DESIGN AND IMPLEMENTATION OF A PORTABLE MACHINE VISION SYSTEM FOR REAL-TIME OBJECT DETECTION AND AUDITORY FEEDBACK

by

Themba Mthembane Sivate
(21207406)

Submitted in fulfilment of the requirements for the degree of:

MASTER OF ENGINEERING
in the
DEPARTMENT OF ELECTRONIC AND COMPUTER ENGINEERING,

FACULTY OF ENGINEERING AND THE BUILT ENVIRONMENT

at the

DURBAN UNIVERSITY OF TECHNOLOGY

November 2024

Supervisor: Prof. N. Pillay
Co-Supervisor: Dr. N. Singh

PREFACE

I, the undersigned, Mr Themba Mthembane Sivate, declare that the work embodied in this dissertation, titled “Design and Implementation of a Portable Machine Vision System for Real-Time Object Detection and Auditory Feedback.”, forms my contribution to the research work carried out under the guidance of Prof N. Pillay and Dr N. Singh at the Durban University of Technology. I hereby declare that all materials presented in this dissertation are my own work, except where in-text citations and references have been used to acknowledge the contributions of others. Furthermore, no part of this work has been submitted, either in whole or in part, for the award of a degree at any other university.

Themba Mthembane Sivate

Durban University of Technology

November 2024

DECLARATION 1: SUPERVISOR

According to the contents of this thesis, as the candidates' Supervisor, I agree to the submission of the thesis.

_____	_____
Prof. N. Pillay	Dr. N. Singh
(Main supervisor)	(Co-supervisor)
Date: 20/2/2025	Date: 20 February 2025

DECLARATION 2: PLAGIARISM

I, **Themba Mthembane Sivate (21207406)**, hereby declare that the research presented in this dissertation is my own original work, except where explicit reference has been made to the work of others. I further declare that the research presented in this dissertation has not been submitted to any other university in partial or full fulfillment of a master's degree or any other form of academic qualification.

Themba Mthembane Sivate

November 2024

DECLARATION 3: PUBLICATIONS

I, **Themba Mthembane Sivate (21207406)** declare that the following publication came out of this dissertation.

1. T. M. Sivate, N. Pillay, K. Moorgas and N. Singh, "Autonomous Classification and Spatial Location of Objects from Stereoscopic Image Sequences for the Visually Impaired," 2022 International Conference on Electrical, Computer and Energy Technologies (ICECET), Prague, Czech Republic, 2022, pp. 1-6, doi: 10.1109/ICECET55527.2022.9872538.

ACKNOWLEDGEMENTS

Throughout the writing of this dissertation, I have received a great deal of support and assistance. I would first like to thank my supervisor, Prof N. Pillay and Dr. N. Singh for helping, guiding, and supporting me in this course from the beginning to the end. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

ABSTRACT

A primary challenge faced by individuals with visual impairments is the difficulty or inability to perform object identification. While conventional aids such as magnifying spectacles may assist with near-field vision, individuals with reduced or absent sight often rely extensively on tactile exploration for object recognition. This reliance presents significant navigational challenges, particularly in dynamic environments such as roadways, where the increased risk of accidents is a major concern. To address this, this research proposes the development of a portable machine vision system designed to provide real-time auditory feedback regarding detected proximal objects, thereby assisting individuals with visual impairments in navigation. The design of the system prioritizes portability, reliability, modularity, and unobtrusiveness during typical operation.

The hardware implementation of the proposed system consists of three key elements: a Single Board Computer (SBC), a wireless camera, and a Bluetooth-enabled earpiece. The experimental results demonstrate that the proposed system is capable of delivering real-time audio feedback of detected objects to visually impaired individuals. The system was evaluated in a real-life environment in both acceptable and poor lighting conditions. The efficacy of the proposed system in well-lit environments resulted in an average detection rate of 87.65%. However, in low-light scenes an average detection rate of 51% was observed because of the low image resolution. The minimum observed delay was 4.9 seconds, while the maximum was 10.2 seconds. This latency encompasses the duration required for image capture, processing, and audio translation. The latency can be mitigated by incorporating an integrated circuit with a dedicated Graphical Processing Unit (GPU), which is more proficient in handling machine learning and video processing tasks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

AI	Artificial Intelligence
AES	Advanced Encryption Standard
ANN	Artificial Neural Network
API	Application Programming Interface
ARM	Advanced RISC Machines
ARQ	Automatic Repeat Request
AVDTP	Audio/Video Distribution Transport Protocol
CA	Collision Avoidance
CISC	Complex Instruction Set Computer
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
CTS	Clear to Send
DB	Database
DTW	Dynamic Time Warping
FHSS	Frequency Hopping Spread Spectrum
GPU	Graphics Processing Unit
HMM	Hidden Markov Models
LPC	Linear Predictive Coding
mAP	mean Average Precision
MFCC	Mel-Frequency Cepstral Coefficients
OpenCV	Open-Source Computer Vision Library
R-CNN	Region-based Convolutional Neural Network
ResNet-101	Residual Network with 101 layers
RISC	Reduced Instruction Set Computer
RTS	Request to Send
SBC	Single Board Computers
TDD	Time-Division Duplex

TTS	Text to Speech
USB	Universal Serial Bus
VI	Visually Impaired
VC	Visually Challenged
WIFI	Wireless Fidelity
WPA	Wi-Fi Protected Access
YOLO	You Only Look Once
L_{cls}	Classification loss
L_{box}	Bounding box regression loss
L_{mask}	Mask loss
a_i	LPC coefficients
y_i	True label
$\hat{y}_i$	Predicted probability
θ	Model parameters
η	Learning rate
$\nabla L(\theta)$	Gradient of the loss function
$s(n)$	Speech signal
f_0	Initial frequency
Δf	Frequency step
T_h	Hopping period
P_e	Probability of a packet error
F_k	Formant frequency
t_e	Certain time
$e(n)$	Prediction error
B_k	Bandwidth
$R_k(z)$	Resonator
M_j	Final mask

CHAPTER 1 Introduction

1.1 Introduction

Globally, a significant proportion of the population, estimated to be at least 2.2 billion individuals, experiences near or distant vision impairment. This condition has been shown to negatively correlate with workplace participation and is associated with an elevated risk of developing depression and anxiety [1]. Visual deficits in older adults can contribute significantly to social isolation, diminished walking ability, an elevated risk of falls and fractures, and a greater tendency towards earlier institutionalization in nursing or care facilities [1]. During leisure activities, visually challenged individuals may encounter various challenges associated with visual perception. These challenges include difficulties in commuting, reading texts in standard fonts, observing nonverbal instructions from a distance, and spatial orientation issues due to inadequate lighting [2]. Additionally, they may experience slower reaction times to non-verbal signals.

Visually impaired (VI) persons often encounter significant challenges in performing accurate object identification tasks. They mostly rely on corrective glasses and walking sticks to help them perform their daily tasks. In this study, a pragmatic solution is proposed to help the VI individual identify objects in front of them. The proposed solution comprises of a computer vision system for real-time object detection and audio feedback. The system is modular in nature thereby allowing for portability. It uses common wireless protocols (WI-FI and Bluetooth) to eliminate any physical connection between the sub-components of the system. The proposed hardware framework is designed to be simple to use, portable, modular, and reliable.

1.2 Problem statement

VI individuals face significant challenges in identifying objects within their environment, often relying on physical touch and magnifying devices, which can be impractical or unsafe in certain situations, such as navigating roads or unfamiliar locations. This reliance on physical touch, especially in public spaces, increases the risk of accidents and limits their independence. Current solutions that assist visually impaired individuals with object identification tend to be bulky or fail to provide real-time feedback in dynamic

environments. Moreover, these systems often struggle in varying lighting conditions, which further hampers their effectiveness.

This research addresses the problem of developing a portable, reliable, and unobtrusive machine vision system that provides real-time auditory feedback to VI individuals. The proposed solution aims to enhance object identification and environmental awareness without requiring physical contact. Additionally, the system must be capable of maintaining a high detection accuracy across different lighting conditions, a challenge often faced by existing technologies. The proposed system will leverage advanced object detection algorithms and speech synthesis to deliver real-time audio descriptions of nearby objects, empowering VI individuals to navigate their surroundings with greater independence and safety.

1.3 Field of research

The field of study is computer vision and machine learning with a focus on object identification and text-to-speech synthesis. Work has been conducted previously to address some of the problems faced by the VI community. Namely, Khlaikhayai et al. [3] proposed and implemented an intelligent walking stick for the elderly and blind safety protection. This resulted in a system that operated only for nearby object detection. Ayat et al. [4] and Amira et al. [5] proposed a smart walking stick for the blind to help them navigate common environments. The result was a walking stick that used ultrasonic sensors to detect objects and send audio feedback via wired earphones. The solution succeeded with the major shortcoming of limited distance range.

Sirouspour et al. [6] suggested an analysis and implementation of a walking support system for VI people which was limited by the short distance and the maximum number of objects it could detect. Perera et al. [7] proposed an assistive system for VI individuals to recognize pure colours on objects. However, the system was limited in the number of colours it could detect and required the use of a smartphone device to convert the identified colours to audio feedback. Krishna et al. [8] proposed and implemented a vision system with 3D audio feedback to assist navigation for the VI. This resulted in a headset battery-powered computer

vision system coupled with wired earphones which introduced weight and restricted movement for the end user.

1.4 Research questions

The research questions are as follows:

i. Can a portable modular-based computer vision system be developed to detect and identify objects?
ii. How reliable will it be in object detection and classification?
iii. Is real-time auditory feedback of detected objects possible with the proposed hardware framework?
iv. How long will the portable system function given the specified battery hardware?

1.5 Objectives of the study

This study seeks to develop a modular and portable computer vision system to help the VI community. To achieve this, the following objectives are:

i. Develop software that automatically identifies objects and translates them into auditory feedback for informed decision-making.
ii. To utilise hardware components that enable portability, non-intrusive in nature and simple to operate.
iii. Analyse the efficacy of the system in terms of its accuracy, processing speed, performance under different lighting conditions and cost effectiveness.

1.6 Justification of the study

VI individuals face daily challenges in identifying objects within their surroundings, which significantly affects their mobility and independence. Traditional methods of object identification, such as the use of magnifying glasses or physical touch, are often inadequate, especially in dynamic environments like roads or public spaces, where the risk of accidents is elevated due to delayed or inaccurate identification. These limitations highlight the urgent need for more advanced and user-friendly solutions that can assist VI persons in real-time,

reducing their dependence on touch and enhancing their ability to navigate safely and independently.

This study is justified by the pressing demand for a technological solution that bridges the gap between visual impairment and environmental awareness. The proposed research seeks to address this challenge by developing a portable machine vision system capable of providing real-time auditory feedback on detected nearby objects. The system, composed of a wireless camera, a Single Board Computer (SBC), and a Bluetooth earpiece, is designed to be modular, reliable, and unobtrusive, making it an accessible tool for visually impaired individuals in their daily lives.

Moreover, the proposed system employs advanced object detection algorithms and speech synthesis technology to provide users with precise and timely information about their surroundings. Given its portability and real-time processing capability, this system has the potential to significantly improve the quality of life for visually impaired individuals by reducing the risks associated with navigation in unfamiliar or unsafe environments.

1.7 Significance of the study

This study is significant for several reasons, particularly in advancing assistive technology for VI individuals. The development of a portable machine vision system that provides real-time auditory feedback on nearby objects addresses a critical gap in the current assistive solutions available for visually impaired persons. By doing so, the research has the potential to significantly improve the quality of life for these individuals, enhancing their independence, safety, and ability to navigate complex environments. The study will focus on the following advancements:

i. The proposed system offers a reliable, unobtrusive, and user-friendly solution for VI individuals, helping them to identify objects without relying on physical touch or external assistance. This increased independence in everyday activities can lead to greater confidence and improved mobility, contributing to an enhanced sense of autonomy and self-reliance.

ii. The integration of real-time object detection and auditory feedback enables VI users to navigate dynamic environments, such as roads or public spaces, more safely. The system can potentially reduce the risk of accidents by alerting users to obstacles or hazards in their path, making it a valuable tool for improving safety and preventing injuries.

iii. By employing advanced technologies such as YOLACT for object detection and the Festival Speech Synthesis system for auditory feedback, this research demonstrates the feasibility of using machine vision and Artificial Intelligence (AI) driven systems to support accessibility. The study contributes to the growing field of assistive technology and machine vision by providing a new approach to addressing the challenges faced by VI individuals.

iv. The modular design of the proposed system, incorporating readily available components such as a wireless camera, SBC, and Bluetooth earpiece, ensures that it is both scalable and cost-effective. This makes the technology accessible to a broader population of VI individuals, particularly in low-resource settings where more expensive alternatives may not be viable.

The results of the study, particularly the system's performance in varying lighting conditions, and auditory latency highlight areas for further optimization and refinement. By laying the groundwork for continued research and development in this area, this study opens new possibilities for enhancing the functionality of portable vision systems, particularly in challenging environments.

1.8 System constraints

The constraints of the proposed portable machine vision system for VI individuals, and particularly factors that affect its design, performance, and usability, are the following:

i. The system's performance is highly dependent on ambient lighting. As the experimental results indicate, object detection accuracy drops significantly in low-light environments. This constraint limits the system's effectiveness in poorly lit areas or at night.

ii. While the SBC used in the system offers real-time processing capabilities, the system's performance may be constrained by its processing power when handling complex scenes with multiple objects. Higher latency or slower response times could occur, affecting real-time feedback.

iii. The system's portability relies on battery-powered components (wireless camera, SBC, Bluetooth earpiece), meaning battery life is a crucial constraint. Continuous use over long periods might drain the battery quickly, limiting its operational time before requiring a recharge.

iv. The system is designed to be portable and unobtrusive, but the inclusion of multiple hardware components, such as the SBC, wireless camera, and Bluetooth earpiece, poses size and weight constraints. Ensuring that these components are compact and lightweight while maintaining functionality is a challenge.

v. The system relies on wireless communication between the camera, SBC, and Bluetooth earpiece. Any disruptions or interference in wireless connectivity, such as poor signal strength or crowded wireless environments, may impact the reliability of data transmission and audio feedback.

vi. The accuracy of the object detection algorithm (YOLACT) is a constraint, especially in dynamic, cluttered, or rapidly changing environments. False positives or negatives, as well as the system's ability to distinguish between objects that are visually similar, could affect the quality of the feedback provided to the user.

vii. The system may be constrained by weather conditions, such as rain, fog, or dust, which could obscure the camera's view and impact the system's ability to accurately detect and identify objects. Harsh environmental conditions might also affect the durability of the hardware components.

viii. The clarity, volume, and timing of the auditory feedback are constrained by the Bluetooth earpiece and the Festival Speech Synthesis system. Background noise, poor-quality audio, or delayed feedback could hinder the user's ability to interpret the information accurately, especially in noisy or crowded environments.

ix. The system must remain unobtrusive during normal operation, but the integration of multiple hardware components can affect the user's comfort. Ensuring that the system is easy to carry, wear, or use without causing discomfort over long periods is a design constraint.

x. To make the system accessible to a wide range of visually impaired individuals, cost is a key constraint. The components used (SBC, camera, Bluetooth earpiece) must be affordable, limiting the choice of high-end or more powerful hardware that could enhance performance but increase the overall cost.

1.9 Structure of the dissertation

This dissertation is structured as follows:

Chapter 1 discusses the background of the study, field of research, research questions, problem statement, objectives of the study, justification of the study, significance of the study, research methods, and finally the structure of the thesis.

Chapter 2 reviews current literature on computer vision, object detection, and identification to assist the VI. This chapter also outlines the gaps and limitations of the existing studies in this field.

Chapter 3 offers an exhaustive examination of the various technologies integrated into the proposed system. Furthermore, it delineates the overarching architectural framework that underpins the system's design.

Chapter 4 presents a meticulous development of both the hardware and software components of the proposed system. Furthermore, it provides a detailed description of the implementation and utilization of the previously discussed algorithms, specifically focusing on object detection and text-to-speech functionalities.

Chapter 5 delineates the steps taken for end-to-end testing of the system, as well as presents the results of the conducted experiments.

Chapter 6 discusses the results obtained after testing and evaluating the performance of the system.

Chapter 7 describes the research challenges and limitations of the proposed system and is followed by the recommendations for future work.

CHAPTER 2 Literature Review

2.1 Introduction

This chapter provides an overview of the current literature on computer vision, object detection, and identification that aims to help VI individuals. The chapter also highlights the gaps and limitations of the existing studies in this research area.

In the development of object detection systems, the initial requirement is to have an input source, typically images or videos, which can be stored locally or accessed via live streams. For instance, screen recording software can generate video files that capture computer activity, while screenshots can produce still images. Most computer vision systems depend on recent image or video data from a camera, making camera access a critical component. OpenCV is a widely used library for camera access and face detection in computer vision applications [9].

OpenCV is a versatile software library designed for computer vision and machine learning, providing a common infrastructure that facilitates machine perception in commercial products. Sivkov et al. [10] developed an algorithm utilizing OpenCV for accessing a camera in a custom electronic device, enabling video streaming and image capture. Their objective was to capture an image of a meter box and extract its reading using character recognition. The image is captured from a video camera over a predetermined period of time and is downloaded for further processing in the memory of the program [10]. Several OpenCV functions were employed to highlight the region of interest, convert the image to grayscale, and perform morphological operations essential for enhancing the targeted area. This demonstrates that OpenCV provides numerous built-in functions for image manipulation and feature extraction, while remaining extensible to accommodate specific project requirements. Nonetheless, it remains extensible to accommodate specific project requirements.

OpenCV can be compiled from source or obtained as a pre-built binary package, supporting multiple programming languages and offering extensibility for various applications. Yu et al. [12] extended OpenCV's functionality through the 'Ch' (C/C++) programming environment, which allows users to write cross-platform code efficiently. While C++ is a compiled programming language, Ch serves as a cross-platform interpreter and scripting language for C and C++. This allows for modifications to the script and its integration into a compiled C++ program without necessitating recompilation. This capability is advantageous as it enables the embedding of Ch scripts within an OpenCV C++ application, thereby facilitating dynamic changes during runtime. While OpenCV is globally recognized in the computer vision and machine learning industries, this project specifically uses OpenCV for camera access and image capture, with the actual object detection performed using the 'You Only Look at Coefficients' (YOLACT) algorithm, rather than relying on OpenCV for this function.

2.2 A review of related literature

The VI people may experience the problem of being unable to identify objects which appear in front of them. This creates difficulty when moving around or navigating places. Past work has been done in this research field to solve some of the problems faced by the VI community. In the subsequent sections, related studies are described.

Ayat et al. [4] proposed an effective fast response smart stick for blind people. The solution employed the use of an ultrasonic sensor to detect objects and used an infrared sensor to detect start-cases and the use of a vibration motor as a speech warning system. Figure 2.1 shows the design of the smart stick detects obstacles in front of the blind. In this case, a PIC Microcontroller 18F46K80 was used as a processing unit and a device ISD1932 for audio recording and playback. This system uses pre-recorded speech messages for conveying any detection of obstacle [4].

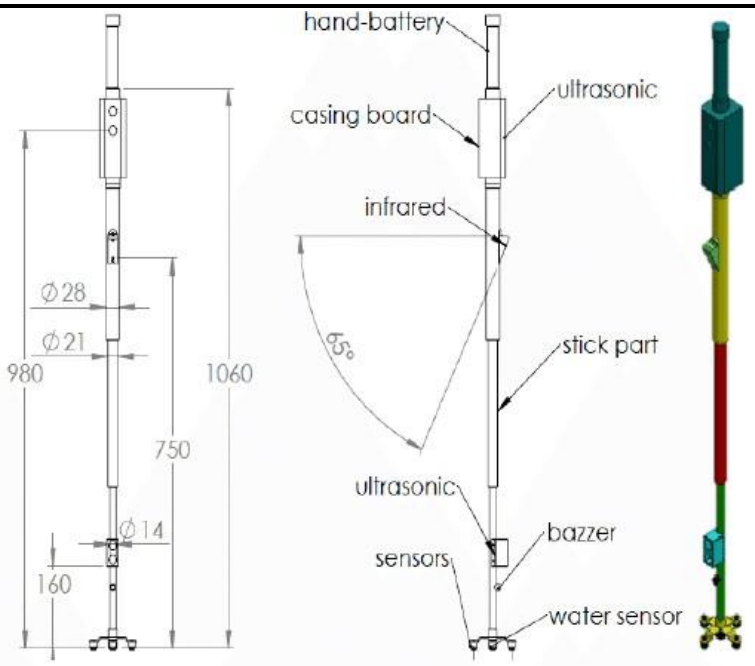


Figure 2.1. Design of the Smart stick for obstacle detection in front of the blind [4].

The product is a portable smart stick that can detect objects within 4 meters. However, this solution has limitations in that it cannot classify or identify objects. Additionally, the user must hold the stick in their hand, which restricts the user from operating with both hands while using the system.

Sirouspour et al. [6] proposed the analysis and implementation of a walking support system for VI people. The system was equipped with four ultrasonic sensors and an infrared sensor. The system was also equipped with a microcontroller, servo motor, and a buzzer to generate outputs that inform the user about the type of obstacle ahead [6]. The final product was a waist-wearable system capable of sensing nearby objects. It was tested and discovered to be functional with limitations. It was claimed that the system is affordable, lightweight, and consumes less power. The terrain analysis feature required the end user to carry a laptop that performs the analysis, which is not very practical for a VI person to carry. This device is limited to the standard pace of mobility and cannot differentiate between animate and inanimate obstacles [6]. Figure 2.2 shows (a) photography of a slanted overhead obstruction, (b) photo of an overhead obstruction out of a wall, and (c) sensor data showing detection of overhead obstruction. Further research was recommended to overcome the identified

deficiencies to improve mobility of the blind people. Ultrasonic sensors are known for their short range of operation and cannot identify the detected object.

It makes sense why a buzzer was used instead of a speaker or earphone. Advantages of the system included cost effectiveness and low power consumption.

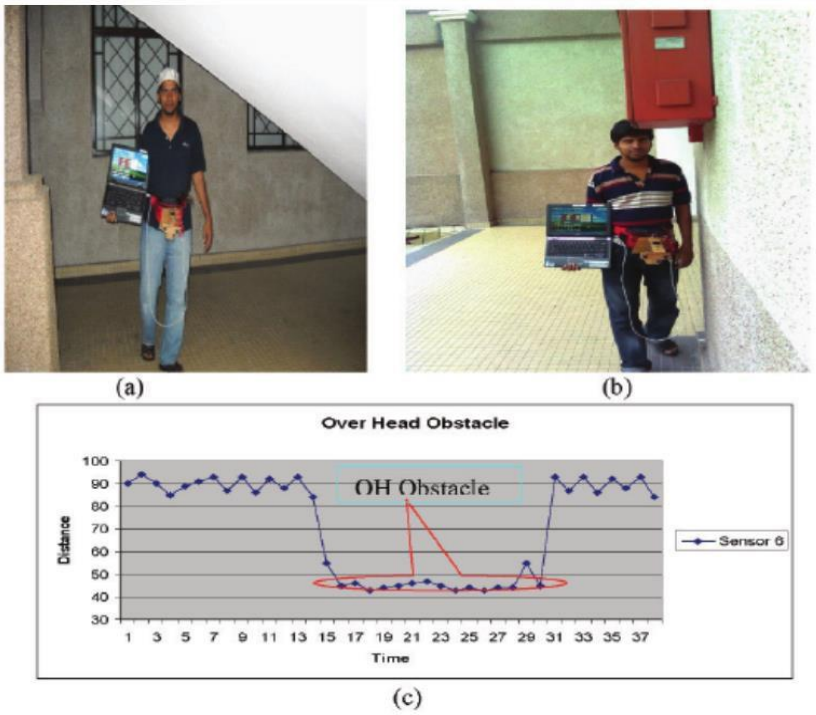


Figure 2.2. (a) Photography of a slanted overhead obstruction; (b) Photo of an overhead obstruction out of a wall; (c) Sensor data showing detection of overhead obstruction [6].

Perera et al. [7] developed an assistive system enabling visually impaired individuals to recognize distinct colours. The design employed an Arduino development board integrated with a TCS230 colour sensor, trained to identify twelve colours. To enhance functionality, several low-cost yet high-performance components were incorporated, including the SR04 ultrasonic sensor, a light-dependent resistor (LDR), and a Bluetooth module. An Android-powered smartphone served as the audio output device, providing voice feedback. The system was evaluated in real-time with both blindfolded participants and visually impaired users across diverse indoor and outdoor environments. The final prototype featured an Android application that converted detected colours into corresponding voice prompts. Although the system demonstrates cost-effectiveness and innovation, its limitations include restricted colour recognition, sensitivity to environmental conditions, dependence on smartphone integration, and a limited scope of functionality.

Krishna et al. [8] proposed a vision system that provides 3D audio feedback to support navigation for visually impaired users. The design employed Raspberry Pi units with cameras to capture images, which were transmitted to a remote server for processing. Using MobileNet, the server performed object detection and generated corresponding audio vectors, delivered to the user through earphones.

Following image processing, the server transmitted audio signals to the Raspberry Pi units for playback. The final prototype was designed as a head-mounted apparatus secured with straps. According to Krishna et al. [8], the system proved effective for navigation when tested with visually impaired participants in real-life environments. Nonetheless, reliance on a remote server introduced a single point of failure, while the overall weight of the device was reported to be excessive for comfortable head-mounted use. To improve portability, reducing battery size was suggested. Additionally, the use of two Raspberry Pi development boards raised concerns regarding high power consumption, which may limit the practicality of the design.

Mallikarjuna, Hajare, and Pavan [13] proposed a cognitive Internet of Things (IoT) system to support independent navigation for visually impaired individuals. The system employs the Google TensorFlow framework to classify real-world objects, thereby enhancing user awareness of the surrounding environment. Interaction is facilitated through audio messages, offering a cost-effective means of guidance. The framework enables object recognition and self-navigation in complex scenarios such as moving through crowds, train stations, and bus stations, as well as detecting both stationary and moving vehicles from a distance. Figure 2.3 shows the block diagram of this system.

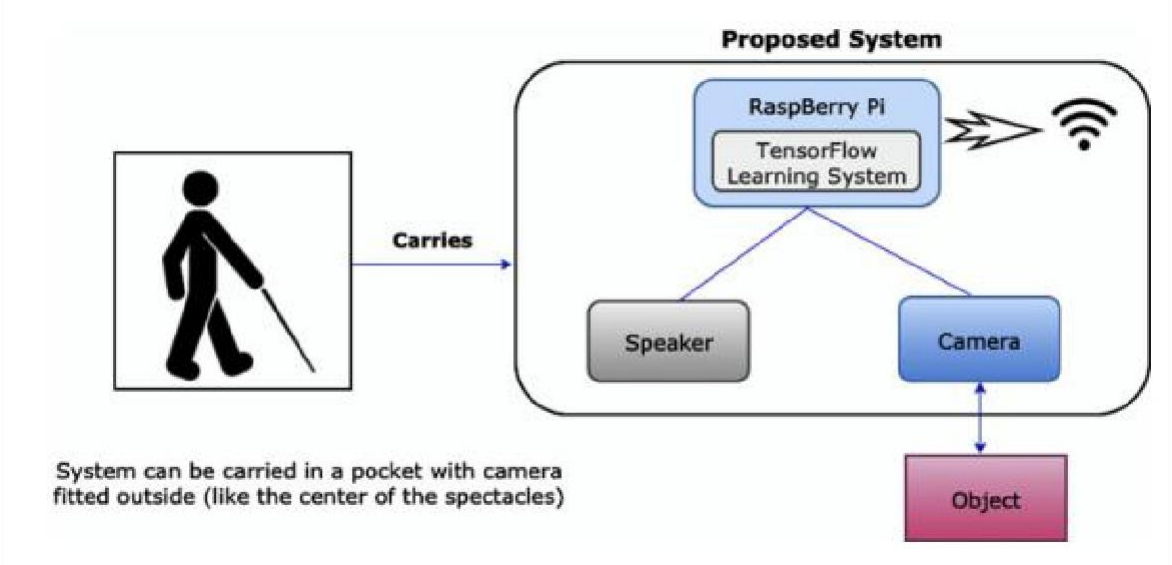


Figure 2.3. Block diagram of the Raspberry Pi based system [13].

The system comprised of a Raspberry Pi 3 and a Raspberry Pi Camera and employed a speaker instead of wired earphones. It was designed to be portable, with the camera mounted externally with the main unit meant to be carried in the user's pocket [13]. Based on the system specifications, it can be surmised that the system is small enough to fit in a pocket but required wired speakers and a camera. The power consumption of the speaker was not provided and the number objects the system can detect is limited.

Kanwal et al. [14] proposed a navigation system for the VI individual using Microsoft Kinect hardware developed for the XboxR, that functions at a rate of 8 to 10 frames per second, utilizing an Intel Core 2 Quad 2.83 GHz processor. The system is based on corners and depth values from Kinect's infrared sensor. Obstacles are detected in images from a camera using corner detection, while input from the depth sensor provides the corresponding distance. This ensures that the system can effectively alert users to potential obstacles in their path, thereby enhancing their spatial awareness and safety during movement.

Although some problems were experienced with the Kinect in outdoor locations, it was found to be reasonably reliable indoors [14]. The developed system is dependent on the Kinect hardware and thereby may be too restrictive, expensive and hardware may be difficult to obtain due to obsolete parts.

Table 2.1 shows a comparison of the related systems found in the literature.

Table 2.1 Related studies.

Author	Problem /Topic	Method Technology Framework	Strength	Weaknesses
Krishna et al. (2020)	A vision system with 3D Audio Feedback to assist Navigation for visually impaired.	Raspberry Pi units to capture the images. MobileNet to analyse and detect objects from the image.	Can detect obstacles. Useful and accurate for navigation.	Heavy, expensive, and single point of failure.
Mallikarjuna, Hajare and Pavan (2022)	A cognitive IoT system for visually impaired which helps them to move from one place to another without depending on others.	Raspberry Pi 3 and a Raspberry Pi Camera, and a speaker. OpenCV used to access the camera.	Less expensive to develop. It is semi-portable.	Reduced battery life. Cannot operate during the night. Was trained to recognize only four objects.
Sirouspour et al. (2011)	Analysis and implementation of walking support system for visually impaired people.	Ultrasonic sensors, infrared sensor, microcontroller, servo motor, and a buzzer.	Less expensive to implement.	Limited to standard pace of mobility and cannot differentiate between animate and inanimate obstacles. Cannot identify the detected objects. Uses a laptop for terrain analysis.
Perera et al. (2017)	An assist system for visually impaired people to recognize pure colours.	Arduino development board, TCS230	Can detect up to 12 colours.	Difficult to use as it requires the user to be

		colour sensor and an Android mobile application.		able to use a smartphone to convert the detected colour into audio. Limited to colour detection.
Kanwal et al. (2015)	A navigation system for the visually impaired: A Fusion of Vision and Depth Sensor.	Microsoft Kinect for Xbox 360, battery, On/Off switch, and a laptop for processing.	It operates reasonably reliable indoors.	High cost, heavy and not easy to use. Cannot operate outdoor.

2.3 Summary

This chapter discussed related studies that have been conducted to assist the VI individual to move around in an environment. Although some of the methods described in the literature offer some assistance to VI people, there is clearly a need for significant improvements that that utilize the advantages of current hardware and software advancements to provide a compact solution. In the next chapter a detailed description of the proposed system is given.

CHAPTER 3 Proposed System

3.1 Introduction

This chapter offers an exhaustive examination of the various technologies integrated into the proposed system. Furthermore, it delineates the overarching architectural framework that underpins the system's design. The subsequent chapter delves into a meticulous discussion of the development processes for both the hardware and software components, providing a detailed account of their implementation and integration.

3.2 An overview of the system architecture

Previous research underscores the critical importance of meticulously considering the weight and dimensions of the system, as these factors significantly influence user interaction and usability. The proposed system must be designed to be modular, energy-efficient, portable, reliable, and user-friendly. To achieve modularity and ease of use, the system is divided into distinct modules or subsystems. Figure 3.1 illustrates the various hardware and software components that constitute the proposed system. Subsequent sections will delve into the specifics of the hardware and software that are proposed.

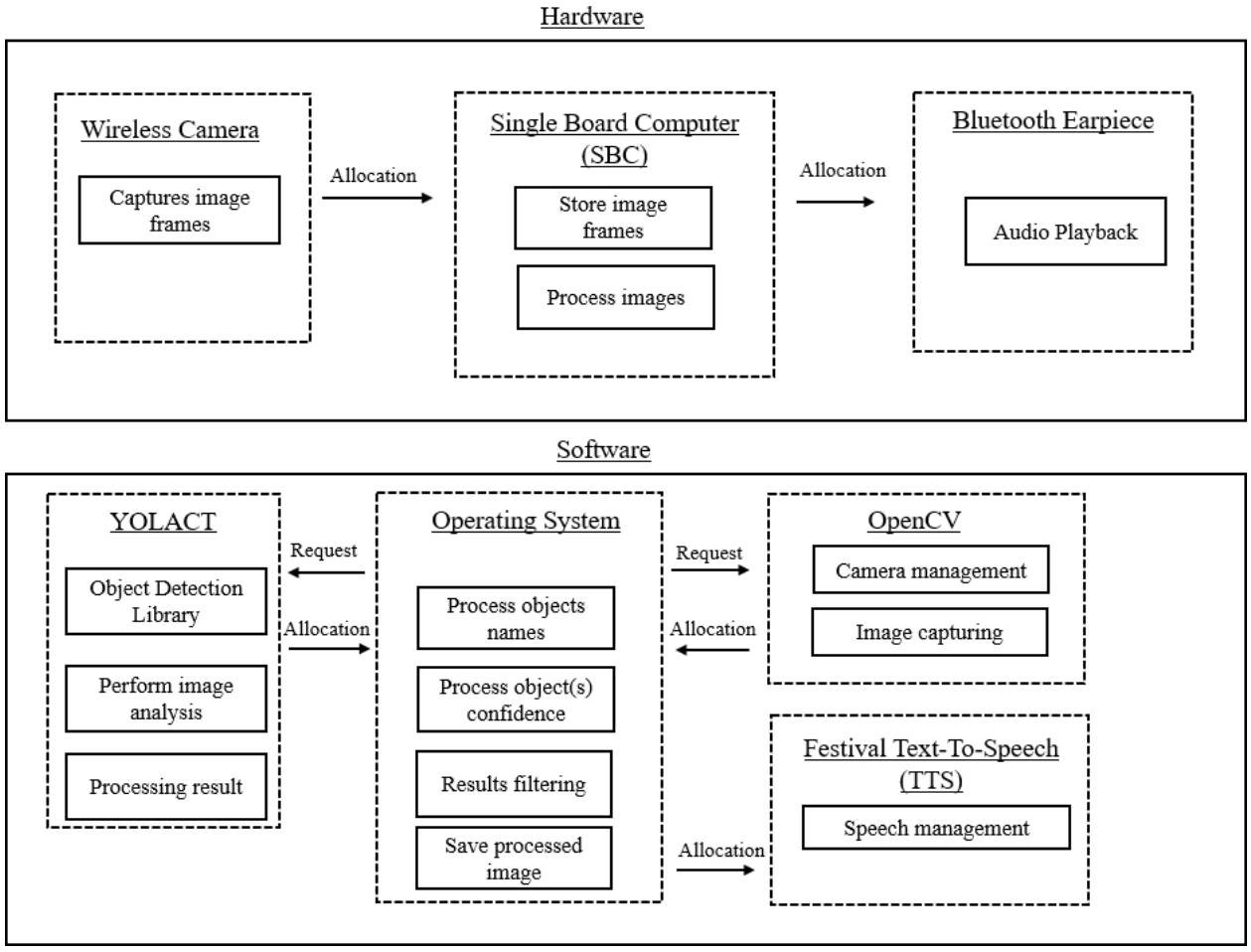


Figure 3.1. Hardware and software components of the proposed system.

3.3 YOLACT object detection algorithm

Object detection algorithms are a fundamental part of computer vision that enables computers to identify and locate objects within an image or video. Object detection is a computer vision task where algorithms detect instances of objects from predefined classes (such as humans, animals, and vehicles) in digital images or videos. The primary goal is to identify what objects are in a visual scene.

There are generally two types of algorithms for object detection namely, 'one-stage' detectors and 'two-stage' detectors. In one-stage detectors, methods like YOLO (You Only Look Once) and SSD (Single Shot Detection), predict object bounding boxes directly from

the image in one go. They are known for their speed but may trade off some accuracy compared to two-stage methods [15]. In two-stage detectors, such as R-CNN (Region-based Convolutional Neural Network) and its variants, it first generates region proposals and then classify those regions into objects. Two-stage detectors tend to be more accurate but slower than one-stage detectors. Over the years, object detection algorithms have evolved significantly. Deep learning-based methods have become the state-of-the-art, providing substantial improvements in accuracy and speed. Algorithms like Faster R-CNN, YOLO, and RetinaNet have set new benchmarks in object detection tasks [16].

Object detection has a wide range of applications, including security and surveillance, autonomous vehicles applications, image retrieval systems, and many more. It is a crucial step towards enabling machines to understand visual data as humans do. In this research, YOLACT is used for object detection and is classified as a one-stage detector [17]. It is designed for real-time instance segmentation and operates by directly predicting the bounding boxes and class probabilities from the image in a single pass, without the need for a separate region proposal step that is characteristic of two-stage detectors [17], [18]. This approach allows YOLACT to achieve a balance between speed and accuracy, making it suitable for applications that require real-time performance [17].

YOLACT, is a real-time instance segmentation algorithm that has made significant strides in the field of computer vision. It has the following unique characteristics:

i. The algorithm is fully convolutional, which means it relies on convolutional neural networks (CNNs) without the need for any dense layers. This design choice contributes to its efficiency and speed [19].

ii. YOLACT breaks instance segmentation into two parallel subtasks: generating a set of prototype masks and predicting per-instance mask coefficients as shown in Figure 3.2. Instance masks are then produced by linearly combining these prototypes with the mask coefficients [19].

iii. YOLACT introduces Fast non-maximum suppression (NMS), a quicker alternative to standard NMS methods. Fast NMS helps to reduce the time taken

for post-processing detections while maintaining a marginal performance penalty [16].

iv. Analysis of YOLACT's prototypes has shown that they learn to localize instances in a translation variant manner, which is quite remarkable considering the fully convolutional nature of the model [19], [20], [21].

v. YOLACT can be added to almost any modern object detector, making it a versatile tool for various applications that require real-time instance segmentation [19], [20], [21].

Figure 3.2 and Figure 3.3 show the architecture of the YOLACT object detection algorithm. YOLACT uses a Residual Network with 101 layers (ResNet-101) with Feature Pyramid Networks (FPN) to create feature maps efficiently, reducing computational requirements compared to conventional approaches.

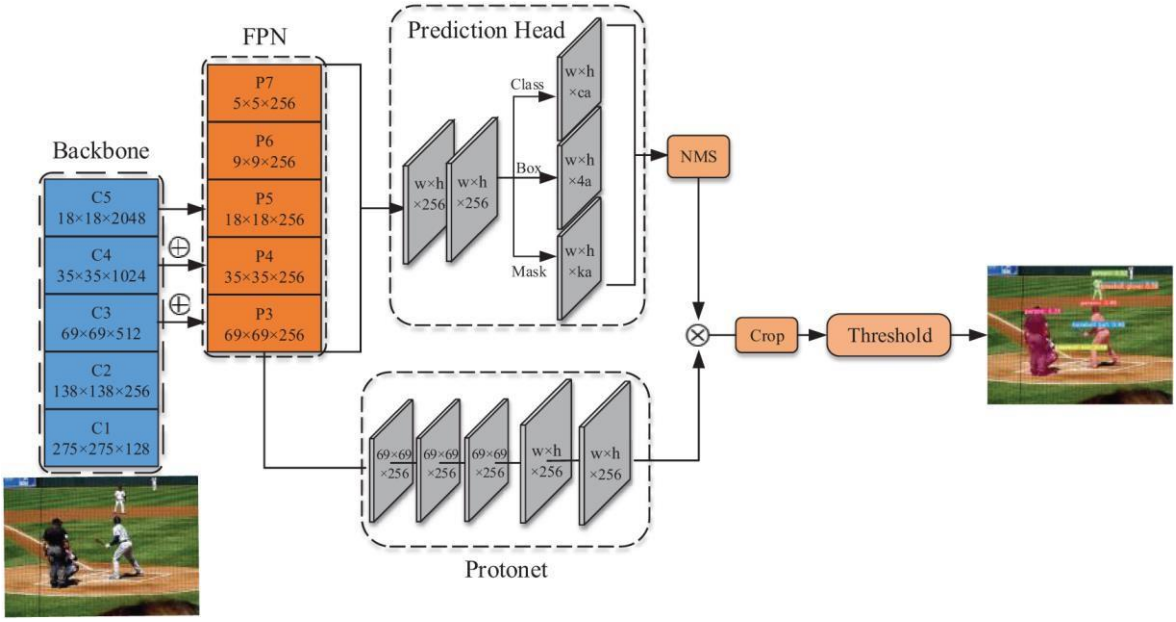


Figure 3.2. Basic architecture of YOLACT [22].

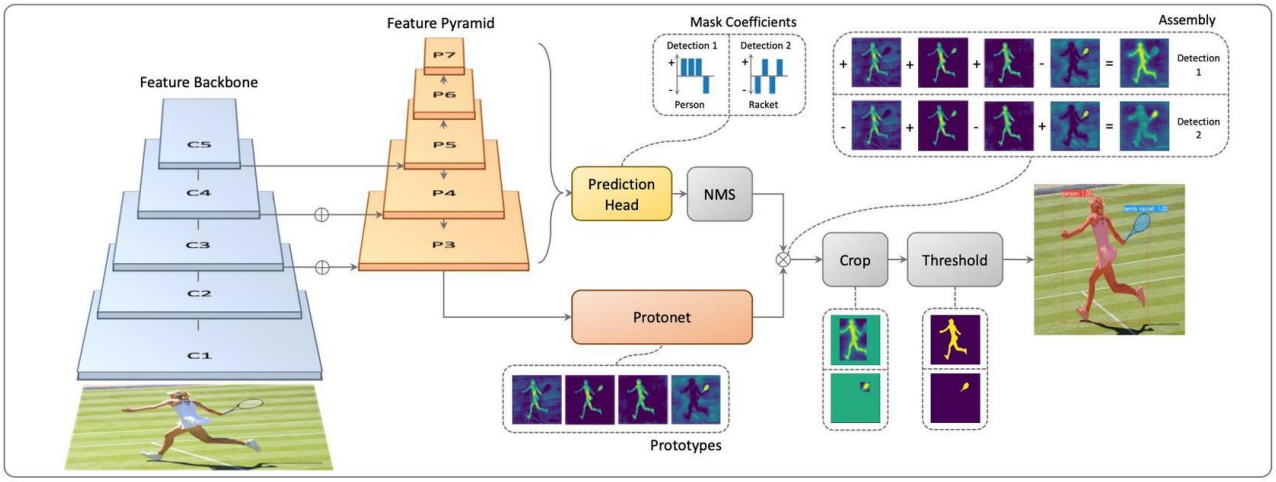


Figure 3.3. Advanced architecture of YOLACT [23].

The YOLACT architecture is defined by:

i. YOLACT utilizes a backbone network, typically based on architectures like ResNet or DarkNet, to extract fundamental features from the input image.

ii. The YOLACT method creates a set of prototype masks using the Protonet. These prototypes serve as initial representations for different object instances.

iii. The model predicts per-instance mask coefficients. These coefficients are used to combine the prototype masks and generate instance-specific masks.

iv. By linearly combining the prototype masks with the predicted coefficients, YOLACT produces high-quality instance masks.

v. YOLACT optimizes its components to achieve real-time performance while maintaining accuracy [24].

The speed and efficiency of the architecture make it unique. YOLACT uses ResNet-101 with FPN to create feature maps efficiently, thereby reducing computational requirements compared to conventional approaches. The model achieves approximately 30 mask mean Average Precision (mAP) on the MS COCO dataset, operating at 33.5 frames per second when trained and evaluated on a single NVIDIA Titan Xp GPU [25]. This means that the object detection model is capable of accurately identifying and segmenting objects in images at a high speed provided a high-end GPU is used. mAP is a common metric used to

evaluate the performance of object detection models. It measures the accuracy of the model by averaging the precision across different recall levels.

ResNet-101 is a powerful convolutional neural network architecture that has significantly improved image classification performance. ResNet addresses the degradation problem encountered when deep networks are trained. As networks become deeper, accuracy saturates and then degrades rapidly. Instead of directly fitting the desired underlying mapping, ResNet explicitly lets layers fit a residual mapping defined by:

$$\mathcal{F}(x) + x \tag{3.1}$$

Where, x represents the input to a layer, and $\mathcal{F}(x)$ denotes the residual mapping learned by the stacked layers [26].

Shortcut connections (also known as 'skip connections') play a crucial role in ResNet. These connections allow gradients to flow directly through the network without vanishing or exploding. The output of each layer is added to the output of a deeper layer, thereby preserving information and aiding optimization. ResNet-101 is a deep network that achieves impressive performance. The architecture includes residual blocks with skip connections, batch normalization, and Rectified Linear Unit (ReLU) activation functions. The shortcut connections ensure that gradients can flow efficiently during training. ResNet-101 is pretrained on the ImageNet dataset, which contains millions of labelled high-resolution images across various categories. The pretrained model can then be fine-tuned for specific tasks like object detection or segmentation. ReLU is a fundamental activation function used in artificial neural networks (ANNs) and is defined as**:**

$$f(x) = \max(0, x) \tag{3.2}$$

If the input x is positive then ReLU outputs the input directly, otherwise it outputs zero. ReLU has become the default activation function for many ANNs due to its advantages which include:

i. Traditional activation functions such as sigmoid and hyperbolic tangent suffer from the vanishing gradient problem. As the networks get deeper, gradients become tiny, making training difficult. ReLU helps mitigate this issue.

ii. ReLU allows models to learn faster and achieve better performance.

iii. ReLU is computationally efficient and easy to implement.

Here are the fundamental operating principles of YOLACT:

i. Prototype Generation

YOLACT generates a set of prototype masks P for the entire image. These prototypes can be represented as:

$$P_i = \{P_1, P_2, \dots, P_k\} \tag{3.3}$$

where P_i is the i -th prototype mask and k is the total number of prototypes.

ii. Mask Coefficients

For each detected object j, YOLACT predicts a set of coefficients C_j :

$$C_j = \{c_{j1}, c_{j2}, \dots, c_{jk}\} \tag{3.4}$$

where c_{ji} is the coefficient for the ith prototype mask for object j.

iii. Instance Mask Generation

The final mask M_j for each object j is obtained by linearly combining the prototype masks using the predicted coefficients:

$$M_j = \sum_{i=1}^{k} c_{ji} P_i \tag{3.5}$$

This equation shows how the prototypes are weighted and summed to form the final instance mask.

iv. Bounding Box and Class Prediction

YOLACT also predicts the bounding box B_j and class $\hat{y}_j$ for each object j

$$B_j = (x, y, w, h) \tag{3.6}$$

where x and y are the coordinates of the center, and w and h are the width and height of the bounding box.

The class prediction $\hat{y}_j$ is obtained using a softmax function over the class scores:

$$\hat{y}_j = \arg\ max(\text{softmax}(s_j)) \tag{3.7}$$

where s_j are the class scores for object *j*.

v. Loss Function

The loss function *L* used to train YOLACT is a combination of multiple components:

$$L = L_{\text{cls}} + L_{\text{box}} + L_{\text{mask}} \tag{3.8}$$

where:

L_{cls} is the classification loss.

L_{box} is the bounding box regression loss.

L_{mask} is the mask loss.

vi. Non-Maximum Suppression (NMS)

To filter out overlapping detections, YOLACT uses Non-Maximum Suppression NMS. The NMS algorithm can be described as:

$$D = \text{NMS}(B, scores, threshold) \tag{3.9}$$

D representing the set of final detections, constitutes the output. *B* are the bounding boxes, *scores* are the confidence scores, and *threshold* is the IoU threshold for suppression

These equations and principles form the core of YOLACT's operation, enabling it to perform real-time instance segmentation efficiently. Image recognition involves identifying and classifying objects or features within digital images. One of the most commonly used models for image recognition is the CNN, in which YOLACT is based on. CNNs are a class of deep neural networks specifically designed for processing structured grid data, such as images. They are highly effective for image recognition tasks due to their ability to capture spatial hierarchies in images.

Key Components of CNNs:

i. *Convolutional Layers:* These layers perform convolution operations on the input image, employing filters (kernels) to extract salient features, including edges, textures, and patterns. The output of a convolutional layer is called a feature map.

$$\text{Feature Map} = \text{Input Image} * \text{Filter} + \text{Bias} \tag{3.10}$$

where (*) denotes the convolution operation.

ii. *Activation Function:* Typically, the ReLU function, as defined in equation (3.2), is used to introduce non- linearity into the model.

iii. *Pooling Layers:* The spatial dimensions of the feature maps are reduced by these layers, a process that retains critical information and concurrently mitigates computational complexity. Max pooling and average pooling represent prevalent pooling operations.

$$\text{Max Pooling}(x) = max(x) \tag{3.11}$$

Where *x* represents the input data, which is typically a small region (or window) of the larger input matrix while *max(x)* denotes taking the maximum value from the input data *x*.

iv. *Fully Connected Layers:* After several convolutional and pooling layers, the high-level reasoning in the neural network is done via fully connected layers. These layers connect every neuron in one layer to every neuron in the next layer.

$$\text{Output} = \text{Activation}(\text{Weights} \cdot \text{Input} + \text{Bias}) \tag{3.12}$$

v. *Softmax Layer:* The final layer in a CNN used for classification tasks. It converts the output into a probability distribution over the target classes.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (3.13)$$

Where z_i represents the i-th element of the input vector z. The input vector z contains the raw scores (also called logits) from the neural network.

e^{z_i} is the exponential function applied to the i-th element of the input vector. The exponential function ensures that all the transformed values are positive.

$\sum_j e^{z_j}$ is the sum of the exponentials of all elements in the input vector z. This sum acts as a normalization factor, ensuring that the output probabilities sum to 1.

Training a CNN model involves the following steps:

i. *Data Preparation:* Collect and preprocess a large dataset of labelled images.

ii. *Model Initialization:* Define the architecture of the CNN, including the number of layers and the size of filters.

iii. *Forward Propagation:* Pass the input images through the network to obtain predictions.

iv. *Loss Calculation:* Compute the loss using a suitable loss function, such as cross-entropy loss for classification tasks.

$$\text{Cross-Entropy Loss} = -\sum_i y_i \log(\hat{y}_i) \quad (3.14)$$

where y_i is the true label and $\hat{y}_i$ is the predicted probability.

v. *Backpropagation*: Calculate the gradients of the loss with respect to the network parameters and update the parameters using an optimization algorithm like stochastic gradient descent (SGD).

$$\theta = \theta - \eta \nabla L(\theta) \quad (3.15)$$

where θ represents the model parameters, η is the learning rate, and $\nabla L(\theta)$ is the gradient of the loss function.

By iteratively training the CNN on a large dataset, the model learns to recognize patterns and features in images, enabling it to accurately classify new images.

Figure 3.4 shows the system operation sequence. The operation sequence is more detailed as it outlines the use of YOLACT for the object detected and the use of OpenCV for camera access and capturing of the image. During the initialization phase of YOLACT, the object detection code executes by loading a pre-trained model file from local storage. The model file contains information and features of the trained object. In this study, the default model that is supplied with the YOLACT C++ library is used.

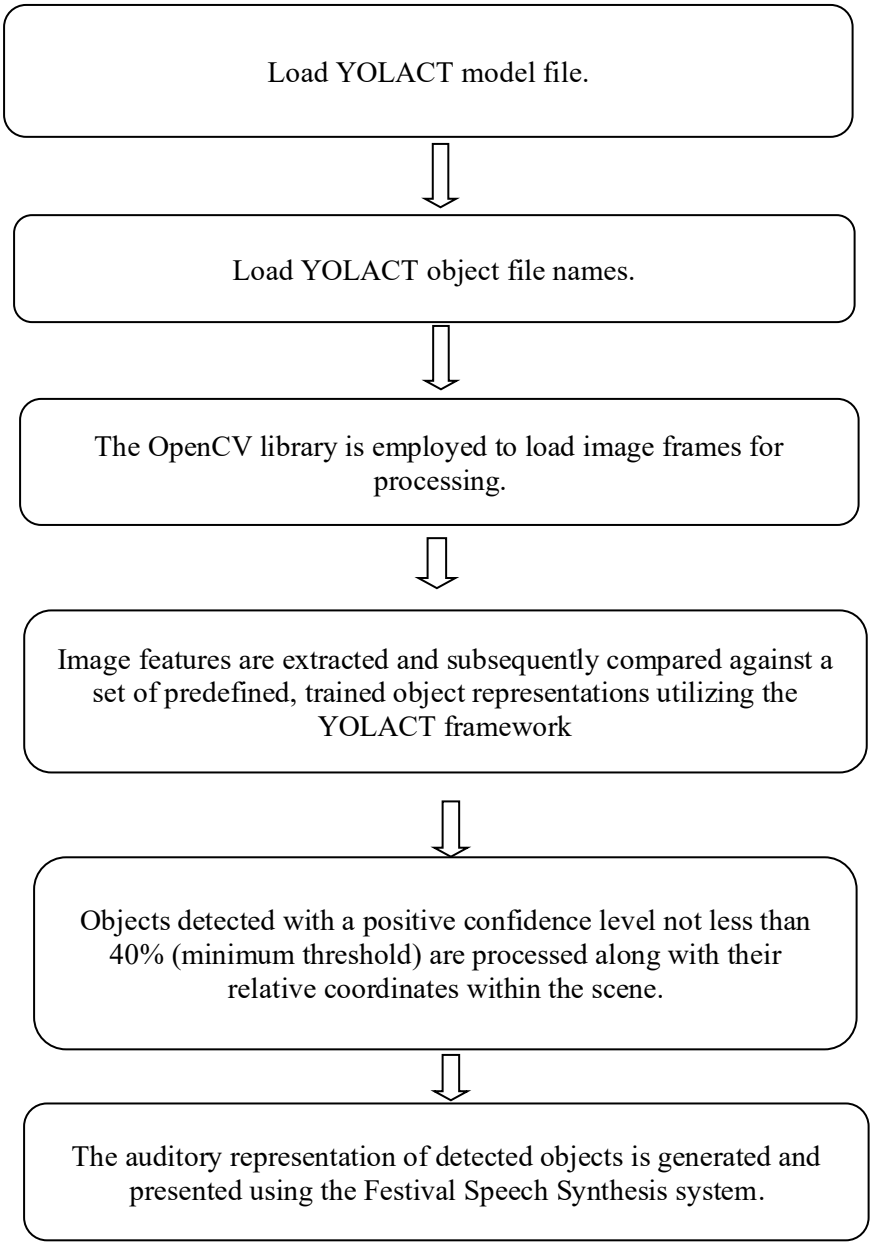


Figure 3.4. System operating sequence.

OpenCV is an open-source computer vision and machine learning software library. Originally developed by Intel™, it is now maintained by the OpenCV Foundation. OpenCV is engineered to establish a standardized infrastructure for computer vision applications and to expedite the integration of machine perception capabilities into commercial products [27]. Key features of OpenCV are as follows:

i. OpenCV offers cross-platform compatibility, supporting a range of programming languages, including C++, Python, Java, and MATLAB, and operating across diverse operating systems such as Windows, Linux, macOS, iOS, and Android.

ii. It is optimized for real-time applications, making it suitable for tasks that require immediate processing.

iii. OpenCV includes over 2500 optimized algorithms for a wide range of computer vision and machine learning tasks, such as object detection, face recognition, and image filtering [28].

The following is a list of commonly used libraries in OpenCV:

i. Basic data structures and operations, including matrix operations, basic image processing, and utility functions.

ii. Functions for image filtering, transformations, and analysis.

iii. Tools for motion analysis, object tracking, and background subtraction.

iv. Algorithms for detecting objects, faces, and other features in images.

v. A suite of ML algorithms, including support vector machines, decision trees, and neural networks.

3.4 Text-to-speech synthesis

Speech synthesis refers to the artificial generation of human voice [29]. A text-to-speech (TTS) system converts written text into spoken language, aiming to produce audio that is both intelligible and natural. TTS enhances accessibility for visually impaired users and supports applications such as virtual assistants, navigation systems, audiobooks, and automated customer service. Synthesized speech is typically produced by concatenating prerecorded segments stored in a database, with quality determined by its clarity and similarity to human voice [30].

TTS systems vary according to the size of the speech units they store. A system that stores phones or diphones provides the widest output range, although this may come at the expense of clarity [30]. Phones represent the smallest distinct units of sound in a language, such as individual phonemes like ‘p’, ‘a’, or ‘t’. A TTS system that stores phones can generate a broad range of outputs, since these basic units can be combined in multiple ways to produce almost any word. Nevertheless, this approach often results in speech that is less natural or less clear, as the system must blend numerous small units together, which can lead to abrupt or unnatural transitions between sounds. Diphones, by contrast, consist of pairs of adjacent phones and represent the transition between two phonemes. By storing diphones, a system can achieve smoother transitions between sounds compared to phones, as diphones capture how phonemes blend into one another in natural speech. Although this improves clarity to some extent, diphone-based synthesis may still fall short of the naturalness achieved when larger speech units are employed.

In certain application domains, storing entire words or sentences enables text-to-speech systems to produce high-quality output. A typical TTS system consists of two primary components: the front-end and the back-end. The front-end processes raw text, converting numbers and abbreviations into their fully spelled-out equivalents through a procedure known as text normalization, preprocessing, or tokenization. For instance, the token ‘Ms’ is expanded to ‘Miss’, while ‘57’ is converted to ‘fifty-seven’. Following normalization, the front-end assigns phonetic transcriptions to each word and segments the text into prosodic units such as phrases, clauses, and sentences [31]. The assignment of phonetic transcriptions is referred to as text-to-phoneme or grapheme-to-phoneme conversion.

The primary function of the back end is to transform symbolic linguistic representations into audible sound waves. One widely used approach for this conversion is concatenative synthesis. Concatenative synthesis is applied in speech and sound synthesis to generate output by combining smaller units or segments. Within speech synthesis, this method involves concatenating short, pre-recorded speech samples or phrases to produce continuous, natural-sounding output [32], [33]. The process requires analyzing the source sound to identify the most appropriate segments that meet the desired criteria, which are then stitched together to form the synthesized speech [32], [33], [34].

The concatenative synthesis approach involves several steps which include:

i. The process starts by dividing a sound recording into individual units or snippets.

ii. Specific units are selected from a sound database to create the desired sound.

iii. The selected units are then reassembled to form new sounds.

Concatenative synthesis is the most used method because of its natural-sounding speech. However, another method that can be used to generate speech is 'formant synthesis' approach.

While concatenative synthesis relies on combining prerecorded speech segments to generate natural-sounding output, another important aspect of speech production is the acoustic structure of the human voice itself. Formants, which are amplitude peaks in an audio signal, are closely associated with specific frequencies in human speech. Each vowel is characterized by distinct formants, and these frequency patterns contribute to the unique quality of the voice producing them [35].

Table 3.1 shows the advantages and disadvantages of both concatenative and formant synthesis.

Table 3.1 Advantages and disadvantages of concatenative and formant synthesis.

	Concatenative synthesis	**Formant synthesis**
Advantages	Produces natural-sounding speech because it uses real recorded samples.	Allows for more nuanced and expressive manipulation of vocal timbre and articulation.
	Allows for variations in prosody, intonation, and expression.	Can simulate different vocal tract shapes and resonances.
		Useful for creating specific voice characteristics such as cartoon voices and robotic voices.
Disadvantages	Requires a large database of recorded speech segments.	May sound less natural than concatenative synthesis.
	Can be computationally expensive due to the need for real-time selection and concatenation.	Requires accurate modeling of vocal tract resonances.
	May have limitations in handling rare or unusual speech patterns.	Limited ability to capture natural variations in speech.

Festival is an open-source Linux text-to-speech system based on concatenative synthesis. In this study, it was employed to convert detected objects into audio due to its natural-sounding output. The framework supports multiple programming languages, is straightforward to use once installed, and allows extensive customization for research and development. Moreover, Festival integrates seamlessly with tools such as OpenCV, enabling comprehensive systems that combine object detection with speech synthesis.

3.5 The use of Festival Speech Synthesis

To enable auditory representation of detected objects, a speech synthesis system was integrated into the design. The Festival Speech Synthesis system was selected due to its open-source licensing and proven effectiveness as a text-to-speech engine within the Linux environment [37]. Developed at the University of Edinburgh's Center for Speech Technology, Festival provides a complete suite of modules for text-to-speech conversion and serves as a robust research platform in speech synthesis [38]. Through its Application Programming Interface (API), Festival supports the development of speech analysis tasks and delivers full text-to-speech functionality [38].

Festival employs a modular design based on a 'blackboard architecture,' where the core data structure, termed the 'utterance,' facilitates information exchange between system modules [39][40]. The architecture is implemented in C++, while individual modules may be developed in C++ or other high-level languages [39][40]. The C++ foundation of Festival ensures compatibility with the proposed system. A shell API was integrated and invoked within the C++ application to streamline development. This approach enables text-to-audio synthesis with a single line of code, removes the need for header file inclusion, and reduces execution time.

Festival Speech Synthesis was selected for this project because it is the most popular text-to-speech pre-compiled binary that supports the Linux environment. Sending the command *echo "There is a stop sign ahead" | festival –tts* into the linux terminal will output quoted sentence into the system speakers as audio. If the Bluetooth USB adapter is connected to the earpiece, the sound will be routed into the earpiece. The person wearing the earpiece will hear the sound *'There is a stop sign ahead'*.

Table 3.2 shows some of the common objects that the model is trained to detect. The algorithm will try to match the objects from the image file with the ones it was trained to detect. The algorithm will return an array of classified objects together with the confidence level. The confidence level is used to determine the level of accuracy per object detected. The minimum confidence level is set to 40% and the Festival Speech Synthesis library is used as a TTS synthesis. The operation sequence is purely used to demonstrate the sequence that the system follows for a lifecycle of successful object detection and audio notification.

Table 3.2 Common detectable objects.

List of common detectable objects	
traffic light	laptop
person	keyboard
car	mouse
bus	tv
motorcycle	oven
fire hydrant	bed

3.6 Summary

In this chapter, we provided a comprehensive overview of the hardware and software components integral to the proposed systems. Additionally, we examined the YOLACT object detection algorithm and the Festival speech synthesis system in detail. The subsequent chapter will delve into an in-depth analysis of the development processes for both the hardware and software components of the proposed system.

CHAPTER 4 Development of Hardware and Software of the VI system

4.1 Introduction

In this chapter, we meticulously describe the development of both the hardware and software components of the proposed system. Furthermore, it provides a detailed account of the implementation and operationalization of the algorithms previously introduced, with a specific emphasis on the object detection and text-to-speech functionalities.

4.2 Operating system and supporting files

The chosen Operating System (OS) for this project is Debian. The command line (CLI) based Debian OS was selected to increase the performance as no graphical user interface was needed in this project. Find below, the reasons why Debian was chosen for this projects.

i. Stability and Reliability: Debian is known for its stability and reliability, making it a solid choice for systems that require consistent performance over time. This is crucial for object detection systems that need to run continuously without frequent crashes or downtime.

ii. Security: Debian has a strong focus on security, with regular updates and a robust security team. This ensures that your object detection system is protected against vulnerabilities and threats.

iii. Package Management: Debian package management system, APT, is highly efficient and user-friendly. It allows for easy installation, updating, and management of software packages, including those required for object detection like OpenCV, TensorFlow, and PyTorch.

iv. Community Support: Debian has a large and active community. This means, one can find extensive documentation, forums, and support channels to help troubleshoot issues or optimize your object detection system.

v. Flexibility and Customization: Debian offers a high degree of flexibility and customization. One can tailor the operating system to meet the specific needs of your object detection system, whether it's optimizing performance, minimizing resource usage, or configuring specific hardware support.

vi. Compatibility: Debian supports a wide range of hardware architectures and peripherals, ensuring compatibility with various cameras, GPUs, and other components used in object detection systems.

vii. Open Source: As an open-source operating system, Debian permits modification and adaptation of the software to meet specific requirements without licensing restrictions. Such flexibility can be particularly advantageous for research and development in object detection.

viii. Long-Term Support (LTS): Debian provides long-term support for its stable releases, ensuring that users receive updates and security patches over an extended period. This sustained support is particularly advantageous for maintaining a reliable and secure object detection system.

The OS is saved into a 16 Gigabyte Secure Digital (SD) card and inserted into the SBC. It is essential to indicate that for any OS to communicate to an external device seamlessly, the relevant drivers were needed for the hardware. For the WIFI USB adapter, the required software driver used was the ‘RTL8192EU’ which is part of the package ‘firmware-realtek’ version 20190114+really20220913-0+deb10u2, and also enables the function for the Bluetooth USB adapter.

4.3 Hardware structure and prototype

The proposed setup makes use of a wireless camera to capture pictures and automatically generates an audio output relaying the detected object information to the VI individual. WIFI and Bluetooth wireless communication standards were chosen for this project because they provide connectivity that offers convenience, flexibility, and have widespread compatibility for many third-party devices. Figure 4.1 shows the block diagram of the specific hardware used in the VI assistance system. Table 4.1 shows the hardware specifications of each device.

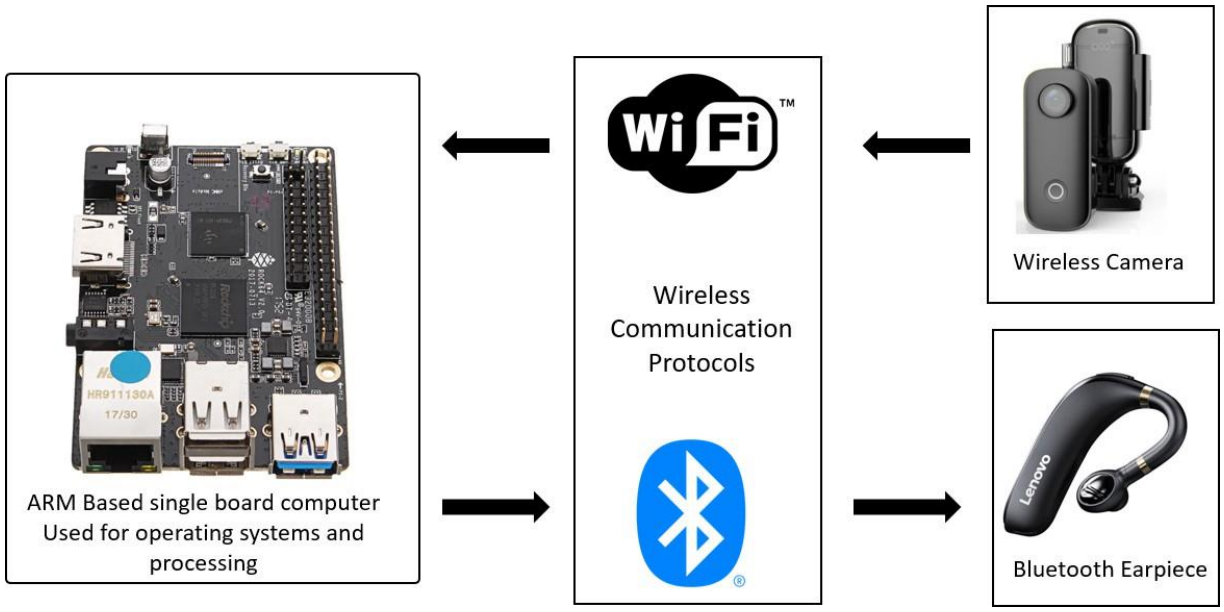


Figure 4.1. Hardware used for the autonomous VI assistance system.

Table 4.1 Hardware specifications of each device.

Device	Purpose	Protocol	Specification
Wireless Camera **SJCAM C100+**	Visual Input	WIFI	**Standard Recording:** 4K 30FPS, 2K 30FPS, 1080P 60/30FPS, 720P 120/60FPS **Photo Resolution:** 15MP, 12MP, 10MP, 8MP, 5MP, 3MP **Video Mode:** Normal mode, Time-lapse, Car mode, Vertical screen(9:16) Loop video **Photo Mode:** Single shooting, Continuous shooting, Timed photo **Video Format:** MP4 **Video Encoding:** H.265 **Photo Format:** JPG **Battery Capacity:** 730mAh

Bluetooth Earpiece **Lenovo HX106**	Audio Output	Bluetooth	**BT Version**: V5.0 **Transmission Distance**: 10m (barrier-free) **Wearing Type**: Ear hook **Speaker Diameter**: 10mm **Sensitivity**: 110dB±3dB at 1KHz **Battery Capacity**: 160mAh **Play Time**: >20 hours **Standby Time**: About 120 hours
Single Board Computer **PINE64 ROCK64**	Image processing Audio Processing	WIFI Bluetooth	**SoC:** 4 x ARM Cortex A53 cores @ 1.5 GHz **GPU:** ARM Mali 450 MP2 GPU **Memory:** LPDDR3 RAM (up to 4GB) **Networking:** Gigabit Ethernet **Storage:** Micro SD Slot eMMC module slot SPI Flash 128Mbit **Connections:** 4K digital video out 2x USB 2.0 Host 1x USB 3.0 Host PI-2 bus PI-P5+ bus IR R/X port Real Time Clock (RTC) port Power Over Ethernet (POE) A/V jack Power, Reset and Recovery buttons 3.5mm barrel power (5V 3A) port

The used SBC was selected because of the following reasons;

i. Memory Support: The board supports up to 4GB of LPDDR3 memory, allowing for efficient handling of large datasets and real-time processing.

ii. Versatile Connectivity: With multiple USB ports, Ethernet, and GPIO pins, the ROCK64 offers extensive connectivity options for integrating various sensors and peripherals. This allow us to add WIFI and Bluetooth connectivity.

iii. Open Source Support: The ROCK64 supports various open-source operating systems like Debian, Android, and Yocto, making it flexible for different development environments.

iv. Cost-Effective: Compared to other high-performance single-board computers, the ROCK64 offers a good balance of power and price, making it an economical choice for prototyping.

Figure 4.2 shows the final prototype of the system implemented with the combined weight of the proposed setup being ±400 grams. Figure 4.3 shows how to wear the wireless camera and the Bluetooth earpiece.

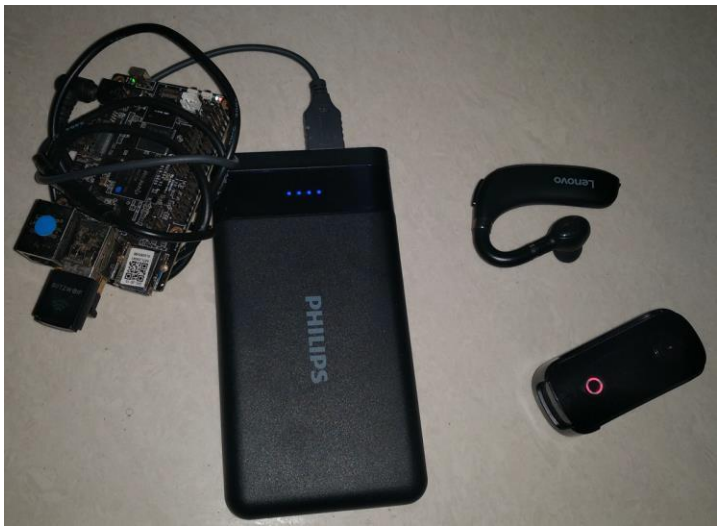


Figure 4.2. Final prototype.

Figure 4.3. Wearing wireless camera and Bluetooth audio earpiece.

4.4 Software architecture

Figure 4.4 shows the proposed software architecture of the system. The software architecture of the proposed system is segmented into three distinct modules.

The image input module requests camera access from the underlying operating system utilizing the *VideoCapture:open(...)* function of OpenCV. The parameter for this function is the IP address of the wireless camera designated for streaming. Access is granted if the camera is available and not currently utilized by another application. This module captures images from the wireless camera and requests to save them to a file. The OS permits saving to a local file provided there is no conflict with an existing file in the same directory that is in use by another application. While it is feasible to append the filename with the current timestamp, this project employs a single filename that is overwritten to prevent excessive storage consumption.

The image processing module is tasked with executing object detection on a provided image. The path of the captured image is supplied to the YOLACT algorithm, which subsequently conducts image analysis and feature extraction. The algorithm returns the results of the detected objects, including their names and confidence levels. This module then performs filtering of the results, retaining only those objects with a confidence level of no less than 40%. Additionally, if necessary, logs of the results are saved to local storage. The filtering process ensures that only objects meeting the specified confidence threshold are considered, thereby enhancing the accuracy and reliability of the detection outcomes.

The audio output module is tasked with transmitting the detected object information to the operating system terminal, accompanied by additional properties that instruct the OS to execute the pre-installed Festival Text-to-Speech (TTS) utility. This utility processes the sentence to be converted into audio output. Upon invocation, the TTS daemon directs the audio output to the system speaker. However, due to the system's pre-configuration for Bluetooth audio streaming, the sound is emitted through a connected Bluetooth earpiece. This configuration ensures seamless audio delivery via the preferred output device.

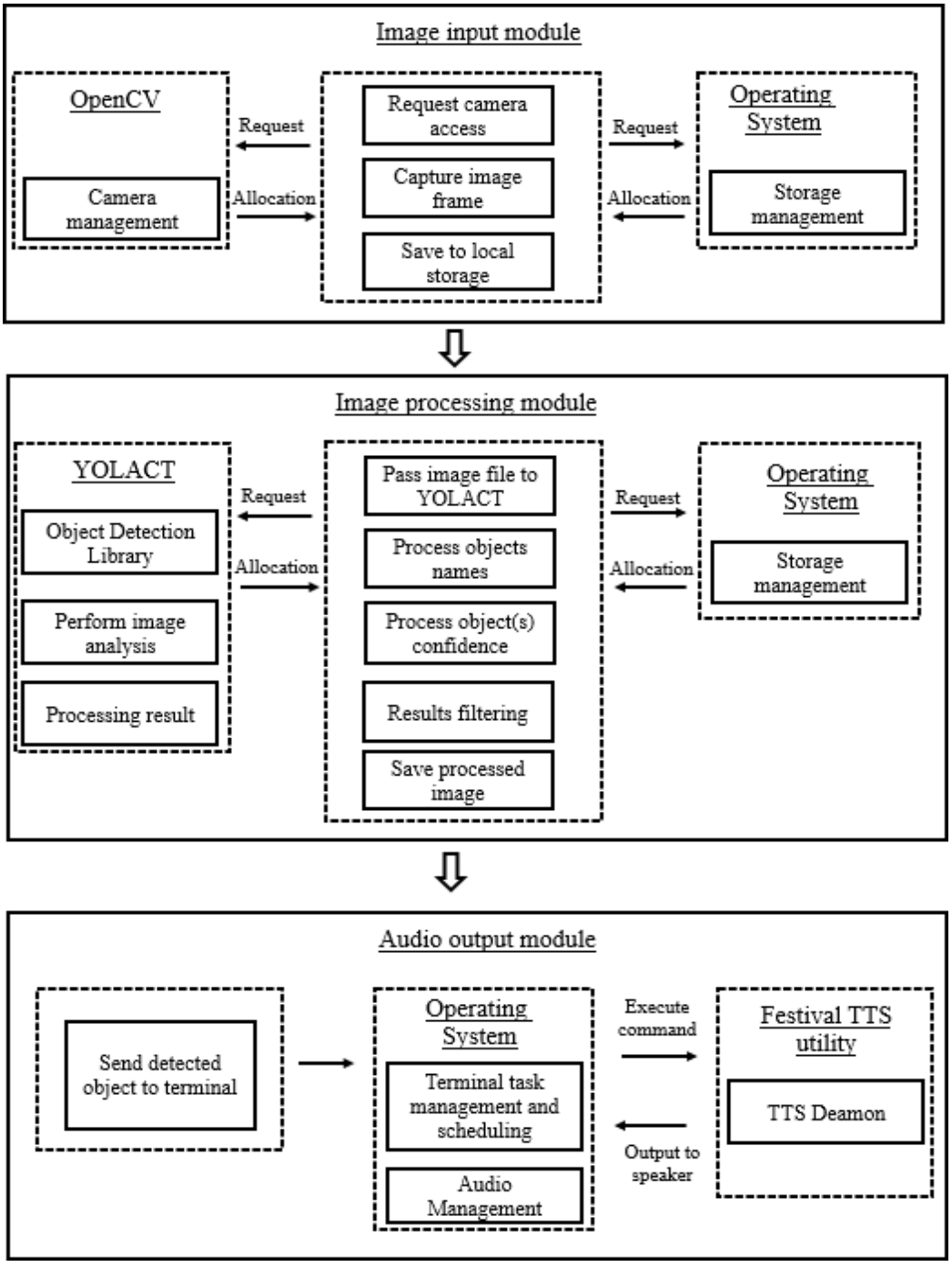


Figure 4.4. Software architecture of the VI assistance system.

Figure 4.5 shows the pseudocode of the proposed system. The preliminary step when the system is executed is to verify if the model file exists. Thereafter the camera is connected to the operating system, either by physical universal serial bus (USB) cable or wirelessly via Bluetooth. Upon gaining access to the camera, the system establishes a minimum acceptable accuracy level, set at 40%, which serves as the threshold to mitigate false alarms. The

program then enters a loop where the main task is performed. The system starts by using the camera handle to access the camera and capture the image. The captured image is then saved to a local file for processing. The saved file is then fed into the object detection algorithm. The algorithm returns the detected objects in a string format together with the confidence level associated with each string as an integer value. The system can save the image after processing highlighting the coordinates in which an object was detected. This is done by drawing a bound box and writing the name of the object on top of the box.

The subsequent procedure involves verifying the quantity of detected objects. When object(s) are detected, a loop is initiated corresponding to the number of detected objects, during which the information pertaining to each object is transmitted to the TTS synthesizer for auditory output. The system repeats the never-ending loop if the system is powered on, and all the supporting hardware is connected and active. The system may crash if any of the supporting hardware is disconnected while in use. The system can generate and save logged data for future debugging if needed but it is disabled to save memory space.

```
BEGIN FUNCTION run
IF file_exists('model_file.data') THEN
CONTINUE
ELSE
re-check
END IF

IF file_exists('model_string_names.data') THEN
CONTINUE
ELSE
re-check
END IF

LOAD 'model_file.data' INTO 'model_data_structure'
LOAD 'model_string_names.data' INTO 'model_strings_structure'

IF camera_access = available THEN
CONTINUE
ELSE
re-check
END IF

camera.open()
SET minimum_accuracy TO 40

WHILE TRUE DO
SET image_buffer TO camera.capture()
save_image_to_file(image_buffer, 'temp_image_file.png')

SET detection_array TO begin_object_detection('temp_image_file.png',
'model_data_structure', 'model_strings_structure', minimum_accuracy)

SET detected_strings TO extract_detection_strings(detection_array)
SET detected_confidence TO extract_detection_confidence(detection_array)

save_detection_into_output_file(detection_array, 'temp_output_file.png')

IF count(detected_strings) > 0 THEN
FOR EACH Index IN detected_strings DO
system_speak(detected_strings[Index] + ' detected ahead')
END FOR
END IF
END WHILE
END FUNCTION
```

Figure 4.5. Developed pseudocode for system execution.

4.5 Software utility development

The utilities are command-line applications that need to be developed for this system to be tested and implemented successfully. The first is the '*Imagecapture'* which is a C++ application developed for this to capture an image from a camera and save the resulting image file into local storage. The advantage is that the same application can be modified to capture an image from either a USB camera or a WIFI camera. Another advantage is that it can be configured to execute in a loop and capture an image for every fixed period and then saved to a different filename.

The other applications developed in this work are the '*yolactUSB'* and **'***yolactWIFI'*. Both these applications are also written in C++ and use the YOLACT object detection algorithm to identify objects in a scene. The main difference between the two applications is that one is utilized for USB camera image capture whilst the other is used for the WIFI camera.

Another developed utility is the '*yolactstatic'* which was responsible for processing a single image file from the local drive and used purely for testing purposes. It was necessary to execute the *imagecapture* application before *yolactstatic* as it would generate the required file for image processing. Namely, the *yolactstatic* application required *imagecapture* to operate independently. *Imagecapture* generated the image, while *yolactstatic* processed the image for object detection. The source code for the *yolactstatic* application is similar to *yolactUSB* and *yolactWIFI* with the only difference being that it does not access the camera hardware directly and only processes the image file as a parameter. The relevant utility source code files are shown in Appendix A1-A4.

4.6 WIFI protocol

WIFI (Wireless Fidelity) is a technology that allows devices to connect to the local network wirelessly. The efficiency and reliability of WIFI networks are largely determined by the underlying protocol algorithms. These algorithms govern various aspects of data transmission, including access control, error handling, and security. For the sake of completeness, a description of the WIFI standard is given.

4.6.1 Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)

CSMA/CA constitutes a foundational protocol within Wi-Fi networks for the management of data packet transmission. Prior to initiating transmission, a device performs a channel assessment to determine its availability. In the event that the channel is occupied, the device implements a random backoff mechanism, deferring its subsequent transmission attempt for a probabilistically determined duration. This method helps to minimize collisions and ensure efficient use of the network. The probability of successful transmission $P_{success}$ can be expressed as:

$$P_{success} = n \cdot p \cdot (1 - p)^{\{n-1\}} \tag{4.1}$$

Where, n is the number of stations and p is the probability of a station transmitting [45].

4.6.2 Request to Send/Clear (RTS/CTS)

The RTS/CTS mechanism is an optional protocol used to reduce collisions further, especially in environments with hidden nodes. A transmitting station initiates communication by sending a RTS frame to the access point, soliciting clearance for data transmission. Upon determining that the communication channel is available, the access point responds with a CTS frame, thereby granting the originating device permission to transmit its data payload. This exchange helps to reserve the channel and prevent other devices from transmitting simultaneously. Throughput S of the RTS/CTS mechanism can be approximated by:

$$S = \frac{E[\text{Payload}]}{E[\text{Payload}]+E[\text{RTS}]+E[\text{CTS}]+E[\text{ACK}]+3\cdot T_{\text{SIFS}}+T_{\text{DIFS}}} \tag{4.2}$$

Where $E[\text{Payload}]$ is the expected payload size, $E[\text{RTS}]$, $E[\text{CTS}]$, and $E[\text{ACK}]$ are the expected sizes of the RTS, CTS, and ACK frames, respectively, and T_{SIFS} and T_{DIFS} are the Short Interframe Space and Distributed Interframe Space times [46].

4.6.3 Automatic Repeat Query (ARQ)

ARQ is an error control protocol that ensures data integrity by retransmitting lost or corrupted packets. When a device receives a data packet, it sends an acknowledgment (ACK) back to the sender. If the sender does not receive an ACK within a specified time, it retransmits the packet. This process continues until the packet is successfully received or a maximum number of attempts is reached. The efficiency η of the ARQ protocol can be given by:

$$\eta = \frac{1}{1+2 \cdot P_e} \tag{4.3}$$

Where, P_e is the probability of a packet error [47].

4.6.4 Wi-Fi Protected Access (WPA)

Since network security is a critical aspect, the WPA is a unique security protocol designed to protect data transmitted over the Wi-Fi standard. This protocol incorporates encryption algorithms to ensure data confidentiality and implements authentication mechanisms to restrict network access to authorized devices exclusively. The most recent iteration, WPA3, provides enhanced security capabilities, including more robust encryption and mitigation against brute-force attacks through the implementation of a 256-bit cryptographic key. The strength of the encryption can be represented by the key length L in bits, where the security level S is:

$$S = 2^L \tag{4.4}$$

WIFI protocol algorithms play a crucial role in the performance and security of wireless networks. By managing data transmission, minimizing collisions, ensuring data integrity, and providing robust security, these algorithms enable reliable and efficient wireless communication.

4.7 Bluetooth protocol

Bluetooth represents a short-range wireless communication technology facilitating data exchange between electronic devices. The efficiency and reliability of Bluetooth networks are largely determined by the underlying protocol algorithms. These algorithms govern various aspects of data transmission, including access control, error handling, and security.

4.7.1 Frequency Hopping Spread Spectrum (FHSS)

Bluetooth uses FHSS to minimize interference and improve security. The algorithm employs rapid carrier frequency hopping across a wide spectrum of channels, governed by a pseudorandom sequence shared between the transmitting and receiving nodes. The hopping sequence $f(t)$ can be represented as:

$$f(t) = f_0 + \Delta f \cdot (\lfloor \frac{t}{T_h} \rfloor \mathrm{mod} N) \tag{4.5}$$

Where, f_0 is the initial frequency, Δf is the frequency step, T_h is the hopping period, and N is the number of channels [47].

4.7.2 Time-Division Duplex (TDD)

Bluetooth employs TDD to allow bidirectional communication on the same frequency channel. In TDD, the time is divided into slots, and each device transmits and receives in different time slots. The throughput S in a TDD system can be expressed as:

$$S = \frac{R}{2} \tag{4.6}$$

Where, R is the raw data rate of the channel [48].

4.7.3 Automatic Repeat Request (ARQ)

ARQ is an error control protocol that ensures data integrity by retransmitting lost or corrupted packets. When a device receives a data packet, it sends an acknowledgment (ACK) back to the sender. If the sender does not receive an ACK within a specified time, it

retransmits the packet. The transmission cycle continues until successful packet reception is confirmed or the pre-configured limit for retransmission attempts is attained. The efficiency η of the ARQ protocol can be given by the same Equation (4.3) [50].

4.7.4 Bluetooth Security Protocols

Security constitutes a fundamental consideration in Bluetooth network deployments. To safeguard data transmission, Bluetooth employs a suite of security protocols encompassing pairing procedures, authentication mechanisms, and encryption algorithms. The strength of the encryption can be represented by the key length S in bits, where the security level S is defined by Equation (4.4).

For example, Bluetooth uses a 128-bit key, providing a security level of (2^{128}) [51].

Bluetooth protocol algorithms play a crucial role in the performance and security of wireless networks. By managing data transmission, minimizing interference, ensuring data integrity, and providing robust security, these algorithms enable reliable and efficient wireless communication.

4.8 System initialization

In order for the system to operate at its best, it needs to be initialized. Initialization is the stage where the system is turned on for the first time and background processes run to prepare it to perform its main functions.

Figure 4.6 illustrates the initialization sequence upon powering on the SBC. While the majority of the steps are elaborated upon in the subsequent subsections, the initial action required once the SBC is powered on is to enable both wireless communication protocols, namely WIFI and Bluetooth. This activation facilitates communication between the SBC, the WIFI camera, and the Bluetooth earpiece.

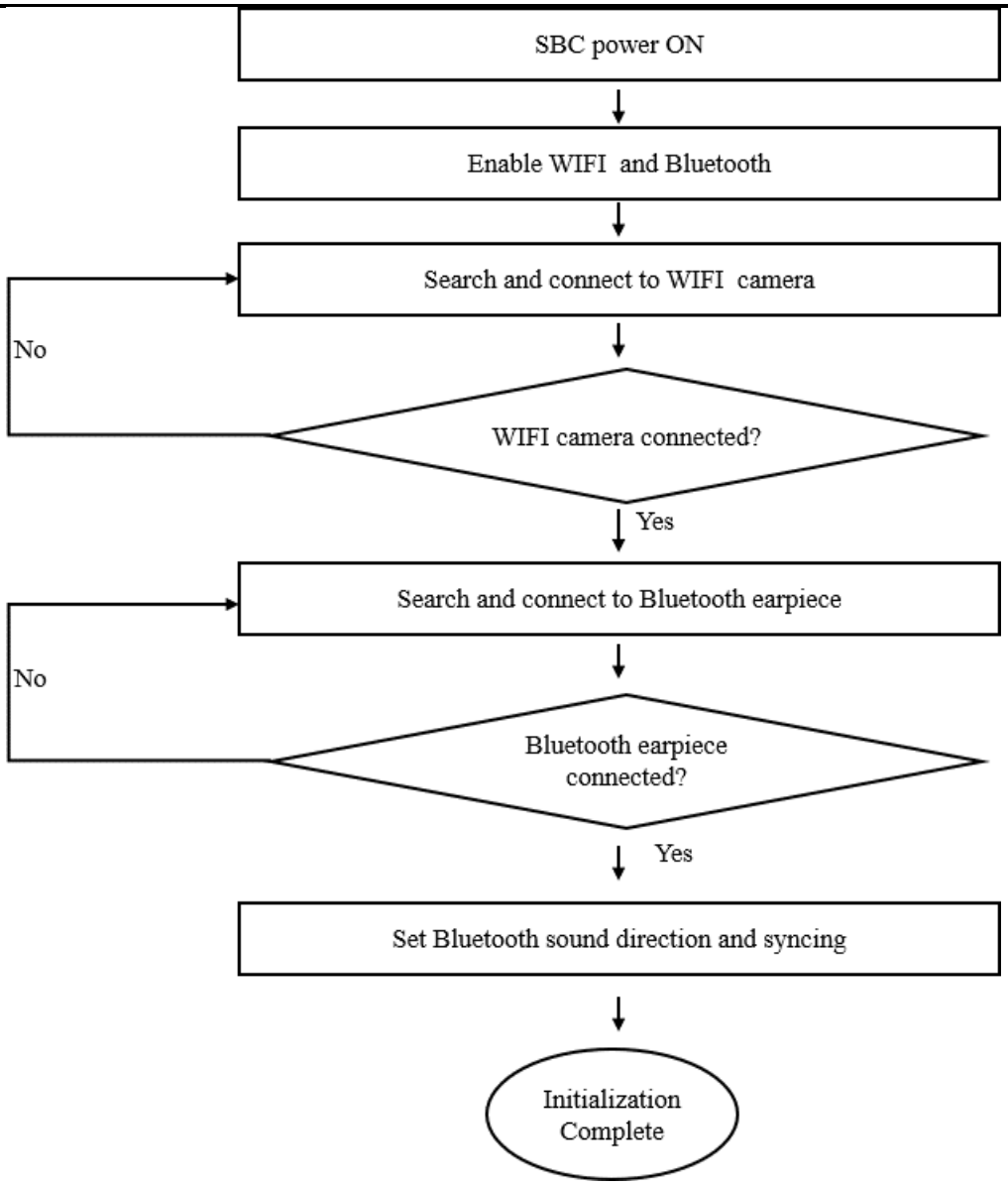


Figure 4.6. System initialization sequence.

4.8.1 Network connectivity for monitoring

It is crucial to set up a communication link with the system to verify it and deploy the software along with the necessary files. In this prototype, the easiest way to connect the development computer to the SBC is by using an Ethernet cable, as the board already has the port. The development computer was utilized to build the software that runs on the SBC. Connect the battery to the SBC and turn it on. Connect the ethernet cable from the development computer to the SBC. The Internet Protocol (IP) address of the SBC must be noted since it is the target device to connect to. The IP address of the SBC is static at

192.168.0.16 while the one for the development computer is set to 192.168.0.17. The development computer is running a Windows operating system and its IP address can be viewed by issuing the *ipconfic* command on the terminal. The SBC can display its IP on the terminal after issuing the *ifconfig*. The *ping* command can be issued from the development computer to verify connectivity to the SBC.

If the initial IP address of the SBC is not known, a software called *angry IP scanner* can be used to find the IP address. An alternative is to use a serial debugger (COM Port) of the SBC to find and set the IP address of the SBC.

Now that a communication link is established, it is time to log in to the SBC so that it can be configured to connect to both the WIFI camera and the Bluetooth earpiece. It is also necessary to deploy the developed CLI application into the SBC. Using the Windows terminal, type the command *ssh user@ip* where *user* is the account of the SBC and *ip* is the IP address of the SBC. The default username and password for this SBC are determined by the OS image, which is *rock64* and *rock64* respectively. Upon successful login, the SBC presents the current date and time, alongside the Central Processing Unit (CPU) architecture, identified as aarch64.

4.8.2 WIFI camera connectivity

To direct the SBC to connect to the WIFI camera, there is a need to run a scan to detect and list all available wireless networks. This can be done by issuing the command *nmcli d wifi list* into the SBC. The SBC uses the WIFI USB adapter, which is connected to the USB port to scan, detect and connect to wireless networks. Now that the SBC can see the wireless network, it is time to connect to the WIFI camera. This is archived by issuing the command *sudo nmcli dev wifi connect "C100+_GC000b6c4477ec" password "12345678"* into the SBC. This command connects the SBC to a wireless network *C100+_GC000b6c4477ec* with a password *12345678*. The *nmcli* utility comes pre-installed on most Linux OS distributions.

For this configuration to work, the camera must be turned on before issuing the above command. Depending on the manufacturer of the camera, it may be required to be put on discover mode too, that is, allowing the camera to be found on the network. For this project,

discovery mode is entered by double-clicking the small button in front of the camera. At this point, the SBC is connected to the wireless camera. The configuration is automatically saved by the network manager of the OS. This configuration is only done once. The SBC will, from now on, automatically connect to the camera if it is turned on and on the range with good signal strength. Independent of which device was turned on first between the SBC and the wireless camera, the connectivity between the two will now be automatic.

4.8.3 Bluetooth earpiece connectivity

The next step is to connect the SBC to the Bluetooth earpiece. Issuing the command *sudo bluetoothctl* into the SBC will enter the Bluetooth configuration mode. The SBC will use the connected Bluetooth USB adapter to discover and connect to Bluetooth devices. Issuing the above command will also list the Bluetooth devices that are already paired with the SBC independent of their current state (on or off). The Bluetooth address of the SBC is also displayed. Issuing the *scan on* command will scan for available Bluetooth devices. Issuing the *untrust E8:07:BF:00:52:18* command will unpair the device of the specified address. Issuing the *trust A0:E9:DB:50:E3:52* command will pair the device of the specified address. Once the devices are paired, they will automatically connect upon power on. Issuing the *connect A0:E9:DB:50:E3:52* command will connect the SBC to the device of the specified address. The SBC will automatically connect to the Bluetooth earpiece after powering on. The Bluetooth earpiece will automatically connect to the SBC after powering on, provided there is no other device paired with it nearby. The *bluetoothctl* utility comes pre-installed on most Linux OS distributions.

4.8.4 Bluetooth sound direction and syncing

Now that the Bluetooth link is established, the configuration needs to be done to direct the sound output into the Bluetooth device connected. This configuration is also done once. Issuing the *pactl set-default-sink bluez_sink.A0_E9_DB_50_E3_52.headset_head_unit* command sets the connected device to default sink audio playback as an output device. Now we can play a simple song to see if the configuration was a success. In this project, the command *mpg321* was used together with a full path to a media file. The file played and the sound came out of the Bluetooth earpiece as expected.

4.8.5 Software deployment

Now that the hardware connectivity is done, it is time to deploy the software that is capable of camera access, object detection, and audio playback. WinSCP is used to transfer files into the SBC using Secure File Transfer Protocol (SFTP). *yolact.bin* is the model file that contains the information about the trained objects. This file is supplied together with the default YOLACT library. *yolactstatic* is responsible for manually testing the developed software by supplying a static image file (frame), in which the software will perform object detection. *yolactUSB* is the software developed to test object detection from a USB camera without the need for a wireless camera. This was done to simplify the testing of the system. *yolactWIFI* is the software developed to use the WIFI camera to capture images for processing.

4.9 Object detection dataset and image size

The YOLACT object detection library includes a pre-trained model file, *yolact.bin*, which encompasses the classifications. Additionally, it contains a file named *yolact.param*, which holds the parameters used for training the model. The model file, approximately 70MB in size, includes classifications of the most common objects found in both indoor and outdoor environments. The TensorFlow object detection library includes a pre-trained model file, *detect.tflite*, which encompasses the classifications. Additionally, it contains a file named *COCO_labels.txt*, which contains 91 labels of the objects. The model, comprising a file size of approximately 4MB, encompasses a taxonomy of frequently observed objects across both indoor and outdoor environments.

In this project, training custom object detection models was excluded from the scope, as the primary focus was on developing software to ensure seamless communication among hardware components and overall system integration. Default pre-supplied datasets were therefore utilized, enabling a comparative evaluation of both algorithms in their baseline configurations.

The dimensions of the images, as presented in Table 4.2, were determined during the development phase. These dimensions were selected to achieve an optimal balance between image clarity, file size, and overall usability. For instance, reducing the resolution enhances processing speed; however, this reduction compromises image quality, resulting in a blurred image. Conversely, increasing the image dimensions leads to larger file sizes, which in turn slows down the processing speed during object detection. Nevertheless, the higher resolution images are clearer and more user-friendly. Therefore, it is crucial to strike a balance between these factors. The bit depth is the number of bits used to represent each pixel in an image.

Table 4.2: Image size properties.

Property	**Value**
Width	640 pixel
Height	360 pixel
Horizontal resolution	96 dpi (dots per inch)
Vertical resolution	96 dpi (dots per inch)
Bit depth	24

4.10 Text to speech sound signal

In the Festival text-to-speech system, the sound signal is typically a synthesized waveform generated from the text input. This process involves several steps. First, the input text is segmented into smaller units called utterances. Each utterance is then processed by a series of functions, including utt.synth, which synthesizes the text into a speech waveform. Finally, the synthesized waveform is played back using the utt.play function [52].

The audio files produced by Festival are in the Microsoft .WAV format with a sampling rate of 16 kHz [53]. Additionally, the developers of Festival provide audio samples in the .wav file format. In the context of this project, the audio is played through speakers and is not stored locally as .wav files. Figure 4.7 shows a waveform of the sample file “axk1.wav”.

Figure 4.7. Waveform of the sample file “axk1.wav”.

4.11 Summary

This chapter has provided a comprehensive exposition of the development process for both the hardware and software constituents of the proposed system. Additionally, we have elaborated on the various software features, provided detailed pseudocode for the proposed software, and thoroughly discussed the communication protocols utilized, specifically focusing on both WIFI and Bluetooth technologies. This holistic overview ensures a clear understanding of the system's architecture and operational mechanisms. The next chapter details the end-to-end testing procedures and reports the results obtained.

CHAPTER 5 Testing and Results

5.1 Introduction

The system developed to assist VI individuals is undergoing testing in this chapter. This chapter delineates the steps taken for end-to-end testing of the system, as well as presents the results of the conducted tests.

5.2 System testing

Tests were conducted to understand the accuracy and reliability of the system. Because the scene changes, a strategy was needed to capture the images in a way that can be analyzed and reused by a different application. The same image needs to be processed by YOLACT and TensorFlow for a fair comparison. That is why a static version of the application was developed purely for this kind of testing prior to using the WIFI version. The static version of the application is available for both YOLACT and TensorFlow. Another minimal application named *imagecapture* was developed to capture images while the user is walking and save them on the onboard storage which is then processed by both YOLACT and TensorFlow.

5.2.1 Image dump from WIFI camera

Powering on the SBC, WIFI camera, and Bluetooth earpiece initializes the system. Navigating to folder *imagecapture* on the SBC and executing the *run* application images are periodically captured every 5 seconds and saved on the *images* folder. It is important to indicate that this image dump testing is purely for feeding the same images to both the YOLACT based and TensorFlow based applications for a fair comparison. In a real-world environment, there is no need to run the imagecapture application as the tests would have concluded.

5.2.2 Object detection using the proposed methodology

The first object detection testing is done by supplying the images captured in Figure 5.1 into the YOLACT-based command line application. The application loads both the model file and the supplied image from the local storage. The application begins object detection and saves the resulting file into the relative path of the application. The resulting file shows the object(s) detected and the accuracy level in percentage. In this test case, no filtering was done, hence the low accuracy levels are also shown. In the live system testing, the filtering is done to ignore objects with an accuracy level of less than 40%. Filtering comes with the benefits of reduced false alarms together with a reduced delay. Blurry objects, for example, may have a low accuracy level compared to clear objects in a scene. The user is only notified when the system is confident enough of the results received, that is, an accuracy level of not less than 40%.

From the image below, YOLACT can detect objects from an image that are then marked as obstacles to the VI people. We can see that objects that are far or with low resolution are also identified with low accuracy results.

Figure 5.1. YOLACT object detection with a manually supplied image file.

The next step is to check the output file size to better understand the algorithm. Figure 5.2 shows a standard input file ready for processing. The input file size is 181kb in this case. This file is carefully selected as it shows objects that are far and close. The purpose of this test is to compare the file size of the output file after processing. The YOLACT algorithm draws additional details on the images to identify objects by creating both labels and bound boxes. This results in the creation of a new result file.

Figure 5.2. Input file of 181kb before processing.

Figure 5.3 shows the output file after processing. Objects are marked and named. The output file is 1.40MB in size. That is an increase of approximately 800% in size after processing. This is the reason it is important to overwrite the files with the new ones for every scene as it ensures that the system cannot run out of capacity when in use.

Figure 5.3. Output file of 1.40MB after processing.

With the solution implemented using two different object detection algorithms (YOLACT and TensorFlow), it is time to compare the reliability of both algorithms. An image of a scene with complex real-life objects was supplied to both algorithms. Figure 5.4 shows an image fed into the TensorFlow-based application. TensorFlow was able to identify some of the objects in a scene reasonably. However, it had false alarms. For example, it failed to identify a *fire hydrant* in this case confusing it with a *person*. Also, TensorFlow failed to identify a stop sign in this case possibly because it was surrounded by other road signs. Table 5.1 shows the results obtained when the TensorFlow-based application was used. The confidence level indicates the detection accuracy for each object, expressed as a percentage, in both algorithms. From the table, we can see the maximum confidence level was 70% while the minimum was 0%. This image is of a real-life scene; therefore, it is fair to supply the same image to the YOLACT based application and monitor the results.

Figure 5.4. Object detection using TensorFlow.

Table 5.1 YOLACT vs TensorFlow results.

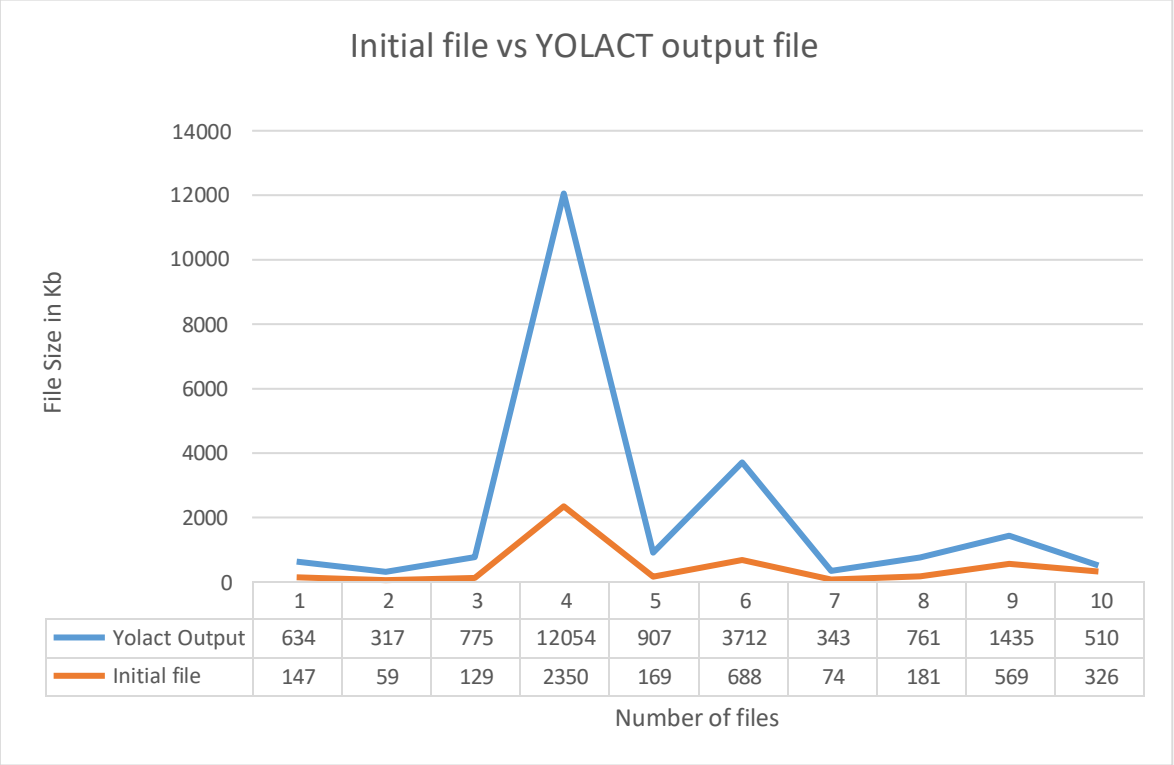
Object of interest	Confidence Level		Comments
	YOLACT	TensorFlow	
Fire Hydrant	48.2%	0.0%	Due to the shape of the fire hydrant, TensorFlow identified it as a person, which is wrong. Hence 0.0%
Stop Sign	99.9%	0.0%	While the stop sign is big and visible, it is slightly tilt, TensorFlow could not identify it. Hence 0.0%
Car 1 (Left)	94.7%	52.0%	Both algorithms were able to identify this object with YOLACT at a high confidence level.
Car 2 (Middle)	95.2%	70.0%	YOLACT has high confidence level.
Car 3 (Right)	66.8%	59.0%	YOLACT has high confidence level.

The same image was supplied as input into the YOLACT-based application. The resulting file is shown in Figure 5.5. Zooming the image provides better readability to the confidence level obtained by the algorithm. YOLACT was able to identify all five objects of interest with a better confidence level. The maximum level was 99.9% while the minimum was 48.2%. YOLACT performed better compared to TensorFlow; however, TensorFlow was always faster than YOLACT in all the test cases. Table 5.2 shows the results obtained when the YOLACT-based application compared with the TensorFlow algorithm. Looking at the stop sign, for example, it is tilted a bit and surrounded by other road signs, but YOLACT was still able to recognize it with 99.9% confidence while TensorFlow failed. It is not that TensorFlow does not know the *stop sign*. The model file supplied contains a *stop sign* as one of the objects that TensorFlow can detect. Moreover, it detects it when the scene is not complex, and when few objects are present.

Figure 5.5. Object detection using YOLACT.

While tests were conducted, it was important to note the file sizes of both the input files and the output files. Output files are the ones generated by the object detection algorithm containing both bound boxes and the labels of the objects found in the scene. Initial file is the image file of the frame captured by the wireless camera and saved in the local storage

before processing. Figure 5.6 shows that in all cases, the output file size generated by the algorithm is directly proportional to the initial file size, and is always higher than the initial file. While multiple images of different sizes were selected for this test case, it is important to indicate that images generated by the system also result in the same behavior. It was also observed that images taken in low-light environments tend to be smaller in size compared to images taken in well-lit environments. This is discussed further in section 5.2.3.



	1	2	3	4	5	6	7	8	9	10
Yolact Output	634	317	775	12054	907	3712	343	761	1435	510
Initial file	147	59	129	2350	169	688	74	181	569	326

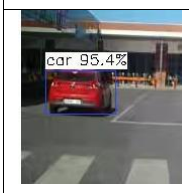
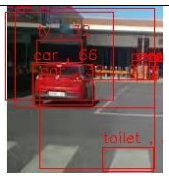
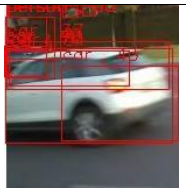
Figure 5.6. File size comparison.

5.2.3 Additional experiments

More tests were conducted, and the results are shown in Table 5.2. The confidence level indicates the detection accuracy for each object, expressed as a percentage, in both algorithms. TensorFlow had outstanding performance compared to YOLACT, but the results were not impressive. In most cases, TensorFlow identified objects that did not exist in the scene, which means more false alarms. Another situation with TensorFlow is that; it breaks down a single object into multiple objects. This behavior can be noticed on the table where the object of interest is a *Person, Bicycle*, *Car*, or *Bus*.

Table 5.3 presents a comparison between the objects present in the scene and those detected by TensorFlow. It is observed that, although TensorFlow successfully identified several objects that were indeed present, it also erroneously detected objects that did not exist in the scene, leading to false alarms. This misidentification highlights the limitations in the accuracy of TensorFlow object detection capabilities in this case.

Table 5.2 YOLACT vs TensorFlow test results.

Captured images		Object of interest	Confidence level (%)		Delay time (sec)	
YOLACT object detection	**TensorFlow object detection**		**YOLACT**	**Tensor Flow**	**YOLACT**	**Tensor Flow**
		Person	99.4	73.0	5.071	0.181
		Car	95.4	66.0	5.791	0.095
		Vehicle in motion	63.3	43.0	4.99	0.095
		Bus	90.4	55.0	6.586	0.101
		Traffic light	87.8	64.0	7.549	0.113
		Motorcycle	99.4	50.0	6.742	0.2
		Bicycle	99.5	43	5.1	0.114
		Fire Hydrant	80.8	82	8.596	0.586

Captured images		Object of interest	Confidence level (%)		Delay time (sec)	
YOLACT object detection	TensorFlow object detection		YOLACT	Tensor Flow	YOLACT	Tensor Flow
		Stop Sign	99.9	79	8.029	0.385
		Complex Scene: Stop Sign	99.9	0	9.315	0.189
		Complex Scene: Fire Hydrant	48.2	0	9.315	0.189
		Complex Scene: Traffic light	87.8	64	7.549	0.113

Table 5.3 TensorFlow false object detection.

Scene number	Actual objects in a scene	TensorFlow detected objects
1	• Delivery motocycles	• Motocycle • Laptop • Person • Suitecase
2	• Cars • Trees • Stop Sign • Fire hydrant	• Person(s) • Car • Stop sign
3	• Sidewalk • Fire hydrant • Fast moving cars • Dry trees	• Umbrella • Person(s) • Cars
4	• Door • Windows	• Refrigerator • Person(s)

5.2.4 Poor quality comparison

Some tests were performed where the input image was blurry. It is possible that due to too much camera movement, the images generated become burry. This scenario is simulated by using software to blur an image, and then supply both images (clear and blurred) into each object detection algorithm. This method allows supplying the same images into both algorithms without the need to recapture the same scene. It is difficult to capture the same scene unless all the objects are stationary, and the environment brightness is constant. Figure 5.7 (a) shows the comparison where both images are fed into the YOLACT-based application. From the clear image, only one false alarm was detected due to the shadow of the tree showing on the frame. From the blurred image, YOLACT identified the *bus* as a *car* with 51.1%. For each object detected, a border surrounding that object is applied to indicate the location of the object in the scene.

(a) YOLACT image object detection

(b) Tensorflow image object detection

Figure 5.7. Burry image comparison.

The same process was followed by supplying both images to the TensorFlow-based application. Observing the border lines surrounding the object detected, there is a clear indication that more false alarms were generated by the application. The *bus* was identified as a *truck* for example. Also, some objects were identified as *persons*, yet they are not. Not a single object was correctly identified for the blurry image by the TensorFlow-based application. All objects identified were false positives in this case.

5.2.5 Different light conditions delays

The integrated system underwent empirical evaluation, with specific tests designed to assess the impact of varying illumination conditions on system latency. Table 5.4 presents a comparative analysis of delay times, the quantity of detected objects, image file sizes, and output latency observed under both low-light and well-lit environmental scenarios. In well-lit scenes, the software detected 6 objects with a 10.19 seconds delay. In low-light scenes, it detected 8 objects with a 9.63 seconds delay. This test was performed only in YOLACT with each image supplied per application. That is, the application only loaded the model file to process a single static image file, hence both the delays appear to be large. In real-life scenarios, the model file is only loaded once but reused for all newly captured scenes, hence the delay reduces depending on the complexity of such scenes. In a well-illuminated environment, the image pixel density increases, resulting in larger file sizes and higher processing latency. Additionally, the latency is particularly high due to the processing that occurs on the CPU. A system equipped with a dedicated GPU may reduce latency for the same image processing tasks, resulting in improved processing speed. This is because a GPU is more adept at handling machine learning and video processing.

Table 5.4 Processed data for different light conditions.

Scene	Number of objects detected	Output image size (kilobytes)	Latency (sec)
Well-lit environment	6	1400	10.19
Poorly-lit environment	8	914	9.63

5.2.6 Live testing with sounds

The live system was tested where a YOLACT-based application connected to the WIFI camera instead of a static file. The audio feedback feature is enabled on the live system (final prototype). The changes are made to ignore objects with a confidence level of less than 40%. When the application *yolactWIFI* is executed, it starts searching for the active camera using an IP address and captures the image for processing. Table 5.5 shows the live application executed and operating in a Linux terminal for three sequential frames. The first delay was 5.883 seconds where only two objects with a confidence level of more or equal to 40%. The first object was 99.458% while the second object was 53.681% which resulted in an average confidence level of 76.56%. The next two scenes are also shown in the table with an average confidence level of 72.98% and 71.78% respectively. For each object detected, a sound is produced on a Bluetooth earpiece saying '*there is a person ahead'* if the object is a *person*. The delay shown in this table does not account for the delay of the sound pronunciation; that is, the time it takes to say the sentence. The delay varies depending on multiple factors such as the number of objects in a scene, the quality of the scene, the brightness of the scene, and the hardware processing power.

Table 5.5 Execution delay and average confidence level.

Scene Number	Delay (Seconds)	Number of objects detected	Average confidence level (%)
1	5.883	2	76.56
2	5.933	2	72.98
3	6.983	3	71.78

5.2.7 System delays

There are two types of delays noted in this system. The first delay is the time it takes the system to identify the objects in a scene and return an accuracy level. The second delay is the time it takes the system to produce a sentence notifying the user of the object ahead. There are further steps that can be taken to reduce the first delay as discussed in chapter 6. However, there is little to no action that can be taken against the second delay as it will only increase or decrease the speed at which a sentence is spoken. For this system to be usable, all sentences are spoken at a normal speed so that a user can clearly understand what the system says. Shortening the sentence automatically reduces the second delay while expanding the sentence does the opposite. It takes longer to say '*There is a person ahead*' than to say '*Person ahead'*.

5.3 Summary

Tests were performed in this chapter where two applications with different object detection algorithms were compared. From the outcome, it was noted that YOLACT was more reliable but slow, while TensorFlow was less reliable but fast. The audio feedback also proved to be reliable. In the next chapter, we will discuss the results obtained in this chapter.

CHAPTER 6 Discussion of Results

6.1 Introduction

In this chapter, we discuss the results obtained in chapter 5, where we focus at the reliability level of the system. This chapter is organized in sections as follows: the dataset results are discussed in 6.2, the system performance and dependencies are discussed in 6.3, feature classification results are discussed in 6.4, interior testing results are discussed in 6.5, sudden brightness results are discussed in 6.6, and finally the summary in 6.7.

6.2 Dataset results

More than 1000 images were captured by the wireless camera while walking the street and stored on the onboard storage for processing. These images were captured for static testing. Some images were discarded during testing as the scene was the same resulting in redundant processing. Each image captured by the camera was of a dimension 640 x 360, that is a width of 640 pixels and a height of 360 pixels. The real-time live application was only working with one image file name to save the capacity of the storage and improve the performance. The camera was capturing a frame to overwrite the previously processed scene. During testing, another 10 different images of different dimensions were downloaded from the internet and fed into the application for processing. The purpose of this download was because the end-user may have a different environment and different condition than the one used for testing. Images captured by the WIFI camera can range from a size of 23kb to 98kb depending on factors such as scene complexity and weather conditions.

6.3 System performance and dependencies

From the portability of the system, it is clear the sub-systems are interdependent. The SBC cannot operate if the camera is offline. Also, the Bluetooth earpiece cannot produce any sound if no object(s) is detected. All three major components of the system need to be powered on for the system to fully operate. Powering on the SBC takes approximately 20 seconds to initialize the overall system where the first 10 seconds is to initialize the base Linux operating system while the other 10 seconds is for connecting to the Bluetooth earpiece and the WIFI camera with an assumption that the camera is already powered on together with the Bluetooth earpiece. The CLI applications developed in this project have a faster startup. That is because CLI applications are generally faster than GUI (Graphical User Interface) applications. Launching a CLI application results in an instant execution.

6.4 Feature classification results

From the tests conducted in Chapter 5, the YOLACT-based application was able to classify objects with better accuracy than TensorFlow. The TensorFlow algorithm frequently exhibited a tendency to decompose recognized objects into spurious sub-components not present within the visual field. YOLACT was slow during classification while TensorFlow was fast. This project prioritized reliability over speed, resulting in YOLACT being used as the main object detection algorithm in this research.

6.5 Interior testing results

During system development and testing, the object detection system was launched while inside the building to understand how it behaves. Figure 6.1 shows a scene with in-house objects such as a chair, TV, laptop, keyboard, and mouse. While the system is not intended to be used inside an office or house, it is still capable of detecting such objects because the default model file contains such objects. Therefore, it is clear that if needed, the system can be improved and optimized to guide the VI people of objects inside the building.

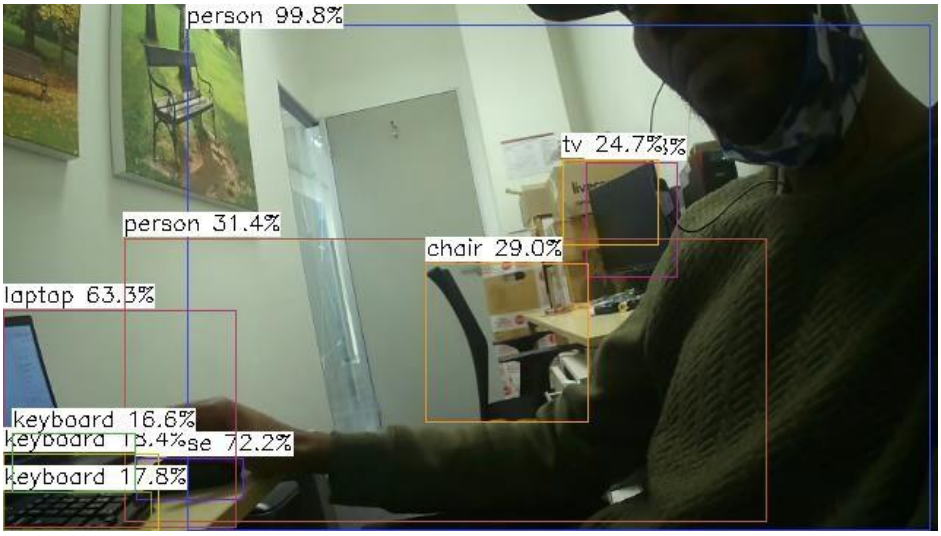


Figure 6.1. Interior object detection.

6.6 Sudden brightness

The system proposed in this research is developed to be used while walking in the street. It can occur that while using the system, a sudden bright light may directly face the camera. It could be direct sunlight, a building light bult brightness, or a vehicle headlight. Figure 6.2 shows a scene where some areas are nearly invisible because of the light directed to the camera. This scene is presented because it was not intentionally captured. Meaning, while the system is in use, a similar scene but a different environment may be captured by the camera. From the picture, we can see that the application was able to identify the object with a reduced confidence level correctly. Therefore, it is clear that, while the system is not intended to be used in direct brightness, it can still detect objects when such an event occurs but with a reduced confidence level.

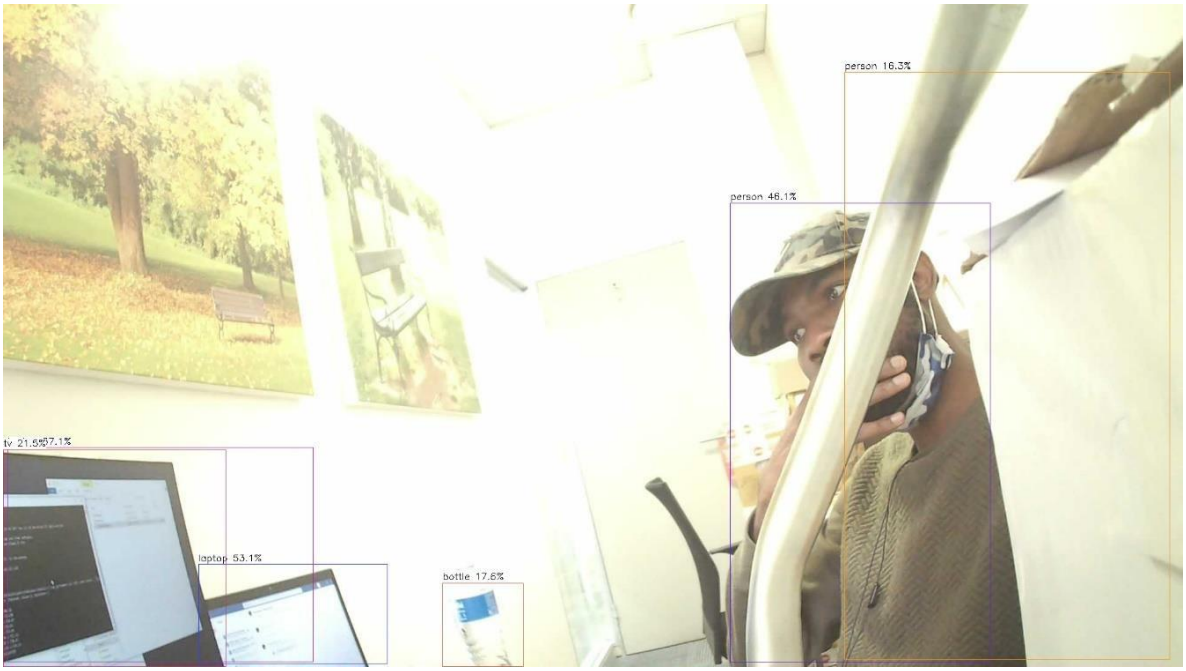


Figure 6.2. Light brightness directed to a camera.

6.7 Summary

Chapter 6 discusses the main results obtained in Chapter 5. From the discussion, we can see that it is possible to re-purpose the entire system for a different purpose such as interior movement guidance and object detection. In the next chapter, we will look at the recommendations that could be used to improve the performance of the YOLACT-based application. The next chapter will also conclude this research.

CHAPTER 7 Conclusion

7.1 Introduction

This chapter begins with an overview of the research challenges and limitations, followed by the recommendations of the study on computer vision and object detection for the visually impaired. It concludes with the findings of the study.

7.2 Research challenges and limitations

The limitation of studies related to object detection and navigation using wireless technologies poses challenges for this study and experiment. Here are some of the challenges in this research:

7.2.1 Moving objects

The system can detect objects that are either moving or stationary. However, by the time the user is informed of an object ahead, it may have already moved by the time the user arrives at that location. In this project, the issue of moving objects was not considered. The user will be notified as new objects are detected in the scene.

7.2.2 System testing

Since the object detection algorithm requires the same frame or image for testing, it is nearly impossible to capture the same scene twice without introducing new properties. This makes it difficult to make fair comparisons between the results. The challenges of varying light conditions, weather, and shadows were considered in this research. To address this, a static application was developed to simply capture and store images. Once an image is stored, it can then be passed to other applications for processing.

7.2.3 System complexity

Due to the complexity of the system, it is difficult to implement all components to operate at their best within the time limit of the project. The first step is to develop a system or script that can wirelessly connect the individual hardware components before creating the object detection system. Hence, this project utilizes the standard model file provided with the library for each object detection algorithm, eliminating the need to conduct the training stage.

7.3 Recommendations

An object detection system was developed and tested in the previous chapters to assist VI people by identifying objects and providing an audio feedback mechanism. Here are some recommendations that can be taken to improve the proposed system.

7.3.1 YOLACT delay reduction

During testing, it was observed that the YOLACT-based application demonstrated greater reliability but exhibited reduced performance, resulting in higher delays compared to the TensorFlow-based application. Several measures can be implemented to mitigate application delay. The current hardware utilized in this project is limited, as it lacks a dedicated GPU, leading to increased delays. Employing more advanced GPU hardware can reduce these delays. Upgrading the hardware of the project can enhance application performance but may also increase the weight of the system. However, given the portability of the system, the SBC can be carried easily, rendering the weight increase negligible. Additionally, optimizing the YOLACT application source code, such as through the implementation of parallel programming, can further improve performance.

Parallel programming is a way of breaking down large tasks into smaller ones and executing them simultaneously to reduce the amount of time it takes for such an operation to complete. It should be noted that parallel programming requires the processor to have more than one core to function. Alternatively, concurrency can be employed; however, it is essential to consider and mitigate the potential for deadlocks.

7.3.2 Camera stabilization

During testing, it was observed that the camera may produce blurry images when there is excessive movement, such as fast walking, jumping, or running by the end user. Although the primary users are not expected to move quickly, there are situations where they might engage in fast movement. In such cases, it is advisable to use a camera with built-in hardware stabilization. Alternatively, pre-processing the images before object detection can help to reduce blurriness. This can be particularly effective if the software is implemented using parallel programming techniques.

7.3.3 Avoiding false positives

During testing, we observed that the system may incorrectly identify objects in the scene. For instance, in the TensorFlow-based application, it tends to identify objects that do not actually exist, resulting in false positives. To address this issue, we can set an acceptable threshold. This means that the user will only be notified if the confidence level is above a certain percentage, for example, 40%. This not only helps in ignoring low confidence levels but also reduces the processing delay for the next scene. By ignoring certain objects, we can save time by not translating them into audio.

7.3.4 Reducing the weight of the system

The SBC used in this project is designed to be generic, meaning it can be used in almost any project. However, this leads to the issue of certain features being enabled in the SBC that are not useful for this specific project. While it works well for testing purposes, for mass production, it is advisable to create a custom SBC tailored specifically for this project, as it will help reduce the weight. For instance, the final product does not require an Ethernet port. Instead of using two USB ports for WiFi and Bluetooth, it is suggested to use a single-chip solution like NXP iw612 in conjunction with the main ARM processor. While there are other single-chip solutions available, IW612 chipset has been identified in this project as a recommended option. This recommendation also reduces the overall size of the circuit.

7.3.5 Removing dependencies

During the development of this project, three wireless cameras were tested before deciding on the final one. It was noted that the manufacturers of the Wi-Fi cameras tend to lock the camera so that it can only be operated using their mobile application. This means that even the information needed to connect to the camera may be kept confidential. It is recommended to develop the camera in-house using a single chip like ESP32 to remove dependencies. This way also removes the need for additional tools that are used to detect the current IP address of the WIFI camera. This recommended method allows configuring the camera to meet specific requirements, such as battery size, image size, and zoom level.

7.4 Research questions answered

7.4.1 Can a portable modular-based computer vision system be developed to detect and identify objects?

The findings of this study confirm the feasibility of developing a portable, modular, unobtrusive, and low-cost computer vision system for object detection and identification. The modular hardware architecture, combined with wireless communication protocols, enabled the creation of a lightweight, body-mountable device suitable for outdoor navigation by individuals with visual impairments.

7.4.2 How reliable will it be in object detection and classification?

Reliability was demonstrated through comparative analysis of object identification frameworks. The YOLACT algorithm consistently outperformed TensorFlow across diverse environmental conditions and image qualities, indicating strong robustness in detection and classification. However, increased processing latency was observed, which may affect real-time responsiveness. Thus, while the system is reliable in terms of accuracy, further optimization is required to balance processing speed and image resolution for enhanced operational reliability.

7.4.3 Is real-time auditory feedback of detected objects possible with the proposed hardware framework?

Yes. The system integrates real-time object detection with auditory feedback delivered via

Bluetooth earpiece. The modular hardware and wireless communication framework support low-latency audio transmission, enabling users to receive immediate feedback regarding obstacles in their path. Minimizing latency in the detection-to-audio pipeline remains essential to ensure effective real-time performance in dynamic outdoor environments.

7.4.4 How long will the portable system function given the specified battery hardware?

Battery performance analysis reveals that the overall system runtime is constrained by the component with the shortest endurance. The Wi-Fi camera, equipped with a 730 mAh battery, sustains continuous operation for approximately 2.5 hours under its specified conditions. In contrast, the Bluetooth earpiece, powered by a 160 mAh battery, achieves an estimated 20 hours of operation, while the SBC, supplied by a 10000 mAh power bank, delivers approximately 5.3 hours of runtime depending on computational load. Consequently, the effective system endurance is limited to around 2 hours, as the specified operation times do not fully account for real-world usage scenarios. Extending runtime would therefore require either supplementing the camera with a higher-capacity battery or implementing power optimization strategies to reduce its consumption.

7.5 Conclusion

This research investigated the development of an autonomous machine vision system designed to enhance outdoor navigation for individuals with visual impairments. The proposed system employs real-time object detection to provide auditory cues regarding the presence of obstacles in the user's path. A preliminary evaluation of the developed prototype indicates the feasibility of implementing a portable, modular, unobtrusive, and reliable low-cost computer vision solution. The modular hardware architecture of the system facilitated the creation of a lightweight, body-mountable device, with wireless communication protocols employed to eliminate physical interconnections between components. Comparative analysis of object identification software revealed that the YOLACT algorithm outperformed TensorFlow across diverse environmental conditions and image qualities, albeit with observed increases in processing latency. Further investigation is warranted to optimize the balance between processing speed and image resolution, thereby achieving accurate and timely object detection.

CHAPTER 8 References

[1] World Health Organization, "Blindness and vision impairment", 2021. [eBook] Available at:https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment.

[2] M. Andjelkovic, V. Vučinić, M. Gligorović, and J. Maksic, “The practical skills of persons with vision impairment”, Vojnosanitetski pregled, 2022, pp.101-101. 10.2298/VSP210328101A.

[3] R. Khlaikhayai, C. Pavaganun, B. Mangalabruks and P. Yupapin, "An Intelligent Walking Stick for Elderly and Blind Safety Protection", Procedia Engineering, vol. 8, 2011, pp.313-316

[4] N. Ayat, M. Samiam, F. Mahmoud and S. Ahmed, “Effective Fast Response Smart Stick for Blind People”, 2015, 10.15224/978-1-63248-043-9-29.

[5] E. Amira, “Smart Blind Stick Design and Implementation”, International Journal of Engineering and Advanced Technology. Vol.10, 2021, pp.17-20. 10.35940/ijeat.D2535.0610521.

[6] S. Sirouspour, E. Hossain, R. Khan, R. Muhida and A Ali, "Analysis and Implementation for a Walking Support System for Visually Impaired People", International Journal of Intelligent Mechatronics and Robotics, vol. 1, 2011, pp.46-62.

[7] M.D.C. Perera, M.I.M. Amjath, H.M.S.C.R. Heenkenda and V. Senthooran, "Assist System for Visually Impaired People to Recognize Pure Colours", Trincomalee International Conference, 2017.

[8] G.S. Aakash Krishna, V.N. Pon, S. Rai and A. Baskar, "Vision System with 3D Audio Feedback to assist Navigation for Visually Impaired", Procedia Computer Science, vol. 167, 2020, pp.235-243.

[9] OpenCV. (2018). “About OpenCV.” [online] https://opencv.org/about/.

[10] S. Sivkov, L. Novikov, G. Romanova, A. Romanova, D. Vaganov, M. Valitov and S. Vasilie, “The algorithm development for operation of a computer vision system via the OpenCV library”, Procedia Computer Science. vol.169, 2020, pp.662-667. 10.1016/j.procs.2020.02.193.

[11] www.softintegration.com, “Ch -- an embeddable C/C++ interpreter, C and C++ scripting language. [online] Available at: https://www.softintegration.com [Accessed 2 Feb 2022].

[12] O. Yu, H. Cheng, W. Cheng and X. Zhou. “Ch OpenCV for interactive open architecture computer vision”, Advances in Engineering Software. vol35, 2004, 10.1016/j.advengsoft.2004.05.003.

[13] C.P.G. Mallikarjuna , R. Hajare and P.S.S. Pavan, "Cognitive IoT System for visually impaired: Machine Learning Approach", Materials Today: Proceedings, vol. 49, 2022, pp.529-535.

[14] N. Kanwal, “A Navigation System for Visually Impaired: A fusion of Vision and Depth Sensor”, Applied Bionics and Biomechanics, 2015. 10.1155/2015/479857.

[15] J. Jordan, “An overview of object detection: one-stage methods.” 2018. [online] Available at: https://www.jeremyjordan.me/object-detection-one-stage.

[16] G. Boesch, “Object Detection in 2021: The Definitive Guide.” 2021. [online] Available at: https://viso.ai/deep-learning/object-detection.

[17] S. Lin, K. Zhu, C. Feng, and Z. Chen, “Align-Yolact: a one-stage semantic segmentation network for real-time object detection.” Journal of Ambient Intelligence and Humanized Computing. 2021. doi:https://doi.org/10.1007/s12652-021-03340-4.

[18] A. Sinha, “YOLACT : Real Time Instance segmentation”. 2022. [online] Available at: https://medium.com/@asmitasinha/yolact-real-time-instance-segmentation-1-af0c27ee0bbc [Accessed 29 May 2024].

[19] D. Bolya, C. Zhou, F. Xiao and Y. J. Lee, "YOLACT: Real-Time Instance Segmentation," 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 9156-9165

[20] R. Perera, “YOLOACT with Jupyter Notebook.” 2021. [online] Available at: https://medium.com/mediio/yoloact-with-jupyter-notebook-b8c65bf788ee [Accessed 29 May 2024].

[21] A.M. Aromal, “Protonet in Yolact and Yolact++.” 2022. [online] Medium. Available at: https://medium.com/@aromalma/protonet-in-yolact-and-yolact-eb39f4f29809 [Accessed 29 May 2024].

[22] W. Dong, Z. Liu, M. Yang, and Y. Wu, “FIR-YOLACT: Fusion of ICIoU and Res2Net for YOLACT on Real-Time Vehicle Instance Segmentation.” Computers, Materials & Continua, 77(3), pp.3551–3572. 2023. doi:https://doi.org/10.32604/cmc.2023.044967.

[23] A. Dua. “YOLACT (Important Points).” 2019. [online] Medium. Available at: https://medium.com/@anmoldua/yolact-important-points-125268f475d0 [Accessed 29 May 2024].

[24] D. Bolya, C. Zhou, F. Xiao and Y. J. Lee, "YOLACT: Real-Time Instance Segmentation," 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 9156-9165

[25] “YOLACT - Real Time Instance Segmentation.” [online] Available at: https://www.augmentedstartups.com/blog/yolact-real-time-instance-segmentation [Accessed 29 May 2024].

[26] math.paperswithcode.com, “ Papers with Code - ResNet Explained.” [online] Available at: https://math.paperswithcode.com/method/resnet [Accessed 29 May 2024].

[27] GeeksforGeeks. “What Is OpenCV Library?” [online] Available at: www.geeksforgeeks.org/opencv-overview [Accessed 1 Sept. 2024].

[28] Geeks, Python. “What Is OpenCV? - an Introduction Guide.” [online] Available at: https://pythongeeks.org/what-is-opencv [Accessed 1 Sept. 2024].

[29] G. Isha. “Speech Synthesis. Speech Synthesis Technology”. 2011. [online] Available at: https://www.researchgate.net/publication/288630068_Speech_Synthesis [Accessed 1 Sept. 2024].

[30] P. Rubin, T. Baer, P. Mermelstein, "An articulatory synthesizer for perceptual research". Journal of the Acoustical Society of America. 70 (2): 321–328. Bibcode:1981ASAJ...70..321R. 1981. doi:10.1121/1.386780.

[31] S. Gopal, M. Trishant, and D. Seema, "A simple phoneme based speech recognition system." International Journal of Modern Communication Technologies and Research, vol. 2, no. 4, Apr. 2014.

[32] D. Schwarz, "Current research in Concatenative Sound Synthesis, " Proceedings of the International Computer Music Conference (ICMC), 2005.

[33] D. Schwarz, “Data-Driven Concatenative Sound Synthesis.” 2004

[34] “What is Concatenative Synthesis.” [online] Available at: https://www.activeloop.ai/resources/glossary/concatenative-synthesis/ [Accessed 29 May 2024].

[35] Audiofanzine, “Formant and Linear Arithmetic Synthesis.” [online] Available at: https://en.audiofanzine.com/sound-synthesis/editorial/articles/formant-and-linear-arithmetic-synthesis.html [Accessed 29 May 2024].

[36] M. Hasegawa-Johnson. “Transformation of formants for voice conversion using artificial neural networks.” 2022. [online] Available at: https://courses.engr.illinois.edu/ece537/fa2022/slides/lec17.pdf [Accessed 29 May 2024].

[37] P. Taylor, R. Caley and H. Zen, “The Festival Speech Synthesis System”, 1997. [online] Available at: https://www.cstr.ed.ac.uk/projects/festival/ [Accessed 29 May 2024].

[38] R.A.J. Clark, K. Richmond and S. King, “Festival 2 - Build your own general purpose unit selection speech synthesiser”, 5th ISCA Speech Synthesis Workshop, 2004, pp.173-178

[39] P. Taylor, A. Black and R. Caley, “The architecture of the Festival speech synthesis system”, The Third ESCA Workshop in Speech Synthesis, 1998, pp. 147–151

[40] R.A.J. Clark, K. Richmond and S. King, “Multisyn: Open-domain unit selection for the Festival speech synthesis system”, Speech Communicatison, vol. 49, 2007, pp.317-330

[41] L. R. Rabiner, and R.W. Schafer, “Digital Processing of Speech Signals.” Prentice-Hall. 1978

[42] S. Davis0 and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences.” IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(4), 357-366. 1980

[43] L.R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition. “Proceedings of the IEEE, 77(2), 257-286. 1989.

[44] H. Sakoe, and S. Chiba, “Dynamic programming algorithm optimization for spoken word recognition.” IEEE Transactions on Acoustics, Speech, and Signal Processing, 26(1), 43-49. 1978.

[45] K. Sookdeo, “The Evolution of Wi-Fi Technology and Standards.” *IEEE Standards Association*, 16 May 2023, standards.ieee.org/beyond-standards/the-evolution-of-wi-fi-technology-and-standards/.

[46] “Understanding IEEE 802.11 and Wi-Fi Standards.” Www.microwavejournal.com. [online] Available at: www.microwavejournal.com/articles/34582-understanding-ieee-80211-and-wi-fi- standards [Accessed 29 May 2024].

[47] D. Niyato, D. I. Kim, Z. Han and M. Maso, "Wireless powered communication networks: architectures, protocol designs, and standardization [Guest Editorial]," in IEEE Wireless Communications, vol. 23, no. 2, pp. 8-9, April 2016, doi: 10.1109/MWC.2016.7462479.

[48] A. Huang, and R. Larry. “Bluetooth for Programmers.”, 2005, Available at: https://people.csail.mit.edu/rudolph/Teaching/Articles/BTBook.pdf

[49] R. Bruno, “Bluetooth: Architecture, Protocols and Scheduling Algorithms.” *Cluster Computing*,2002, www.academia.edu/4611651/Bluetooth_Architecture_Protocols_and_Scheduling_Algorithms. Accessed 4 Sept. 2024.

[50] R. Bruno, M. Conti, and E. Gregori, “Bluetooth: Architecture, Protocols and Scheduling Algorithms.” *Cluster Computing* **5**, 117–131, 2002. https://doi.org/10.1023/A:1013989524865

[51] R. Bruno, M. Conti, and E. Gregori, “Bluetooth: Architecture, Protocols and Scheduling Algorithms.” Cluster Computing, 5(2), 117-131. 2004.

[52] “Festival Speech Synthesis System - 29 Examples.” Festvox.org, 2024. [online] Available at: festvox.org/docs/manual-1.4.3/festival_29.html [Accessed 2 Sept. 2024].

[53] “Festival: General Demo.” Cmu.edu, 2024. [online] Available at: www.cs.cmu.edu/~awb/festival_demos/general.html [Accessed 2 Sept. 2024].

[54] Z. Zhang, S. Huang, X. Liu, B. Zhang and D. Dong, "Adversarial attacks on YOLACT instance segmentation", Computers & Security, vol. 116, 2022

[55] Z. Shang, X. Wang, Y. Jiang, Z. Li and J. Ning, "Identifying rumen protozoa in microscopic images of ruminant with improved YOLACT instance segmentation", Biosystems Engineering, vol. 215, 2022, pp.156-169

[56] Z.Zhang, S. Huang, X. Liu, B. Zhang and D. Dong, "Adversarial attacks on YOLACT instance segmentation", Computers & Security, vol. 116, 2022

[57] P.S. Janardhanan, "Project repositories for machine learning with TensorFlow", Procedia Computer Science, vol. 171, 2020, pp.188-196

[58] www.v7labs.com. (n.d.)., “Object Detection: Models, Architectures & Tutorial”, 2022. [online] Available at: https://www.v7labs.com/blog/object-detection-guide [Accessed 2 Feb 2023].

[59] www.wikidocs.net. (n.d.). Part F. Convolutional Neural Networks - EN. [online] Available at: https://wikidocs.net/165403 [Accessed 2 Feb 2023].

[60] Analytics Vidhya. “Supervised Deep Learning Algorithms: Types and Applications”, 2021. [online] Available at: https://www.analyticsvidhya.com/blog/2021/05/introduction-to-supervised-deep-learning-algorithms/ [Accessed 2 Feb 2023].

[61] The data science blog, “An Intuitive Explanation of Convolutional Neural Networks”, 2017. [online] Available at: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ [Accessed 2 Feb 2023].

[62] P. Baheti, “12 Types of Neural Networks Activation Functions: How to Choose?”, 2022. [online] www.v7labs.com. Available at: https://www.v7labs.com/blog/neural-networks-activation-functions.

[63] T. Wood, “Convolutional Neural Networks”, 2019. [online] DeepAI. Available at: https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network.

[64] L. Yann, L. Bottou, Y. Bengio and P. Haffner, “Gradient-Based Learning Applied to Document Recognition”, Proceedings of the IEEE. vol.86, 1998, pp. 2278 - 2324. 10.1109/5.726791.

[65] Wikipedia Contributors, “Kernel (image processing)”, 2019. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Kernel_(image_processing).

[66] A.F. Shokraei Fard, D. Reutens and V. Vegh, “From CNNs to GANs for cross-modality medical image estimation”, Computers in Biology and Medicine, 2022, pp.146. 105556. 10.1016/j.compbiomed.2022.105556.

[67] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, D.G Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur and X. Zheng, Xiaoqiang, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”, 2016.

[68] H. Tang, R. Shi, T. He, Y. Zhu, T. Wang, M. Lee and X. Jin, “TensorFlow solver for quantum PageRank in large-scale networks”, Science Bulletin. 2020, pp66, 10.1016/j.scib.2020.09.009.

[69] D. Bolya, ”yolact, You Only Look At CoefficienTs”, 2020. [online] Available at: https://github.com/dbolya/yolact [Accessed 2 Feb 2023].

[70] A. Chirita, B. Neluș-Constantin and C. Dragos. Intel x86 and ARM processors: A survey on architectural differences, 2002.

[71] D. Smith, “ARM and Intel Battle over the Mobile Chip's Future”, Computer. vol.41, 2008, pp.15 - 18. 10.1109/MC.2008.142.

[72] R. Shu, C. Liu, H. Liang and Y. Liang, "Potential mediators of the relationship between vision impairment and self-rated health in older adults: A comparison between long-term care insurance claimants in residential care institutions versus those living in the community", Geriatric Nursing, vol.44, 2022, pp.259-265

[73] Open Source Computer Vision, “OpenCV: Introduction.” [online] Available at: https://docs.opencv.org/4.x/d1/dfb/intro.html [Accessed 2 Feb 2023].

CHAPTER 9 Appendices

9.1 Appendix A1

Source code: imagecapture

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <iomanip>
#include <string>
#include <ctime>
#include <fstream> // to write logs to file
#include <sstream>
#include <chrono>
#include <thread>
using namespace cv;
using namespace std;

int main()
{

        auto t_start = std::time(nullptr);
        auto tm_start = *std::localtime(&t_start);
         ostringstream oss;
          string current_file="";
           while(1)
           {
                    Mat save_img;
                     VideoCapture cap;
```

```
                if(cap.open("rtsp://192.168.1.254:554/videoMain"))
              //   if(cap.open(0))
                {
                        cap >> save_img;
                        cap.release();
                }
                else
                {
                    cout<<"Failed to open camera"<<endl;
                }
                    if (save_img.empty())
                    {
                        cout<<"Failed to save frame"<<endl;
                    }
                    else
                    {
                        auto t_end = std::time(nullptr);
                        auto tm_end = *std::localtime(&t_end);
                        oss.str("");
                        oss.clear();
                        oss<<put_time(&tm_end, "%d-%m-%Y_%H%M%S");
                        current_file="images/processing_"+oss.str()+".jpg";
                        imwrite(current_file.c_str(), save_img);
                        // a jpg file is being saved to file (the line above)
                    }
                    std::this_thread::sleep_for(std::chrono::milliseconds(5000));
                    save_img.release();
            }
    return 0;
}
```

9.2 Appendix A2

Source code: yolactWIFI

```
#include <ncnn/net.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <stdio.h>
#include <vector>
#include <fstream>
#include <ctime>
#include <sstream>
#include <chrono>
#include <iomanip>
using namespace std;
using namespace cv;

struct Object
{
    cv::Rect_<float> rect;
    int label;
    float prob;
    std::vector<float> maskdata;
    cv::Mat mask;
};

static inline float intersection_area(const Object& a, const Object& b)
{
    cv::Rect_<float> inter = a.rect & b.rect;
    return inter.area();
}
```

```
static void qsort_descent_inplace(std::vector<Object>& objects, int left, int right)
{
    int i = left;
    int j = right;
    float p = objects[(left + right) / 2].prob;

    while (i <= j)
    {
        while (objects[i].prob > p)
            i++;

        while (objects[j].prob < p)
            j--;

        if (i <= j)
        {
            // swap
            std::swap(objects[i], objects[j]);

            i++;
            j--;
        }
    }

    #pragma omp parallel sections
    {
        #pragma omp section
        {
            if (left < j) qsort_descent_inplace(objects, left, j);
        }
        #pragma omp section
```

```
        {
            if (i < right) qsort_descent_inplace(objects, i, right);
        }
    }
}

static void qsort_descent_inplace(std::vector<Object>& objects)
{
    if (objects.empty())
        return;

    qsort_descent_inplace(objects, 0, objects.size() - 1);
}

static void nms_sorted_bboxes(const std::vector<Object>& objects, std::vector<int>& picked, float nms_threshold)
{
    picked.clear();

    const int n = objects.size();

    std::vector<float> areas(n);
    for (int i = 0; i < n; i++)
    {
        areas[i] = objects[i].rect.area();
    }

    for (int i = 0; i < n; i++)
    {
        const Object& a = objects[i];

        int keep = 1;
```

```
        for (int j = 0; j < (int)picked.size(); j++)
        {
            const Object& b = objects[picked[j]];

            // intersection over union
            float inter_area = intersection_area(a, b);
            float union_area = areas[i] + areas[picked[j]] - inter_area;
            //              float IoU = inter_area / union_area
            if (inter_area / union_area > nms_threshold)
                keep = 0;
        }

        if (keep)
            picked.push_back(i);
    }
}

static int detect_yolact(const cv::Mat& bgr, std::vector<Object>& objects,string current_file)
{
    ncnn::Net yolact;
    std::chrono::steady_clock::time_point Tbegin, Tend;

    yolact.opt.use_vulkan_compute = true;

            current_file=current_file+".txt";
    ofstream outFile(current_file.c_str());
    // original model converted from https://github.com/dbolya/yolact
    // yolact_resnet50_54_800000.pth
    // the ncnn model https://github.com/nihui/ncnn-assets/tree/master/models
    yolact.load_param("yolact.param");
    yolact.load_model("yolact.bin");
```

```
    const int target_size = 550;

    int img_w = bgr.cols;
    int img_h = bgr.rows;

    ncnn::Mat in = ncnn::Mat::from_pixels_resize(bgr.data, ncnn::Mat::PIXEL_BGR2RGB,
img_w, img_h, target_size, target_size);

    const float mean_vals[3] = {123.68f, 116.78f, 103.94f};
    const float norm_vals[3] = {1.0 / 58.40f, 1.0 / 57.12f, 1.0 / 57.38f};
    in.substract_mean_normalize(mean_vals, norm_vals);

    ncnn::Extractor ex = yolact.create_extractor();
    //     ex.set_num_threads(4);

    ex.input("input.1", in);

    Tbegin = std::chrono::steady_clock::now();

    ncnn::Mat maskmaps;
    ncnn::Mat location;
    ncnn::Mat mask;
    ncnn::Mat confidence;

    ex.extract("619", maskmaps); // 138x138 x 32

    ex.extract("816", location);  // 4 x 19248
    ex.extract("818", mask);       // maskdim 32 x 19248
    ex.extract("820", confidence); // 81 x 19248

    int num_class = confidence.w;
```

```
int num_priors = confidence.h;

// make priorbox
ncnn::Mat priorbox(4, num_priors);
{
    const int conv_ws[5] = {69, 35, 18, 9, 5};
    const int conv_hs[5] = {69, 35, 18, 9, 5};

    const float aspect_ratios[3] = {1.f, 0.5f, 2.f};
    const float scales[5] = {24.f, 48.f, 96.f, 192.f, 384.f};

    float* pb = priorbox;

    for (int p = 0; p < 5; p++)
    {
        int conv_w = conv_ws[p];
        int conv_h = conv_hs[p];

        float scale = scales[p];

        for (int i = 0; i < conv_h; i++)
        {
            for (int j = 0; j < conv_w; j++)
            {
                // +0.5 because priors are in center-size notation
                float cx = (j + 0.5f) / conv_w;
                float cy = (i + 0.5f) / conv_h;

                for (int k = 0; k < 3; k++)
                {
                    float ar = aspect_ratios[k];
```

```
                ar = sqrt(ar);

                float w = scale * ar / 550;
                float h = scale / ar / 550;

                // This is for backward compatability with a bug where I made everything
square by accident
                // cfg.backbone.use_square_anchors:
                h = w;

                pb[0] = cx;
                pb[1] = cy;
                pb[2] = w;
                pb[3] = h;

                pb += 4;
            }
        }
    }
}

const float confidence_thresh = 0.05f;
const float nms_threshold = 0.5f;
const int keep_top_k = 200;

std::vector<std::vector<Object> > class_candidates;
class_candidates.resize(num_class);

for (int i = 0; i < num_priors; i++)
{
    const float* conf = confidence.row(i);
```

```
const float* loc = location.row(i);
const float* pb = priorbox.row(i);
const float* maskdata = mask.row(i);

// find class id with highest score
// start from 1 to skip background
int label = 0;
float score = 0.f;
for (int j = 1; j < num_class; j++)
{
    float class_score = conf[j];
    if (class_score > score)
    {
        label = j;
        score = class_score;
    }
}

// ignore background or low score
if (label == 0 || score <= confidence_thresh)
    continue;

// CENTER_SIZE
float var[4] = {0.1f, 0.1f, 0.2f, 0.2f};

float pb_cx = pb[0];
float pb_cy = pb[1];
float pb_w = pb[2];
float pb_h = pb[3];

float bbox_cx = var[0] * loc[0] * pb_w + pb_cx;
float bbox_cy = var[1] * loc[1] * pb_h + pb_cy;
```

```
            float bbox_w = (float)(exp(var[2] * loc[2]) * pb_w);
            float bbox_h = (float)(exp(var[3] * loc[3]) * pb_h);

            float obj_x1 = bbox_cx - bbox_w * 0.5f;
            float obj_y1 = bbox_cy - bbox_h * 0.5f;
            float obj_x2 = bbox_cx + bbox_w * 0.5f;
            float obj_y2 = bbox_cy + bbox_h * 0.5f;

            // clip
            obj_x1 = std::max(std::min(obj_x1 * bgr.cols, (float)(bgr.cols - 1)), 0.f);
            obj_y1 = std::max(std::min(obj_y1 * bgr.rows, (float)(bgr.rows - 1)), 0.f);
            obj_x2 = std::max(std::min(obj_x2 * bgr.cols, (float)(bgr.cols - 1)), 0.f);
            obj_y2 = std::max(std::min(obj_y2 * bgr.rows, (float)(bgr.rows - 1)), 0.f);

            // append object
            Object obj;
            obj.rect = cv::Rect_<float>(obj_x1, obj_y1, obj_x2 - obj_x1 + 1, obj_y2 - obj_y1 + 1);
            obj.label = label;
            obj.prob = score;
            obj.maskdata = std::vector<float>(maskdata, maskdata + mask.w);

            class_candidates[label].push_back(obj);
        }

        objects.clear();
        for (int i = 0; i < (int)class_candidates.size(); i++)
        {
            std::vector<Object>& candidates = class_candidates[i];

            qsort_descent_inplace(candidates);

            std::vector<int> picked;
```

```
    nms_sorted_bboxes(candidates, picked, nms_threshold);

    for (int j = 0; j < (int)picked.size(); j++)
    {
        int z = picked[j];
        objects.push_back(candidates[z]);
    }
}

qsort_descent_inplace(objects);

// keep_top_k
if (keep_top_k < (int)objects.size())
{
    objects.resize(keep_top_k);
}

// generate mask
for (int i = 0; i < objects.size(); i++)
{
    Object& obj = objects[i];

    cv::Mat mask(maskmaps.h, maskmaps.w, CV_32FC1);
    {
        mask = cv::Scalar(0.f);

        for (int p = 0; p < maskmaps.c; p++)
        {
            const float* maskmap = maskmaps.channel(p);
            float coeff = obj.maskdata[p];
            float* mp = (float*)mask.data;
```

```
            // mask += m * coeff
            for (int j = 0; j < maskmaps.w * maskmaps.h; j++)
            {
                mp[j] += maskmap[j] * coeff;
            }
        }
    }

    cv::Mat mask2;
    cv::resize(mask, mask2, cv::Size(img_w, img_h));

    // crop obj box and binarize
    obj.mask = cv::Mat(img_h, img_w, CV_8UC1);
    {
        obj.mask = cv::Scalar(0);

        for (int y = 0; y < img_h; y++)
        {
            if (y < obj.rect.y || y > obj.rect.y + obj.rect.height)
                continue;

            const float* mp2 = mask2.ptr<const float>(y);
            uchar* bmp = obj.mask.ptr<uchar>(y);

            for (int x = 0; x < img_w; x++)
            {
                if (x < obj.rect.x || x > obj.rect.x + obj.rect.width)
                    continue;

                bmp[x] = mp2[x] > 0.5f ? 255 : 0;
            }
        }
```

```
        }
    }
    Tend = std::chrono::steady_clock::now();
    float f = std::chrono::duration_cast <std::chrono::milliseconds> (Tend - Tbegin).count();

    std::cout << "time : " << f/1000.0 << " Sec" << std::endl;
          outFile<< "time : " << f/1000.0 << " Sec" << std::endl;
          outFile.close();

    return 0;
}

static void draw_objects(const cv::Mat& bgr, const std::vector<Object>& objects,string current_file)
{

    static const char* class_names[] = {"background",
                                        "person", "bicycle", "car", "motorcycle", "airplane", "bus",
                                        "train", "truck", "boat", "traffic light", "fire hydrant",
                                        "stop sign", "parking meter", "bench", "bird", "cat", "dog",
                                        "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe",
                                        "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee",
                                        "skis", "snowboard", "sports ball", "kite", "baseball bat",
                                        "baseball glove", "skateboard", "surfboard", "tennis racket",
                                        "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl",
                                        "banana", "apple", "sandwich", "orange", "broccoli", "carrot",
                                        "hot dog", "pizza", "donut", "cake", "chair", "couch",
                                        "potted plant", "bed", "dining table", "toilet", "tv", "laptop",
                                        "mouse", "remote", "keyboard", "cell phone", "microwave",
"oven",
                                        "toaster", "sink", "refrigerator", "book", "clock", "vase",
                                        "scissors", "teddy bear", "hair drier", "toothbrush"
```

```
                    };

    static const unsigned char colors[19][3] = {
        {244, 67, 54},
        {233, 30, 99},
        {156, 39, 176},
        {103, 58, 183},
        {63, 81, 181},
        {33, 150, 243},
        {3, 169, 244},
        {0, 188, 212},
        {0, 150, 136},
        {76, 175, 80},
        {139, 195, 74},
        {205, 220, 57},
        {255, 235, 59},
        {255, 193, 7},
        {255, 152, 0},
        {255, 87, 34},
        {121, 85, 72},
        {158, 158, 158},
        {96, 125, 139}
    };

    cv::Mat image = bgr.clone();

    int color_index = 0;

    for (size_t i = 0; i < objects.size(); i++)
    {
        const Object& obj = objects[i];
```

```
if (obj.prob < 0.40) // 40% minimum accaptable confidence level
   continue;

fprintf(stderr, "%d = %.5f at %.2f %.2f %.2f x %.2f\n", obj.label, obj.prob,
      obj.rect.x, obj.rect.y, obj.rect.width, obj.rect.height);

const unsigned char* color = colors[color_index++];

cv::rectangle(image, obj.rect, cv::Scalar(color[0], color[1], color[2]));

char text[256];
sprintf(text, "%s %.1f%%", class_names[obj.label], obj.prob * 100);

int baseLine = 0;
cv::Size label_size = cv::getTextSize(text, cv::FONT_HERSHEY_SIMPLEX, 0.5, 1,
&baseLine);

int x = obj.rect.x;
int y = obj.rect.y - label_size.height - baseLine;
if (y < 0)
   y = 0;
if (x + label_size.width > image.cols)
   x = image.cols - label_size.width;

cv::rectangle(image, cv::Rect(cv::Point(x, y), cv::Size(label_size.width,
label_size.height + baseLine)),
            cv::Scalar(255, 255, 255), -1);

cv::putText(image, text, cv::Point(x, y + label_size.height),
         cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(0, 0, 0));

            char textspeak[256];
```

```
                sprintf(textspeak, "echo \"There is a %s ahead\" | festival --tts",
class_names[obj.label]);
            system(textspeak);
      // draw mask
     /* for (int y = 0; y < image.rows; y++)
      {
         const uchar* mp = obj.mask.ptr(y);
         uchar* p = image.ptr(y);
         for (int x = 0; x < image.cols; x++)
         {
            if (mp[x] == 255)
            {
               p[0] = cv::saturate_cast<uchar>(p[0] * 0.5 + color[0] * 0.5);
               p[1] = cv::saturate_cast<uchar>(p[1] * 0.5 + color[1] * 0.5);
               p[2] = cv::saturate_cast<uchar>(p[2] * 0.5 + color[2] * 0.5);
            }
            p += 3;
         }
      }*/
   }
         // SINGLE FILE
         current_file=current_file+".png";
   imwrite(current_file.c_str(),image);

}

int main()
{
   std::vector<Object> objects;
         ostringstream oss;
         string current_file="";
         string current_file2="";
```

```
do
{
        Mat save_img;
VideoCapture cap;

        if(cap.open(0))
        {
                cap >> save_img;
                cap.release();
        }
        else
        {
            cout<<"Failed to open camere"<<endl;
                return -1;
        }


            if (save_img.empty())
            {
                cout<<"Failed to save frame"<<endl;
                return -1;
            }

            // MULTIFILE PROCESS
            auto t_end = std::time(nullptr);
            auto tm_end = *std::localtime(&t_end);

            oss.str("");
            oss.clear();

            oss<<put_time(&tm_end, "%d-%m-%Y_%H%M%S");
            current_file="logs/processing_"+oss.str();
```

```
                    // SINGLE FILE
                    current_file2="logs/processing_singlefile";

                    detect_yolact(save_img, objects,current_file /*current_file */);
                    draw_objects(save_img, objects,current_file2 /*current_file */);
                    save_img.release();

        }while(1);



    return 0;
}
```

9.3 Appendix A3

Source code: yolactUSB

Similar to yolactWIFI, except the main function is implemented to access the camera via a USB cable as shown below.

```
int main()
{

   VideoCapture cap;
   Mat save_img;
   std::vector<Object> objects;

  if(cap.open(0))
  {
    cap >> save_img;

    if (save_img.empty())
    {
      cout<<"Faile to save frame"<<endl;
      return -1;
    }
   detect_yolact(save_img, objects);
   draw_objects(save_img, objects);
  }
   return 0;
}
```

9.4 Appendix A4

Source code: *yolactstatic*

Similar to yolactWIFI, except the main function is implemented to process an image file that is passed to an application as a parameter, as shown below.

```
int main(int argc, char** argv)
{
   std::vector<Object> objects;
         ostringstream oss;
         string current_file="";
         string current_file2="";
         int objectCount=0;
                        Mat save_img=imread(argv[1], IMREAD_COLOR);// LOAD FILE
                        auto t_end = std::time(nullptr);
                        auto tm_end = *std::localtime(&t_end);

                        oss.str("");
                        oss.clear();

                        oss<<put_time(&tm_end, "%d-%m-%Y_%H%M%S");
                        current_file="static/processing_"+oss.str();

                        // SINGLE FILE
                        //current_file2="static/processing_singlefile";

                        detect_yolact(save_img, objects,current_file );
                        draw_objects(save_img, objects,current_file,objectCount);
                        save_img.release();
   return 0;
}
```