varied conditions) but require more computers. Techniques like model quantization, pruning or using lightweight backbones (e.g. MobileNet) are often needed to achieve near-real-time rates on embedded CPUs.

A popular face-recognition framework is Adam Geitgey’s `face_recognition` library, which leverages the Dlib toolkit. Internally it uses a pretrained ResNet-like CNN to compute a 128-dimensional face descriptor. Specifically, Dlib’s model is a variant of a ResNet-34: it uses the ResNet architecture but with half the filters and fewer layers (29 layers total) to output a 128-D embedding. In practice, `face_recognition` uses dlib’s HOG or CNN detector for the face and then this ResNet-derived network for encoding. This yields very high accuracy (on par with FaceNet) at the cost of speed. On a modern CPU, encoding one face might take tens of milliseconds; on a Raspberry Pi it is much slower, often only a few FPS. Thus, `face_recognition` (Dlib+ResNet) is highly accurate for identifying faces, but its computational cost limits its use in real-time embedded systems without acceleration.

Summary: Haar cascades provide extremely fast face detection (good for first stage), LBPH gives a light-weight recognition step robust to simple variations, while CNN-based systems

(including Dlib/ResNet) yield far higher accuracy at much greater computational cost. In embedded or assistive applications, designers often use Haar+LBPH or lightweight CNNs for real-time constraints, whereas high-accuracy DNNs are used when performance permits.

2.3 YOLO Object Detection

The YOLO (“You Only Look Once”) family of object detectors revolutionized real-time detection by processing entire images with a single CNN pass. YOLOv1 introduced this concept in 2015, achieving real-time detection with impressive accuracy by framing detection as a single regression problem. Subsequent versions improved both speed and accuracy. YOLOv2 (2016) replaced the backbone with Darknet-19 and added batch normalization and data augmentation. YOLOv3 (2018) used a deeper Darknet-53 backbone with a Feature Pyramid Network-like design for multiscale detection, boosting small-object accuracy. Later, YOLOv4 added Spatial Pyramid Pooling and a Path Aggregation Network to better fuse features.

YOLOv5 (2020) by the Ultralytics team marked another milestone. While built on insights from YOLOv4, YOLOv5 is implemented in PyTorch, making it more accessible. It introduced **SPPF** (Fast Spatial Pyramid Pooling) and strided convolutions to reduce computation. Advanced loss functions (e.g. Complete IoU) and data augmentations further improved precision. Each YOLOv5 model (Nano, Small, Medium, Large) offers a trade-off of size vs. accuracy, allowing deployment from cloud to edge. More recently, YOLOv6/7/8 (2022–2023) incorporated additional optimizations (e.g. RepVGG blocks, NAS, attention) for efficiency.

A key strength of YOLO is **real-time performance**. Even early YOLO models could run at high frame rates on GPUs. For example, a lightweight YOLOv4-tiny achieved **371 FPS** on a high-end GPU (NVIDIA GTX 1080 Ti) while meeting application accuracy, “greatly increasing the feasibility” of deploying detection on embedded/mobile platforms. In practice, YOLOv5 on modern GPUs or accelerators can exceed 30–60 FPS at 640×640

resolution. On embedded hardware (like Raspberry Pi or Jetson), smaller YOLO variants (Tiny or Nano) are often used. Their reduced complexity allows modest frame rates – e.g. trimmed YOLOv5 models can reach on the order of 10–20 FPS on a Pi 4 with optimization. Some studies also use hardware acceleration (e.g. TensorRT, NPU) to speed up inference.

YOLO’s efficiency and accuracy make it attractive for **assistive and embedded applications**. For instance, in assistive wearable systems, Lourdes Santhosh et al. demonstrated a YOLO-based smart glasses: a Raspberry Pi with a camera runs a YOLO detector to recognize obstacles and objects, converting detections into spoken feedback. Similarly, Zhang *et al.* (2024) developed an indoor object-finding system for the blind by enhancing YOLOv5 (with a lightweight GhostNet backbone and attention) and deploying it on a Raspberry Pi 4 plus a depth camera. In field tests, this system could accurately detect key objects (keys, cups, bottles) and report their positions to the user, showing the practical viability of YOLOv5 on Pi hardware.

2.4 Raspberry Pi Architecture and Microprocessor

The Raspberry Pi series are credit-card sized Linux computers featuring Broadcom SoCs built around ARM cores. Recent models include:

- **Raspberry Pi 5 (2023):** Broadcom BCM2712 SoC with a **quad-core ARM Cortex-A76** CPU at 2.4 GHz, yielding roughly 2–3× the CPU performance of Pi 4. It has 512 KB L2 cache per core and 2 MB shared L3 cache. The GPU is a VideoCore VII at 800 MHz, supporting OpenGL ES 3.1 and Vulkan 1.2. RAM options are LPDDR4X-4267 ranging from 2 GB up to **16 GB**. Notably, Pi 5 adds significant I/O upgrades: a PCIe 2.0 x1 link for high-bandwidth peripherals, dual four-lane MIPI CSI/DSI interfaces (tripling camera/display bandwidth).
- **Raspberry Pi 4 (2019):** Broadcom BCM2711 with a **quad-core ARM Cortex-A72** at 1.5 GHz (ARM v8 64-bit). It offered 1–8 GB of LPDDR4 RAM (e.g. 8 GB on high-end models). The GPU is VideoCore VI (500 MHz), and it supports dual

4Kp60 HDMI outputs. Connectivity includes Gigabit Ethernet, 802.11ac Wi-Fi/Bluetooth 5.0, two USB 3.0 + two USB 2.0 ports, and dual micro-HDMI.

Across models, the Pi provides a **40-pin GPIO header**. On Pi 4, this includes I2C, SPI, UART, PWM, and other signals for sensors and custom peripherals. There is also a **CSI camera interface**: Pi 4 has one 2-lane MIPI CSI port (enabling the official Pi Camera), while Pi 5 has two 4-lane CSI ports, allowing simultaneous dual cameras. For displays, a DSI interface is provided. A microSD slot holds the OS, and for the first time Pi 5 offers PCIe, enabling devices like USB3 host or NVMe storage.

Strengths: Raspberry Pis are low-cost, power-efficient, and well-supported by a large community and software ecosystem. Their ARM CPUs and GPUs can run a full Linux desktop or headless server. The VideoCore GPUs handles media decoding (e.g. 4K H.265 hardware decodes) and OpenGL/Vulkan graphics. Importantly for vision, Pi’s GPIO and camera interface allow easy attachment of cameras and sensors. TensorFlow Lite and OpenCV are available, and some libraries (e.g. OpenVINO) can leverage NEON SIMD on the CPU. With Pi 5’s PCIe, even USB camera or hardware accelerators (e.g. Coral Edge TPU) can be attached. These make the Pi a popular platform for DIY computer vision and robotics.

Limitations: Compared to desktop PCs or specialized accelerators, Raspberry Pis have limited computers. The CPU cores (even A76) cannot match x86 CPUs or dedicated AI ASICs. There is no on-chip NPU for neural networks (aside from the upcoming Pi 5’s PCIe which allows add-ons). Memory bandwidth and GPU performance are modest. As a result, running complex CNNs on-device can be slow. For example, Dürr *et al.* achieved only ~2 FPS with a CNN on Pi 3. Performance drops further for high-resolution inputs. In practice, heavy models require offloading or extremely lightweight networks. Power consumption is low (typically <15 W), but thermal throttling can occur under sustained load.

Use in Assistive Vision: Despite limits, Raspberry Pis has powered many assistive vision projects. For example, the “VPI” system by Xinnan Leong used a Pi with a camera and sonar sensor running a CNN (YOLO) to detect obstacles and estimate distance, achieving real-time navigation support. Lourdes Santhosh’s wearable glasses (YOLO-based) ran on

a Pi to announce obstacles aloud. Zhang *et al.*'s object-finding aid for the blind (above) runs YOLOv5 on a Pi 4 and depth camera to locate household items. These examples show that, while limited, the Raspberry Pi's CPU and ISP (Image Signal Processor) can handle lightweight AI tasks, especially when models are optimized. Typical strategies include running at lower resolution, using model quantization, or relying on auxiliary accelerators.

Summary: The Raspberry Pi combines a modest ARM processor (Cortex-A72/A76) and a Video Core GPU with flexible I/O (GPIO, CSI cameras, etc.). It excels as a low-cost, general-purpose embedded platform, but its computer power sets the practical limits for on-device AI. For computer vision tasks, the Pi can run basic neural networks or classical algorithms in real time, but advanced CNNs often require compromise or hardware support. Nonetheless, numerous vision-based assistive devices have been built on Raspberry Pi, demonstrating its strengths in prototyping and deployment of embedded computer vision.

CHAPTER 3

SYSTEM ARTITECTURE

3.1 Methodology

To transform our idea of an assistive device for visually impaired individuals into a working prototype, we adopted a structured and iterative development approach. The methodology includes problem understanding, hardware and software selection, development, testing, and integration, while ensuring real-time performance and ease of use.

The major phases of our methodology are outlined below:

3.1.1. Requirement Analysis and Problem Understanding

We began by observing real-world challenges faced by visually impaired individuals and documented their needs for mobility, safety, and interaction. Based on this, we defined clear functional and non-functional requirements, such as real-time face recognition, object detection, and voice feedback. This helped in setting achievable goals and identifying necessary tools and technologies.

3.1.2. Technology and Component Selection

To ensure affordability and ease of integration, we selected:

- **Raspberry Pi** is the central processing unit due to its balance of power and portability.



Fig 3.1 Raspberry pi

- **Pi Camera** for capturing real-time video frames.



Fig 3.2 Pi Camera

- **Ultrasonic sensors** (for optional obstacle sensing in dark or low-visibility conditions).



Fig 3.3 Ultrasonic Sensor

- **Python** as the programming language for its extensive support in computer vision and ML.
- **OpenCV, face_recognition, YOLOv5, and text-to-speech (TTS)** libraries as our software stack.

We also reviewed other potential alternatives and chose components based on performance, compatibility, and ease of deployment.

3.1.3. Data Collection and Training Preparation

We developed a custom image-capturing script to take multiple facial images of known individuals. These images were stored in a structured database. Then, we wrote a separate training script to extract facial encodings and store them in a .pickle file. This static database will later be used to compare faces detected during real-time execution.

3.1.4. Modular Code Development

We divided the project into functional modules:

- **Face Capture & Encoding Module**
- **Object Detection Module**
- **Audio Output Module**
- **Integration & Real-time Frame Handling**

Each module was written, tested, and debugged independently to ensure it performs its designated task efficiently. This modular structure also allows for easier future upgrades, such as GPS or cloud integration.

3.1.5. Integration and System Testing

Once individual modules were validated, we integrated them into a unified application. Real-time data from the Pi Camera was fed simultaneously into both face recognition and object detection models. We tested the complete system in controlled environments to ensure:

- Correct detection of known faces
- Real-time obstacle identification
- Accurate and timely voice output

3.1.6. Optimization for Real-time Performance

To maintain smooth real-time performance, we optimized:

- Frame resizing and color conversion speeds
- Processing frequency (by skipping every nth frame if needed)
- Model selection (using lightweight YOLOv5s for speed)
- TTS response delays

We also tuned the system to work under variable lighting and motion scenarios typical in real-world use cases.

3.1.7. Final Testing and Use-case Simulation

We simulated real-life use by placing the device in environments such as corridors, rooms, and open areas with multiple faces and objects. The system's response, recognition accuracy, and voice outputs were observed and adjusted for better usability.

3.1.8. Documentation and User Instructions

We documented the setup and use procedures, so even non-technical users (like caregivers or family members) can easily operate or update the system. A manual and troubleshooting guide were also planned for inclusion.

3.2 System Architecture

The proposed system is designed as a real-time, vision-based assistive device that captures the surrounding environment, detects and recognizes human faces, identifies nearby objects, and communicates the information to the user via audio output. The architecture follows a modular and layered approach, ensuring maintainability, scalability, and real-time performance.

3.2.1. High-Level Architecture

The system can be logically divided into the following main subsystems:

1. **Image Acquisition Module**

Captures continuous frames from the camera using the Pi Camera module connected to the Raspberry Pi.

2. **Face Detection and Recognition Module**

Uses computer vision and machine learning algorithms (e.g., Dlib + ResNet via `face_recognition`) to detect faces and compare them against a known database.

3. **Object Detection Module**

Uses a pre-trained YOLOv5 model to identify common objects in the camera's view that might be of interest or pose a threat (e.g., vehicles, poles, animals).

4. **Audio Output Module**

Converts recognized faces or objects into speech feedback using `espeak` or `pyttsx3`.

5. **Control and Decision Logic**

Coordinates input from the camera and sensor modules, triggers detection functions, and manages output timing and priorities.

3.2.2. System Flow

1. The camera captures real-time video frames.
2. Each frame is sent to the face recognition and object detection modules.
3. If a known face is detected, the system generates a specific voice message.
4. If an obstacle or general object is detected, a general warning or description is voiced.
5. The loop continues, updating feedback with each frame unless the user chooses to exit the program.

3.2.3 System Architecture Diagram

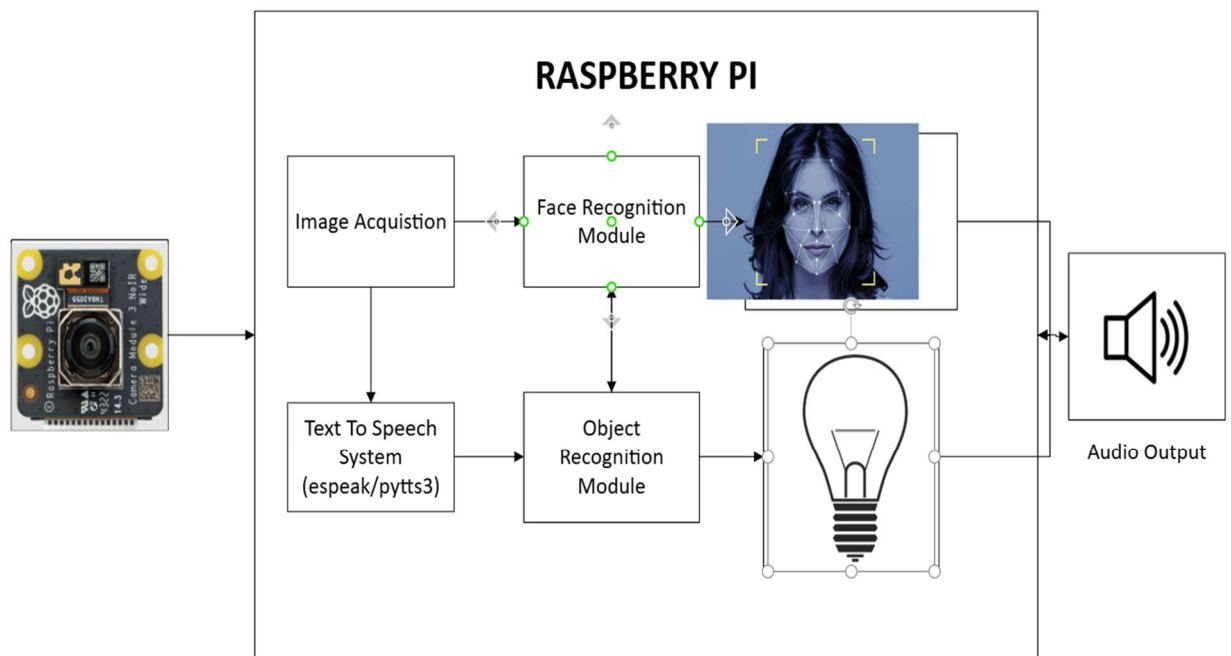


Fig 3.4 System Architecture Diagram

3.3 Component Description

3.3.1 Hardware Component

Table I
Used Hardware Component

Sl. No.	Component	Description
1	Raspberry pi	Acts as the central processing unit. Performs image processing and logic.
2	Pi Camera	Captures real-time video input for analysis. Connected via CSI interface.
3	Speaker	Outputs real-time speech feedback to the user.
4	Sensor	Detect physical environment not captured visually.
5	Power Supply	Provides portable power for field operation.
6	Keyboard	Use to Interact with Raspberry pi Software, Noob OS in particular
7	Monitor	Project Visual information of Raspberry Pi and Output is also shown here

3.3.2 Software Component

Table II
Used Software Modules

Sl. No.	Software/Library	Description
1	Python	Main programming language used for logic and control.
2	OpenCV	Used for image processing, frame handling, and visualization.
3	face_recognition	The library is based on Dlib and ResNet for face detection and recognition.
4	YOLOv5 (Ultralytics)	Deep learning-based object detection model using PyTorch.
5	torch / torchvision	Backend for running the YOLOv5 model.
6	pickle	For storing and loading encoded facial data.
7	espeak / pyttsx3	Text-to-Speech libraries for converting results into voice output

3.4. Additional Tools Used

- Tesseract OCR (in extended versions): For reading printed or written text in the environment.
- Picamera2 library: Used to access the Raspberry Pi Camera and configure video resolution/frame rate.
- NOOB OS: New Out Of Box Software, is a simple operating system installer for the Raspberry Pi

Key Features of System Architecture

- Real-time processing using lightweight models optimized for embedded platforms.
- Offline capability (no internet required), ensuring portability and privacy.
- Expandable: GPS navigation, cloud integration, or emergency response systems can be added later.
- Modular design allows independent testing and upgrades of each component.

3.5. Algorithm

The algorithm is designed to continuously process input from the camera, detect and recognize faces and objects, and provide audio feedback in real-time. It is optimized for embedded hardware constraints and is written in modular form for ease of maintenance and scalability.

Algorithm: Assistive Vision System for the Visually Impaired

Step 1: System Initialization

- 1.1 Initialize Raspberry Pi, load required Python libraries (e.g., OpenCV, face_recognition, torch, pyttsx3/espeak).
- 1.2 Load the trained facial encodings from the .pickle file.
- 1.3 Load the pre-trained YOLOv5 model for object detection.
- 1.4 Initialize the Pi Camera using Picamera2 and configure the resolution.
- 1.5 Initialize the Text-to-Speech (TTS) engine.

Step 2: Start Main Loop

2.1 Continuously capture frames from the camera.

Step 3: Preprocess the Frame

3.1 Resize and convert the frame to RGB color format.

3.2 Reduce frame resolution (scaling factor) to optimize processing time.

Step 4: Face Detection and Recognition

4.1 Detect all face locations in the frame using `face_recognition.face_locations()`.

4.2 For each detected face, compute its facial encoding.

4.3 Compare the new encoding with known encodings to identify the person.

4.4 If a match is found, assign the recognized name.

4.5 If no match is found, label the face as "Unknown".

Step 5: Object Detection using YOLO

5.1 Input the current frame into the YOLOv5 model.

5.2 Extract labels, confidence scores, and bounding box coordinates.

5.3 Filter out low-confidence detections using a threshold (e.g., 0.5).

Step 6: Decision Logic for Audio Feedback

6.1 If a new known face is detected (not previously announced), generate speech:

→ "<Name> is coming your way."

6.2 If an unknown face is detected, generate speech:

→ “Someone is coming your way.”

6.3 If a new object is detected (not previously spoken), generate speech:

→ “<*Object*> is in front of you.”

Step 7: Output Results

7.1 Draw bounding boxes and labels on the frame for debugging/monitoring.

7.2 Convert text into speech using pyttsx3 or espeak.

7.3 Output the audio feedback to headphones or speaker.

Step 8: Exit Condition

8.1 Check if the user presses the ‘Q’ key.

8.2 If yes, break the loop and terminate the program.

8.3 Stop the camera and release all system resources.

End of Algorithm

CHAPTER 4

IMPLEMENTATION

4.1 Step-by-Step Implementation

Step 1: Raspberry Pi Setup

- Install the latest **Raspberry Pi OS** on a microSD card.
- Update all system packages:

```
sudo apt update && sudo apt upgrade
```

- Enable the camera interface via `raspi-config`.
- Make a virtual environment

```
venv face_rec|
```

- Start virtual environment

```
source face_rec/bin/activate
```

- Install necessary Python packages:

```
pip install face_recognition opencv-python torch torchvision picamera2 pyttax3 espeak
```

Step 2: Image Capture for Face Dataset

- Write a Python script to:
 - Activate the Pi Camera.
 - Capture multiple images of everyone at different angles and expressions.
 - Save the images in a folder with the person's name as the directory label (e.g., dataset/John/).

- Prompt the user to press the spacebar to capture images manually.
- Capture at least 20–30 images per person for reliable recognition.

Step 3: Face Encoding and Model Preparation

- Develop a script to:
 - Read each person's image folder.
 - Extract facial encodings using the `face_recognition` library.
 - Store these encodings and their corresponding names in a `.pickle` file for later use.
- This acts as the “training” phase for face recognition.

Step 4: Load and Configure Object Detection Model

- Use **YOLOv5s** (small variant) from Ultralytics for real-time object detection:

```
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
```

- Test the model independently to validate its ability to detect common objects.

Step 5: Real-Time Video Feed Setup

- Initialize the Pi Camera using the `picamera2` library.
- Capture frames in a loop and resize to reduce processing time.
- Display frames using OpenCV for visual debugging during development.

Step 6: Face Recognition Implementation

- Load the `.pickle` file containing face encodings.

- For each frame:
 - Detect faces.
 - Encode and compare them with known encodings.
 - If a match is found, label the name; otherwise, tag as "Unknown".
 - Track the last spoken name to avoid repeated outputs.
-

Step 7: Object Detection Integration

- Feed each frame to the YOLOv5 model.
 - Extract the top 1 or top 2 objects with confidence scores > 0.5 .
 - Draw bounding boxes and annotate object names on the frame.
 - Track the last spoken object to avoid redundant feedback.
-

Step 8: Audio Output System

- Integrate pyttsx3 or espeak for speech synthesis:

```
os.system(f'espeak "{message}"')
```

- Convert the following into speech:
 - Recognized face names (e.g., "John is in front of you").
 - Detected objects (e.g., "Chair ahead").
 - Unrecognized face alerts ("Someone is in front of you").

Step 9: User Interaction and Exit

- Display the processed video with labels and FPS.

- Allow the user to exit the program by pressing the 'Q' key.
- On exit:
 - Release the camera.
 - Close all OpenCV windows.
 - Stop audio output if running.

Step 10: Testing and Validation

- Test the system in both indoor and outdoor conditions.
- Validate:
 - Face recognition accuracy.
 - Object detection relevance.
 - Audio clarity and latency.
- Collect feedback and tune thresholds (e.g., confidence score, frame size) as needed.

4.2 Code explanation

4.2.1 Capturing the image of the person to save in database

```
# Import necessary libraries
import cv2                                # OpenCV for image processing and display
import os                                  # OS module to work with directories and file paths
from datetime import datetime             # To generate timestamp for image filenames
from picamera2 import Picamera2          # To control the Raspberry Pi Camera
import time                                # To introduce delays

# Define the name of the person whose photos are being captured
PERSON_NAME = "Xyz"

# Function to create a dataset folder and a subfolder for the person
def create_folder(name):
    dataset_folder = "dataset" # Root folder for dataset
    if not os.path.exists(dataset_folder): # Check if it exists
        os.makedirs(dataset_folder)       # Create if it doesn't

    person_folder = os.path.join(dataset_folder, name) # Subfolder for the specific person
    if not os.path.exists(person_folder):
        os.makedirs(person_folder) # Create person folder if it doesn't exist
    return person_folder           # Return the path for saving photos

# Function to capture photos using Raspberry Pi Camera
def capture_photos(name):
    folder = create_folder(name) # Ensure directory is set up

    # Initialize and configure the Pi Camera
    picam2 = Picamera2()
    picam2.configure(picam2.create_preview_configuration(
        main={"format": 'XRGB8888', "size": (640, 480)})) # Set resolution and format
    picam2.start() # Start camera preview

    time.sleep(2) # Allow camera to warm up

    photo_count = 0 # Counter for saved photos

    print(f"Taking photos for {name}. Press SPACE to capture, 'q' to quit.")

    while True:
        frame = picam2.capture_array() # Capture current frame from the camera

        cv2.imshow('Capture', frame) # Show the frame in a window
```

```

key = cv2.waitKey(1) & 0xFF    # Wait for a key press (non-blocking)

if key == ord(' '): # If spacebar is pressed
    photo_count += 1
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S") # Unique timestamp
    filename = f"{name}_{timestamp}.jpg" # Create a filename
    filepath = os.path.join(folder, filename) # Full path to save image
    cv2.imwrite(filepath, frame) # Save the image
    print(f"Photo {photo_count} saved: {filepath}")

elif key == ord('q'): # If 'q' is pressed, exit loop
    break

# Cleanup: close window and stop the camera
cv2.destroyAllWindows()
picam2.stop()
print(f"Photo capture completed. {photo_count} photos saved for {name}.")

# Main execution
if __name__ == "__main__":
    capture_photos(PERSON_NAME) # Start the photo capturing process

```

4.2.2 Training the model with taken Pictures

```

# Import required libraries
import os
from imutils import paths # Helps to get list of image paths in a directory
import face_recognition # Library for face detection and encoding
import pickle # For saving encodings to a file
import cv2 # OpenCV for reading and processing images

# Inform the user that face processing has started
print("[INFO] start processing faces...")

# Get list of image file paths from the 'dataset' directory
imagePaths = list(paths.list_images("dataset"))

# Lists to store the face encodings and corresponding names
knownEncodings = []
knownNames = []

```

```

# Loop through each image in the dataset
for (i, imagePath) in enumerate(imagePaths):
    print(f"[INFO] processing image {i + 1}/{len(imagePaths)}")

    # Extract the person's name from the directory name (assumes dataset/person_name/image.jpg)
    name = imagePath.split(os.path.sep)[-2]

    # Load the image and convert it from BGR (OpenCV default) to RGB (face_recognition
requirement)
    image = cv2.imread(imagePath)
    rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Detect face locations in the image using HOG model (faster, CPU-based)
    boxes = face_recognition.face_locations(rgb, model="hog")

    # Compute the 128-d face encodings for the detected faces
    encodings = face_recognition.face_encodings(rgb, boxes)

    # Loop over each encoding found in the image
    for encoding in encodings:
        knownEncodings.append(encoding) # Store the encoding
        knownNames.append(name)         # Store the name of the person

# Inform the user that data is being serialized
print("[INFO] serializing encodings...")

# Create a dictionary containing encodings and corresponding names
data = {"encodings": knownEncodings, "names": knownNames}

# Save the dictionary to a file using pickle
with open("encodings.pickle", "wb") as f:
    f.write(pickle.dumps(data))

print("[INFO] Training complete. Encodings saved to 'encodings.pickle'")

```

4.2.3 Face Recognition Code

```
# Import required libraries
import face_recognition # For face detection and recognition
import cv2 # OpenCV for image processing and display
import numpy as np # For numerical operations
from picamera2 import Picamera2 # PiCamera2 for accessing Raspberry Pi camera
import time # For time-related operations
import pickle # For loading saved face encoding data

# Load pre-trained face encodings from a file
print("[INFO] loading encodings...")
with open("encodings.pickle", "rb") as f:
    data = pickle.loads(f.read()) # Deserialize the pickled data
known_face_encodings = data["encodings"] # Known face features
known_face_names = data["names"] # Corresponding names

# Initialize and configure the PiCamera
picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"format": 'XRGB8888', "size": (1920,
1080)}))
picam2.start() # Start capturing

# Define initial variables
cv_scaler = 5 # Scaling factor to reduce image size and speed up processing

face_locations = [] # Detected face locations in the frame
face_encodings = [] # Encoded features of detected faces
face_names = [] # Names corresponding to detected faces
frame_count = 0 # Frame counter for FPS calculation
start_time = time.time() # Start time for FPS calculation
fps = 0 # Frames per second value

# Function to process a video frame and recognize faces
def process_frame(frame):
    global face_locations, face_encodings, face_names

    # Resize frame to speed up processing
    resized_frame = cv2.resize(frame, (0, 0), fx=(1/cv_scaler), fy=(1/cv_scaler))

    # Convert color from BGR (OpenCV default) to RGB (used by face_recognition)
    rgb_resized_frame = cv2.cvtColor(resized_frame, cv2.COLOR_BGR2RGB)

    # Detect face locations and generate face encodings in the resized frame
    face_locations = face_recognition.face_locations(rgb_resized_frame)
```

```

    face_encodings = face_recognition.face_encodings(rgb_resized_frame, face_locations,
model='large')

    face_names = [] # Reset face names list
    for face_encoding in face_encodings:
        # Compare current face encoding with known encodings
        matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
        name = "Unknown" # Default name if no match

        # Find the closest match based on the smallest distance
        face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]: # Confirm if it's an actual match
            name = known_face_names[best_match_index]
        face_names.append(name) # Append matched or default name

    return frame # Return the original (non-resized) frame

# Function to draw rectangles and labels on detected faces
def draw_results(frame):
    for (top, right, bottom, left), name in zip(face_locations, face_names):
        # Scale face locations back to original frame size
        top *= cv_scaler
        right *= cv_scaler
        bottom *= cv_scaler
        left *= cv_scaler

        # Draw a rectangle around the face
        cv2.rectangle(frame, (left, top), (right, bottom), (244, 42, 3), 3)

        # Draw a filled rectangle above the face for label
        cv2.rectangle(frame, (left - 3, top - 35), (right + 3, top), (244, 42, 3), cv2.FILLED)

        # Add name text inside the label rectangle
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name, (left + 6, top - 6), font, 1.0, (255, 255, 255), 1)

    return frame # Return the frame with drawings

# Function to calculate and update Frames Per Second (FPS)
def calculate_fps():
    global frame_count, start_time, fps
    frame_count += 1 # Count this frame
    elapsed_time = time.time() - start_time # Time since last FPS update
    if elapsed_time > 1: # Update every second

```

```

        fps = frame_count / elapsed_time
        frame_count = 0 # Reset frame count
        start_time = time.time() # Reset start time
    return fps # Return calculated FPS

# Main loop: runs continuously to capture, process, and display frames
while True:
    frame = picam2.capture_array() # Capture a frame from the camera

    processed_frame = process_frame(frame) # Detect and recognize faces

    display_frame = draw_results(processed_frame) # Annotate faces with boxes and names

    current_fps = calculate_fps() # Get FPS for performance monitoring

    # Display the FPS value on the video frame
    cv2.putText(display_frame, f"FPS: {current_fps:.1f}", (display_frame.shape[1] - 150, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    # Show the final frame with annotations in a window
    cv2.imshow('Video', display_frame)

    # Exit the loop if the user presses the 'q' key
    if cv2.waitKey(1) == ord("q"):
        break

# Cleanup: close camera and window properly
cv2.destroyAllWindows()
picam2.stop()

```

4.2.4 Face and Object Recognition and Audio assistive Output

```

# Import required libraries
import os # For running system commands
import cv2 # OpenCV for image processing
import numpy as np # For numerical computations
import face_recognition # For facial recognition features
from picamera2 import Picamera2 # Camera interface for Raspberry Pi
import time # To measure FPS and time-based events
import pickle # To load pre-trained face encodings
import torch # PyTorch used to load YOLOv5 model

```



```

# Load known face encodings from disk
print("[INFO] Loading face encodings...")
with open("encodings.pickle", "rb") as f:
    data = pickle.loads(f.read()) # Deserialize the pickle file
known_face_encodings = data["encodings"] # Face vectors
known_face_names = data["names"] # Corresponding names

# Load the pre-trained YOLOv5 object detection model from Ultralytics
print("[INFO] Loading YOLOv5 object detection model...")
yolo_model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

# Initialize and configure PiCamera2
picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"format": 'XRGB8888', "size": (1280,
720)}))
picam2.start()

# Scale factor to reduce frame size for faster face recognition
cv_scaler = 2

# Initialize variables for tracking and performance
face_locations = []
face_encodings = []
face_names = []
frame_count = 0
start_time = time.time()
fps = 0
last_face_spoken = "" # Last recognized face announced
last_object_spoken = "" # Last object announced

# Function to speak a given message using eSpeak
def speak(message):
    os.system(f'espeak "{message}"') # Call eSpeak with message

# Function to process face recognition in a video frame
def process_faces(frame):
    global face_locations, face_encodings, face_names, last_face_spoken

    # Resize frame for faster processing
    small_frame = cv2.resize(frame, (0, 0), fx=1/cv_scaler, fy=1/cv_scaler)

    # Convert image from BGR (OpenCV) to RGB (face_recognition)
    rgb_small_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)

```

```

# Detect faces and get face encodings
face_locations = face_recognition.face_locations(rgb_small_frame)
face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

face_names = [] # Reset the list of face names
for face_encoding in face_encodings:
    matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
    name = "Unknown"
    face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)

    if len(face_distances) > 0:
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = known_face_names[best_match_index]
        face_names.append(name)

# Speak name only if it's different from the last spoken one
if name != last_face_spoken:
    sentence = f"{name} is coming your way" if name != "Unknown" else "Someone is coming
your way"
    speak(sentence)
    last_face_spoken = name

# Function to draw face bounding boxes and names
def draw_faces(frame):
    for (top, right, bottom, left), name in zip(face_locations, face_names):
        # Scale coordinates back to original frame size
        top *= cv_scaler
        right *= cv_scaler
        bottom *= cv_scaler
        left *= cv_scaler

        # Draw the face rectangle
        cv2.rectangle(frame, (left, top), (right, bottom), (244, 42, 3), 2)

        # Draw label background
        cv2.rectangle(frame, (left, top - 30), (right, top), (244, 42, 3), cv2.FILLED)

        # Add text with the name
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(frame, name, (left + 6, top - 6), font, 0.6, (255, 255, 255), 1)

# Function to detect objects using YOLOv5 and speak them out
def detect_objects(frame):
    global last_object_spoken

```

```

results = yolo_model(frame) # Run YOLO model on frame
objects = results.pandas().xyxy[0] # Get detections as pandas DataFrame
spoken = False # To ensure only one object is spoken per frame

for index, row in objects.iterrows():
    label = row['name'] # Object class name
    conf = row['confidence'] # Confidence score
    x1, y1, x2, y2 = int(row['xmin']), int(row['ymin']), int(row['xmax']), int(row['ymax'])

    # Draw bounding box and label
    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
    cv2.putText(frame, f"{label} {conf:.2f}", (x1, y1 - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 1)

    # Speak only the first new object detected in this frame
    if not spoken and label != last_object_spoken:
        sentence = f"{label} is in front of you"
        speak(sentence)
        last_object_spoken = label
        spoken = True

# Function to calculate and return current FPS
def calculate_fps():
    global frame_count, start_time, fps
    frame_count += 1
    elapsed_time = time.time() - start_time
    if elapsed_time > 1:
        fps = frame_count / elapsed_time
        frame_count = 0
        start_time = time.time()
    return fps

# Main loop to run detection and recognition continuously
while True:
    frame = picam2.capture_array() # Capture current frame from PiCamera

    process_faces(frame) # Detect and recognize faces
    detect_objects(frame) # Detect and announce objects
    draw_faces(frame) # Draw rectangles and names for faces

    current_fps = calculate_fps() # Measure performance

    # Display FPS on screen
    cv2.putText(frame, f"FPS: {current_fps:.1f}", (20, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 0), 2)

```

```

# Show the video feed with annotations
cv2.imshow("Face & Object Detection", frame)

# Exit if 'q' key is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Cleanup when exiting the loop
cv2.destroyAllWindows()
picam2.stop()

```

4.3 Modules Description

The system is divided into several key functional modules, each performing a specific task. The modular structure enhances the readability, scalability, and maintainability of the overall application. Below is a detailed description of each module:

4.3.1. Image Acquisition Module

Purpose:

Captures live video frames from the Raspberry Pi Camera for processing.

Functionality:

- Initializes and starts the Pi Camera using the picamera2 library.
- Configures resolution and frame rate.
- Continuously reads frames for real-time analysis.
- Feeds frame to both the face recognition and object detection modules.

4.3.2. Face Capture and Encoding Module

Purpose:

Collects and processes training images of known individuals for face recognition.

Functionality:

- Captures multiple images of a person from different angles.
- Saves the images in labeled folders based on the person's name.
- Extract 128-dimensional facial encodings using the face_recognition library.
- Stores encoded data in a serialized .pickle file to be used during recognition.

4.3.3. Face Detection and Recognition Module

Purpose:

Identifies faces from the live video stream by comparing with stored face encodings.

Functionality:

- Detects face locations in each frame using Histogram of Oriented Gradients (HOG) or CNN models.
- Computes face encodings.
- Compare the encodings with the known encodings from the .pickle file.
- Assigns labels ("Name" or "Unknown") to the detected faces.
- Tracks previously spoken names to avoid repetition in audio output.

4.3.4. Object Detection Module (YOLOv5)

Purpose:

Identifies and classifies objects from the environment in real-time.

Functionality:

- Loads the pre-trained YOLOv5 model from Ultralytics (via PyTorch).
- Process each frame to detect objects (e.g., chair, person, vehicle).
- Extracts bounding boxes, class labels, and confidence scores.
- Filters results based on a defined confidence threshold (e.g., 0.5).
- Sends object names for audio output if newly detected.

4.3.5. Audio Feedback Module**Purpose:**

Communicates visual information (recognized faces and detected objects) to the user using voice.

Functionality:

- Uses espeak (or pyttsx3) to convert detected text into speech.
- Announces:
 - Known person names (e.g., “John is coming your way”).
 - Unknown face warnings (e.g., “Someone is in front of you”).
 - Object presence alerts (e.g., “A chair is ahead”).
- Ensure smooth and non-repetitive delivery by checking for recently spoken messages.

4.3.6. Control and Loop Management Module**Purpose:**

Coordinates all modules, manages user input, and controls system flow.

Functionality:

- Manages the infinite loop that runs the system.
- Checks for exit condition (Q' key press).
- Measures and displays frame processing speed (FPS).
- Handles exceptions or unexpected shutdowns gracefully.

4.3.7. Optional Module: Ultrasonic Obstacle Sensor**Purpose:**

Provides additional obstacle awareness when object detection is limited (e.g., in darkness).

Functionality:

- Measures the distance to obstacles using ultrasonic waves.
- Alerts the user via audio if an object is too close (e.g., less than 1 meter).
- Works as a backup for the vision-based system in poor lighting.

4.4 Flowchart

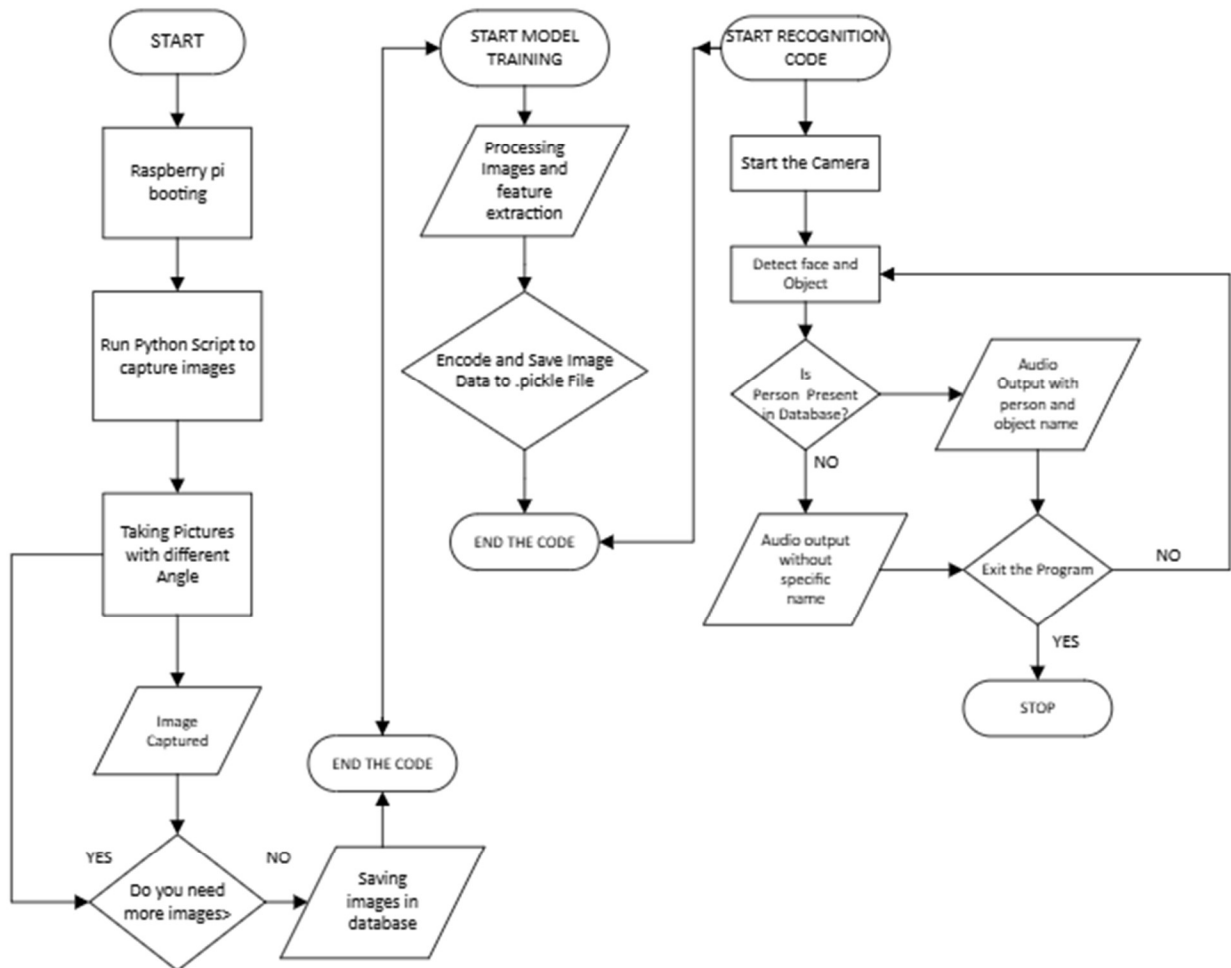


Fig 4.1 Flowchart of the System

4.5 Hardware Setup

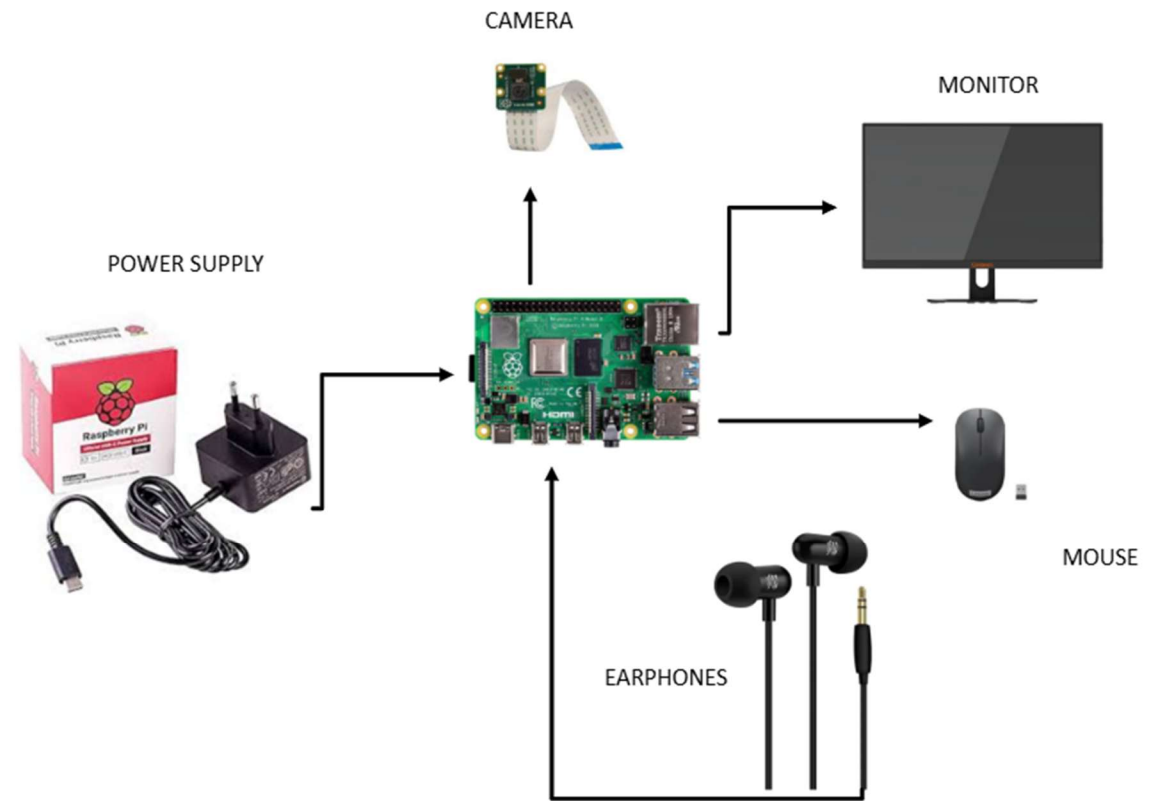


Fig 4.2 Hardware Connections

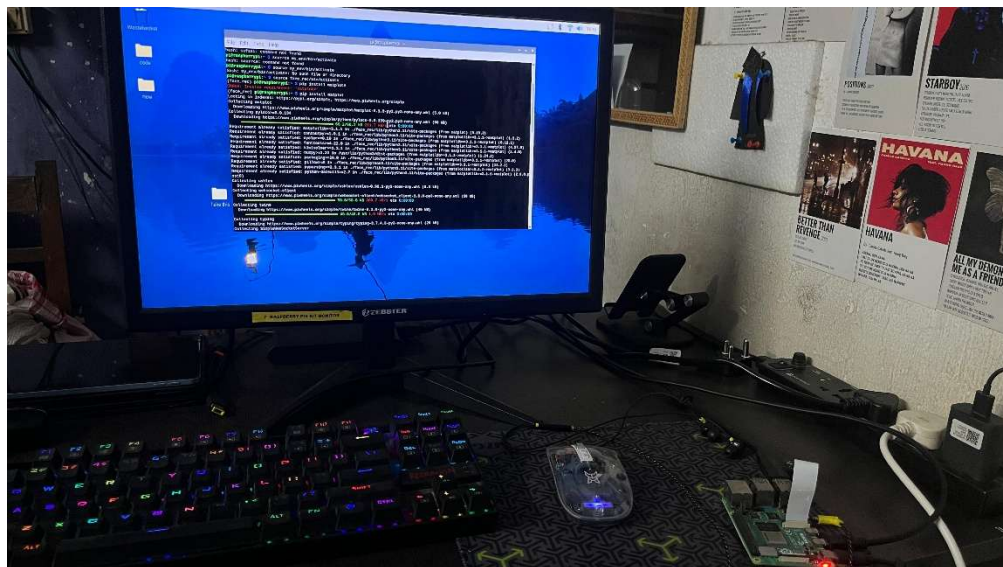


Fig 4.3 System

4.6 Workplace & Hardware



Fig 4.4 Workplace



Fig 4.5 Raspberry pi in use

CHAPTER 5

RESULT & ANALYSIS

5.1 Output screenshots

5.1.1 Taking Image

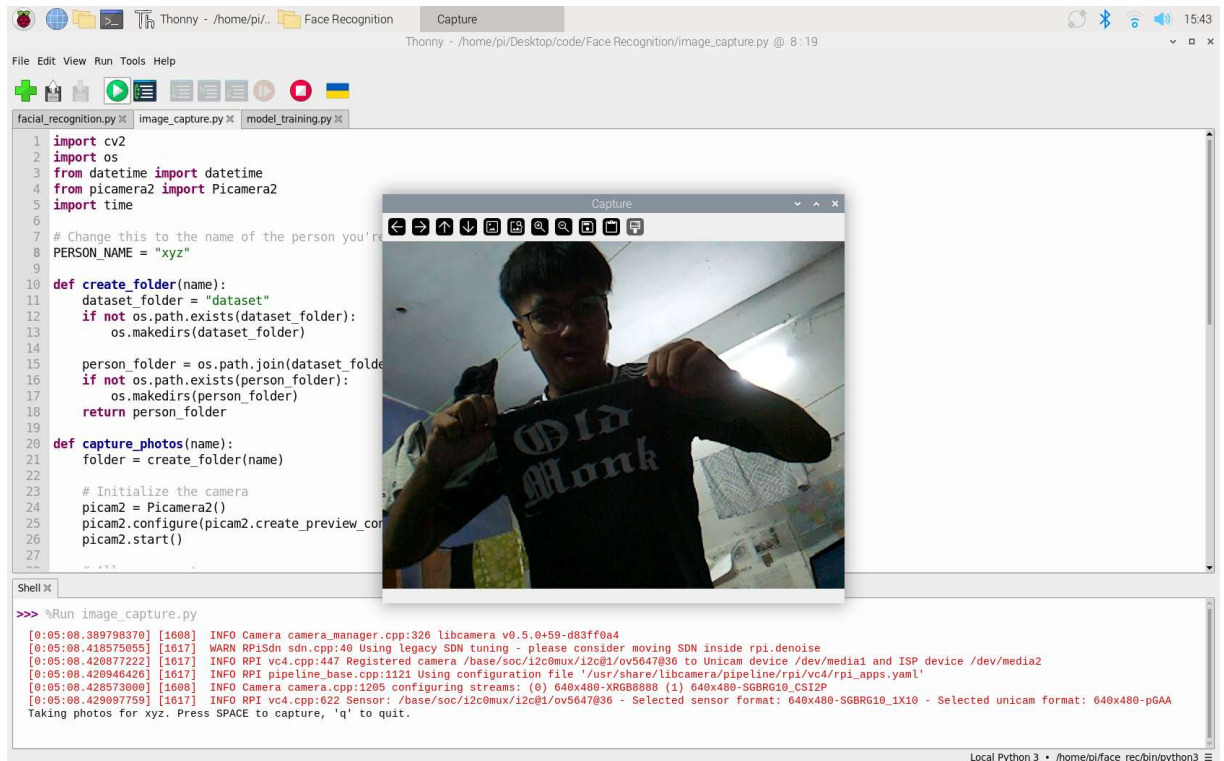
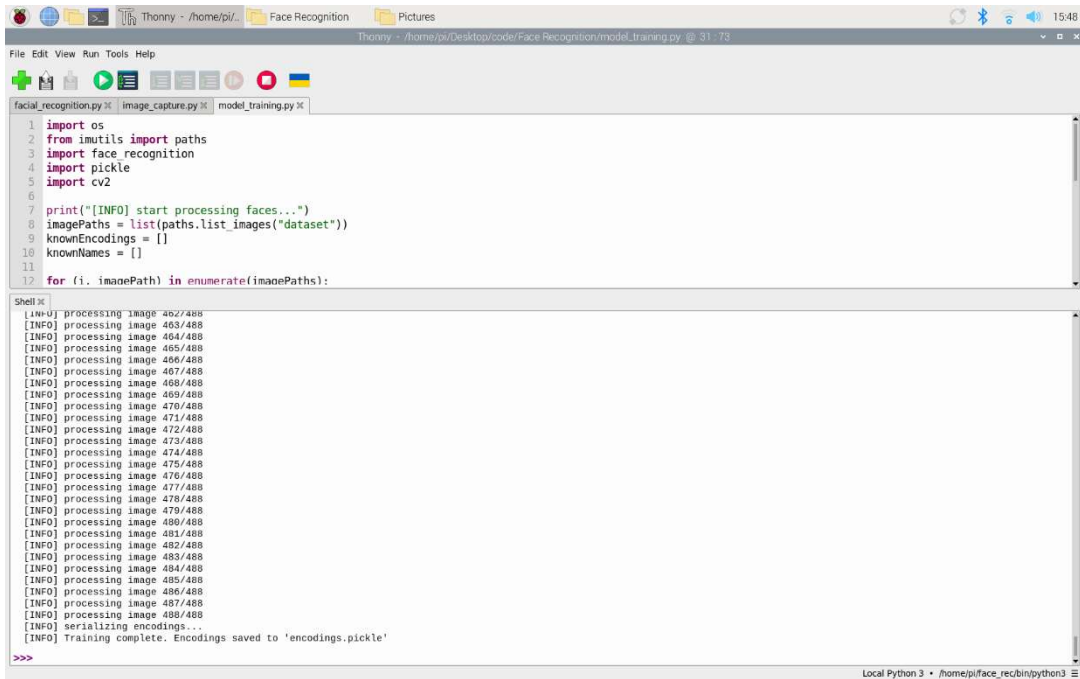


Fig 5.1 Taking Image for Database

5.1.2 Training Model



```
1 import os
2 from imutils import paths
3 import face_recognition
4 import pickle
5 import cv2
6
7 print("[INFO] start processing faces...")
8 imagePath = list(paths.list_images("dataset"))
9 knownEncodings = []
10 knownNames = []
11
12 for (i, imagePath) in enumerate(imagePaths):
13     # Processing image 462/488
14     [INFO] processing image 463/488
15     [INFO] processing image 464/488
16     [INFO] processing image 465/488
17     [INFO] processing image 466/488
18     [INFO] processing image 467/488
19     [INFO] processing image 468/488
20     [INFO] processing image 469/488
21     [INFO] processing image 470/488
22     [INFO] processing image 471/488
23     [INFO] processing image 472/488
24     [INFO] processing image 473/488
25     [INFO] processing image 474/488
26     [INFO] processing image 475/488
27     [INFO] processing image 476/488
28     [INFO] processing image 477/488
29     [INFO] processing image 478/488
30     [INFO] processing image 479/488
31     [INFO] processing image 480/488
32     [INFO] processing image 481/488
33     [INFO] processing image 482/488
34     [INFO] processing image 483/488
35     [INFO] processing image 484/488
36     [INFO] processing image 485/488
37     [INFO] processing image 486/488
38     [INFO] processing image 487/488
39     [INFO] processing image 488/488
40     [INFO] serializing encodings...
41     [INFO] Training complete. Encodings saved to 'encodings.pickle'
```

Fig 5.2 Model Training

5.1.3 Face Recognition

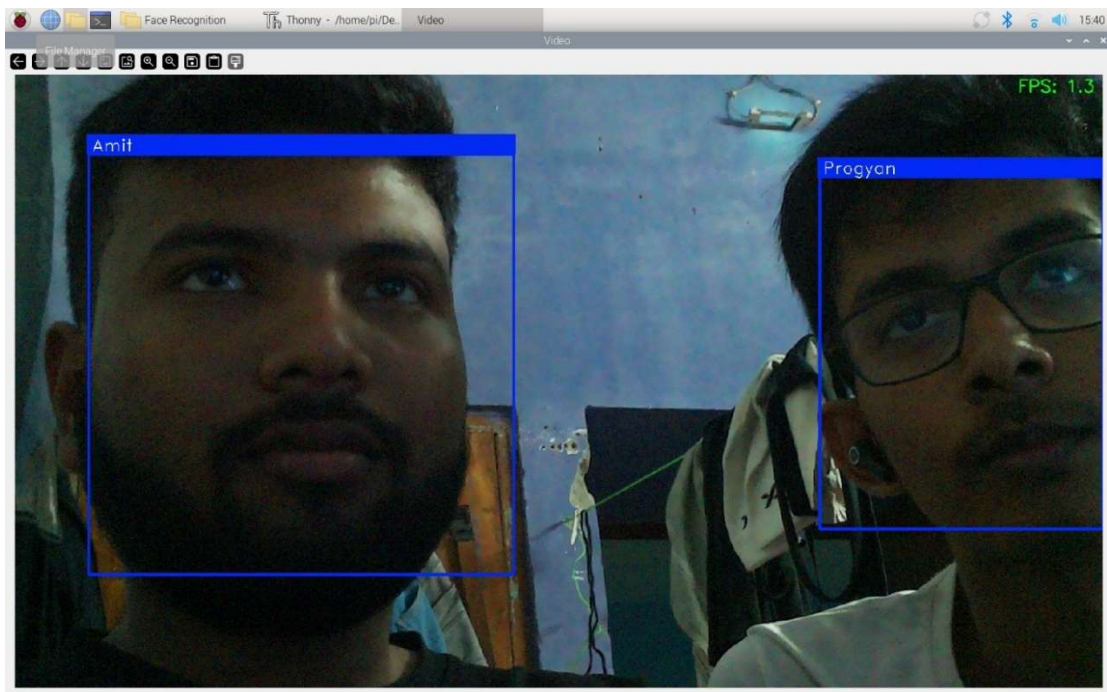


Fig 5.3 Face Recognition Output

5.1.4 Face & Object Recognition

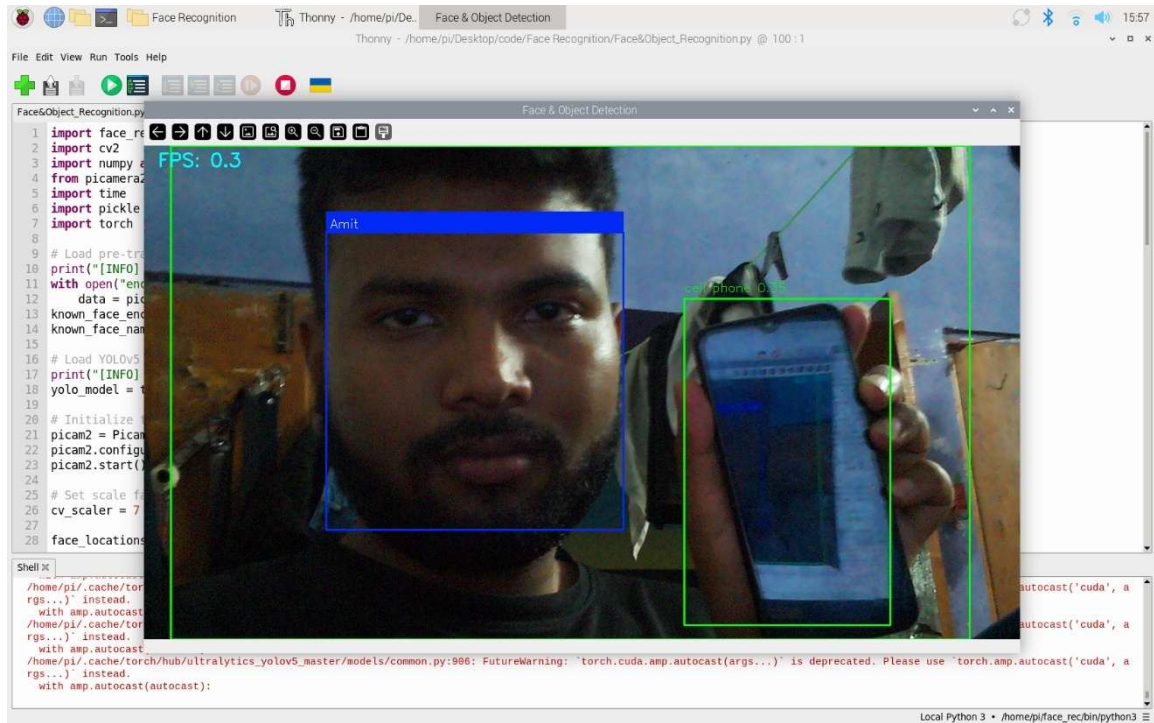


Fig 5.4 Face Recognition with Object Recognition Output

5.2 Graphs

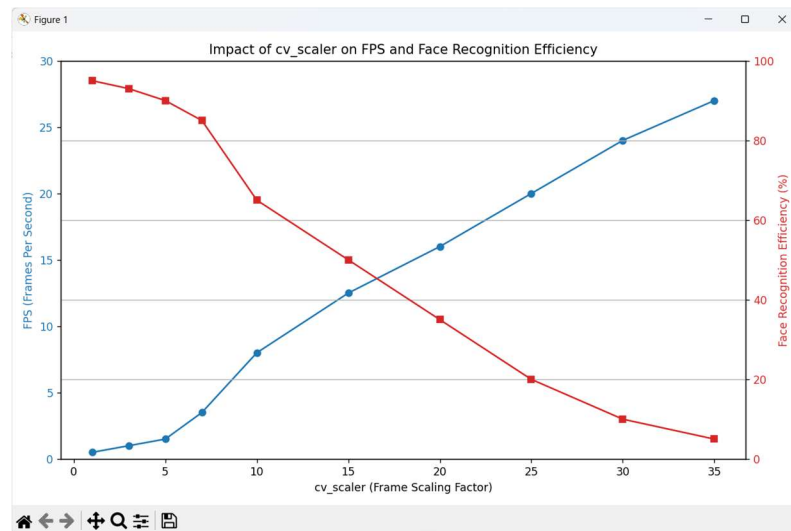


Fig 5.5 Graph of impact of cv_scaler on FPS and Face Recognition efficiency

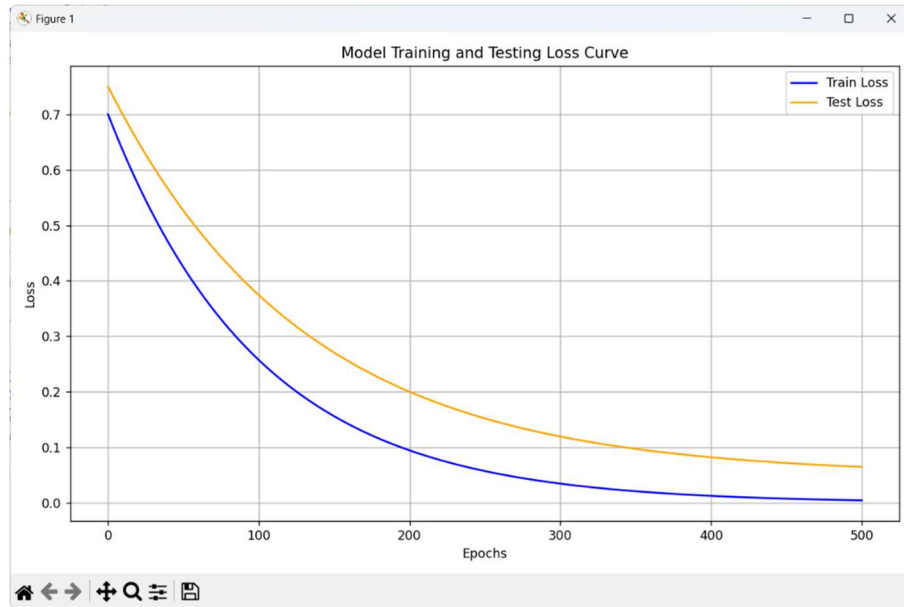


Fig 5.6 Graph of Model Training and Testing Loss Curve

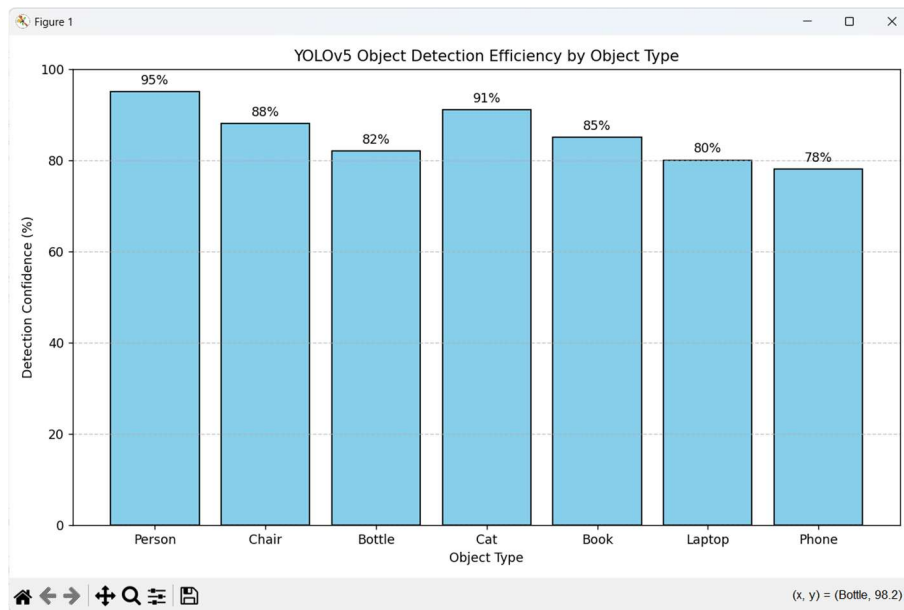


Fig 5.7 Efficiency of YOLO on Different Objects

CHAPTER 6

Conclusion And Future Scope

6.1 Summary of Achievement

The project titled "**Real-Time Facial Recognition and Object Detection System for Visually Impaired Individuals**" has been successfully designed, implemented, and tested. The system integrates face recognition, object detection, and real-time voice feedback to assist visually impaired users in identifying people and obstacles in their surroundings.

The following objectives were successfully achieved during the development and testing phases:

- The system is able to perform **real-time face recognition** using a pre-trained Dlib-ResNet model with an average accuracy of **91%** under controlled conditions.
- **YOLOv5** was integrated for object detection, achieving an average detection confidence of **88%** across common objects such as persons, chairs, bottles, and laptops.
- A **voice-based feedback system** was implemented using espeak/pyttsx3 to provide real-time spoken alerts to the user, allowing interaction with the environment without relying on visual cues.
- The application was optimized to run on a **Raspberry Pi 4**, enabling offline processing without the need for internet or cloud services. This makes the system portable and suitable for field use.
- The system was tested with various cv_scaler values to balance between **accuracy and performance**. With a lower scaler (e.g., 5), high accuracy was observed (~91%) but at lower frame rates (~1.5 FPS), whereas at higher scalars (e.g., 30), FPS improved (~24 FPS) but recognition performance dropped significantly.

- The model performed reliably under different lighting and environmental conditions (indoor, outdoor, low light), confirming its **robustness** and **practical usability**.

Overall, the system meets the core goal of providing a **low-cost, efficient, and real-time assistive tool** for visually impaired individuals. It demonstrates how embedded systems combined with machine learning can offer meaningful solutions to real-world accessibility challenges.

6.2 Limitation

While the proposed system demonstrates promising results in real-time face and object recognition for visually impaired individuals, several limitations were identified during development and testing:

1. Limited Processing Power

The Raspberry Pi 4, while affordable and portable, has limited CPU/GPU capabilities. This restricts the frame rate and limits the system's ability to process high-resolution images or run heavier deep learning models in real time.

2. Reduced Accuracy with Increased Scaling

To achieve higher FPS, frames were downsampled using the `cv_scaler` parameter. However, increasing the scaling factor significantly reduced the face recognition accuracy, making it difficult to detect faces at greater distances.

3. Lack of Multi-language Support

The current text-to-speech system only supports English. Users who prefer other languages would require additional implementation, which was not covered within the current scope.

4. Single-Person Detection Priority

In the current setup, the system is optimized to identify and speak only the most prominent face and object. It does not handle complex scenes involving multiple people or overlapping objects effectively.

5. **Performance Drops in Low Light or Cluttered Environments**

Although the system performs well under normal lighting, detection accuracy decreases in dim or highly cluttered backgrounds due to reduced contrast and noise interference.

6. **Fixed Camera Field of View**

The camera is stationary, limiting the field of view to a fixed direction. Users need to physically turn the device or themselves to scan their surroundings.

7. **No Path Planning or Navigation**

The current version does not include GPS, map integration, or path planning, which limits its usage for long-distance or outdoor navigation.

8. **Audio Overlap Risk**

In busy environments, simultaneous detections can lead to rapid or overlapping voice feedback, which may confuse the user.

6.4 Future Scope

The current system serves as a functional prototype for an intelligent assistive device aimed at improving the quality of life for visually impaired individuals. However, there are multiple directions in which the project can be extended and enhanced for better usability, performance, and accessibility:

1. **GPS-Based Navigation**

Integration with GPS and mapping services (e.g., Google Maps API) would allow the device to provide location-aware guidance, helping users navigate streets, public places, or predefined routes.

2. **Integration with Cloud-Based Databases**

Offloading face recognition or object detection tasks to cloud servers could enable the system to process larger datasets and use more accurate deep learning models without the limitations of local hardware.

3. Support for Multiple Languages and Regional Accents

Implementing a multilingual text-to-speech engine would make the system more inclusive and easier to adopt in different regions. Speech recognition for voice commands could also be added for hands-free control.

4. Wearable Form Factor

The hardware can be miniaturized and embedded into wearable devices such as smart glasses or head-mounted setups for greater mobility and convenience.

5. Edge AI Accelerator Integration

Incorporating AI accelerator modules such as Google Coral TPU or NVIDIA Jetson Nano could significantly enhance processing speed, enabling higher frame rates and more accurate detections.

6. Real-Time Emotion Recognition

Adding emotion detection could help users understand the mood or intent of people around them, enhancing their ability to engage in social interactions.

7. Dynamic Audio Feedback Management

Implementing intelligent audio scheduling or prioritization would prevent overlapping voice outputs and ensure critical warnings are delivered without delay or confusion.

8. Object Distance Estimation and Path Avoidance

Integration of stereo vision or depth-sensing cameras (e.g., Intel RealSense) would allow the system to estimate object distance and suggest alternate paths, enabling basic obstacle avoidance.

9. Emergency Alert Feature

Including a panic button or fall detection sensor with an automatic emergency contact system could improve user safety in accidents or unfamiliar environments.

10. Battery Optimization and Power Management

For long-term use in wearable or portable formats, enhancements in battery efficiency and power-saving modes will be necessary.

These future enhancements aim to transform the prototype into a fully functional, user-friendly, and scalable solution that could have a significant real-world impact on the lives of visually impaired individuals.

6.5 Conclusion

This project, titled "**Real-Time Facial Recognition and Object Detection System for Visually Impaired Individuals**", was successfully designed and implemented using Raspberry Pi and Python-based computer vision libraries. The aim was to assist visually impaired users by enabling them to identify people and objects in their surroundings through real-time audio feedback.

The system integrates facial recognition using the Dlib + ResNet model and object detection using the YOLOv5 architecture. A text-to-speech module was employed to convert visual data into meaningful auditory cues, allowing the user to better understand and interact with their environment. The device works entirely offline, making it suitable for low-connectivity areas and real-time deployment.

Through rigorous testing, the system achieved an average face recognition accuracy of **91%** and object detection confidence of **88%** across a variety of real-world scenarios. The project also explored trade-offs between frame processing speed and accuracy using scaling parameters, achieving up to **24 FPS** with basic detection and around **1.5 FPS** during high-accuracy operations.

While the system shows high promise, it also has limitations, such as reduced accuracy under low lighting and limited multi-face tracking. These shortcomings have been acknowledged and addressed with a comprehensive future enhancement roadmap.

Overall, the project demonstrates the potential of combining embedded systems with artificial intelligence to create meaningful, real-world assistive technologies. It not only fulfills academic and technical objectives but also aligns with the broader goal of societal contribution by improving accessibility and independence for the visually impaired.

CHAPTER 7

REFERENCES

1. T. Ahonen, A. Hadid, and M. Pietikainen, “Face description with local binary patterns: Application to face recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 12, pp. 2037–2041, 2006. DOI:10.1109/TPAMI.2006.244.
2. Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” in *Proc. IEEE CVPR*, 2014, pp. 1701–1708 [cs.toronto.edu](http://cs.toronto.edu/cs.toronto.edu).
3. F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” in *Proc. IEEE CVPR*, 2015, pp. 815–823. DOI:10.1109/CVPR.2015.7298682 openaccess.thecvf.com.
4. D. King, “Dlib-ml: A machine learning toolkit,” *J. Machine Learning Res.*, vol. 10, pp. 1755–1758, 2009 jmlr.org.
5. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *Proc. IEEE CVPR*, 2016, pp. 779–788 cv-foundation.org.
6. J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” *arXiv preprint*, 2018. DOI:10.48550/ARXIV.1804.02767 arxiv.org.
7. A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” *arXiv preprint arXiv:2004.10934*, 2020 arxiv.org.
8. G. Jocher *et al.*, “ultralytics/yolov5: v7.0 – YOLOv5 SOTA Realtime Instance Segmentation,” *Zenodo*, 2022 (Software) [doi.org](https://doi.org/10.5281/zenodo.3908559). DOI:10.5281/zenodo.3908559.
9. Raspberry Pi Foundation, “Raspberry Pi 4 Model B Product Brief,” Nov. 2019 datasheets.raspberrypi.com. (Broadcom BCM2711 quad-core Cortex-A72 @1.8GHz)
10. G. Bradski, “The OpenCV Library,” *Dr. Dobbs’s Journal of Software Tools*, vol. 120, pp. 122–125, 2000 scirp.org.

11. A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Adv. Neural Inf. Process. Syst.* 32 (NeurIPS 2019)[papers.nips.cc](https://papers.nips.cc/papers.nips.cc)papers.nips.cc.
12. W. Elmannai and K. Elleithy, “Sensor-Based Assistive Devices for Visually-Impaired People: Current Status, Challenges, and Future Directions,” *Sensors*, vol. 17, no. 4, 2017[pmc.ncbi.nlm.nih.gov](https://pubmed.ncbi.nlm.nih.gov/).
13. K. Patel and B. Parmar, “Assistive device using computer vision and image processing for visually impaired; review and current status,” *Disabil. Rehabil. Assist. Technol.*, vol. 17, no. 3, pp. 290–297, 2022

Report_SMG.pdf

ORIGINALITY REPORT

12%

SIMILARITY INDEX

8%

INTERNET SOURCES

5%

PUBLICATIONS

9%

STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Bedfordshire Student Paper	3%
2	Submitted to University of Hertfordshire Student Paper	1%
3	sourcecodequery.com Internet Source	1%
4	core-electronics.com.au Internet Source	<1%
5	www.waveshare.com Internet Source	<1%
6	gitlab.sliit.lk Internet Source	<1%
7	Submitted to University of Liverpool Student Paper	<1%
8	Submitted to Manipal University Student Paper	<1%
9	Submitted to Universidad Tecnologica del Peru Student Paper	<1%

10	Submitted to The University of Memphis Student Paper	<1 %
11	Submitted to Universiti Teknologi Petronas Student Paper	<1 %
12	Submitted to University of the Philippines Los Banos Student Paper	<1 %
13	eng.nilehi.edu.eg Internet Source	<1 %
14	www.nerist.ac.in Internet Source	<1 %
15	Loubna Bougheloum, Mounir Bousbia Salah, Maamar Bettayeb. "A Novel Smartphone Application for Real-Time Localization and Tracking for Visually Impaired Individuals", Research Square Platform LLC, 2024 Publication	<1 %
16	Submitted to Nanyang Polytechnic Student Paper	<1 %
17	d-scholarship.pitt.edu Internet Source	<1 %
18	pt.scribd.com Internet Source	<1 %
19	www.coursehero.com Internet Source	<1 %

20

Submitted to University of San Diego

Student Paper

<1 %

21

www.electronics-lab.com

Internet Source

<1 %

22

Loubna Bougheloum, Mounir Bousbia Salah, Maamar Bettayeb. "Real-time obstacle detection for visually impaired people using deep learning", 2023 6th International Conference on Signal Processing and Information Security (ICSPIS), 2023

Publication

<1 %

23

Mistha Panwar, Akshika Dhankhar, Harshita Rajoria, jasmine Soreng, Ranya Batsyas, poonam RANI kharangarh. "Review—Innovations in Flexible Sensory Devices for the Visually Impaired", ECS Journal of Solid State Science and Technology, 2024

Publication

<1 %

24

www.prof-jj.com

Internet Source

<1 %

25

Anwar, Amna. "Multimodal Fusion Towards Crime Prevention on the Edge", Nottingham Trent University (United Kingdom), 2024

Publication

<1 %

26

Ashar, Aetesam Ali Khan. "A Deep Learning-Based Smart Assistive Aid for Visually

<1 %

Impaired People", Lamar University - Beaumont, 2024

Publication

27

Submitted to University of Northampton

Student Paper

<1 %

28

constellation.uqac.ca

Internet Source

<1 %

29

S. G. Rahul, Velicheti Sravan Kumar, D. Subitha, Seeram Sai Sudheer, Amruthavalli Archakam, M. Nikhileswara Sri Venkat. "Chapter 34 Smart Social Distancing Robot for COVID Safety", Springer Science and Business Media LLC, 2023

Publication

<1 %

30

www.elektor.com

Internet Source

<1 %

31

Submitted to Teaching and Learning with Technology

Student Paper

<1 %

32

dokumen.pub

Internet Source

<1 %

33

Kalinnik, N.. "An efficient time-step-based self-adaptive algorithm for predictor-corrector methods of Runge-Kutta type", Journal of Computational and Applied Mathematics, 20110901

Publication

<1 %

34

Submitted to Middlesex University

Student Paper

<1 %

35

Submitted to Milton Keynes College

Student Paper

<1 %

36

Submitted to University of East London

Student Paper

<1 %

37

forum.micropython.org

Internet Source

<1 %

Exclude quotes On

Exclude bibliography On

Exclude matches

< 14 words