THE HASHEMITE UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

SENIOR DESIGN PROJECT-(II)

Insert SDP Title Here

STUDENTS

Aya Omar Al Shawabkah	1938144
Taher Mahmoud Horani	1933960
Mustafa Abed Al Muttaleb Mathhar	1835052
Mohammad Akram Saleh	1935480

Senior Design Project Advisor
Dr.Islam Al Malkawi

DEPARTMENT OF *COMPUTER* ENGINEERING

Academic Year
2023-2024

ACKNOWLEDGMENT

We would like to express our profound gratitude to the Computer Engineering Department at Hashemite University for their unwavering support and guidance throughout the entirety of our senior project. Their commitment to fostering an enriching learning environment and providing invaluable resources has played an integral role in the successful completion of our project.

A special acknowledgment is extended to Dr. Islam Al-Malkawi, our esteemed project advisor. His expertise, encouragement, and insightful feedback have been pivotal in shaping the trajectory of our project. We deeply appreciate his patience, guidance, and mentorship, which have proven invaluable on our journey.

Our thanks also extend to Hashemite University for furnishing the necessary infrastructure and facilities that facilitated our research effectively. Their steadfast dedication to academic excellence significantly contributed to the quality and success of our senior project.

Furthermore, we express our appreciation to the Sho'ola Club for their unwavering support and collaboration. Their active engagement in promoting inclusivity and accessibility for visually impaired individuals has played a vital role in the development and testing of our project. Their commitment to making a positive impact in the community serves as a true inspiration.

In conclusion, heartfelt thanks are extended to all the professors, staff, and fellow students who have enriched our academic journey at Hashemite University. Your encouragement, camaraderie, and shared knowledge have been immensely valuable, and we are grateful for the experiences and friendships gained during our time here.

Once again, our sincere thanks go to the Computer Engineering Department, Dr. Islam Al Malkawi, Hashemite University, and the Sho'ola Club for their unwavering support, guidance, and collaboration. It is through their collective efforts that our senior project has come to fruition.

ABSTRACT

This documentation addresses the prevalent challenge of blindness, the primary form of disability in Jordan. Our strategy employs cutting-edge technology, advanced algorithms, and robust protocols to enhance accessibility for individuals with visual impairments. The core of our approach involves the integration of hardware embedded systems, particularly the Raspberry Pi Zero W, in conjunction with cloud services. This collaboration enables the efficient handling of computationally intensive tasks by offloading heavy processing to the cloud.

Our system utilizes text detection and object detection techniques, leveraging the capabilities of the Raspberry Pi Zero W. Through the integration with a cloud streaming server, we achieve near-real-time text detection from video input, rendering the traditional need for braille obsolete. The identified text is then transformed into speech using sophisticated text-to-speech technology.

A pivotal aspect of our system design is its energy efficiency, meticulously crafted to ensure optimal resource utilization, especially given the constraints of the Raspberry Pi Zero W. This documentation serves as a comprehensive repository of insights into the development and implementation of our solution, focusing on text detection and object detection using Raspberry Pi Zero W and cloud services. The overarching objective is to strive to empower and simplify the lives of the blind community through the innovative application of text detection and object detection technologies.

Keywords: Embedded systems, Cloud computing, Text-to-speech, Text detection, Object detection, Raspberry Pi Zero W.

TABLE OF CONTENTS

LIST OF FIGURES

Number *Page*

LIST OF TABLES

EXECUTIVE SUMMARY

The objective of our project was to develop a smart glasses system to assist blind individuals through text detection, object detection, and text-to-speech conversion. This innovative solution aimed to enhance the independence and daily lives of visually impaired individuals.

The project was structured into two distinct 14-week phases. In the initial phase, extensive literature research was conducted to understand the cloud environment, its characteristics, and features. The team gained expertise in dealing with cloud technologies and explored Optical Character Recognition (OCR [1]) algorithms for text detection and the YOLO [2] algorithm for object detection. The challenge during this phase involved selecting and integrating appropriate components onto the smart glasses.

In the second phase, the focus shifted to achieving near-real-time capabilities at a cost-effective price. To address this, heavy processing and computing were offloaded from the Raspberry Pi to a cloud server. The server, written in the high-performance language Go [3], leveraged Google Cloud Vision APIs [4] for cutting-edge computer vision capabilities. This strategic decision allowed us to provide an efficient and scalable solution for the smart glasses.

The outcome of the project is a successful implementation of smart glasses designed to assist blind individuals in their daily lives. The system incorporates text detection, object detection, and text-to-speech conversion, offering a range of functionalities to support and enhance the daily activities of visually impaired users.

Keywords: Smart Glasses, Optical Character Recognition (OCR), YOLO Algorithm, Go Programming Language, Google Cloud Vision APIs.

1 CHAPTER 1: INTRODUCTION

1.1 Problem Statement and Purpose

In Jordan, approximately 11.5% of the population is affected by disabilities, with visual impairment being the most prevalent, impacting around 6% of individuals [5]. This substantial portion of the population encounters challenges in independently accessing and comprehending textual and visual information. Although existing solutions like Braille and audio-based devices offer some support, their availability is often limited, and they do not fully address the issue.

Compounding the challenge is the fact that a significant amount of text and reading materials are not translated into Braille. Additionally, understanding the visual context of the surroundings remains a significant hurdle for visually impaired individuals.

Consequently, there is an immediate need for an innovative and efficient solution. Smart glasses, equipped with both text and object detection technology, can empower visually impaired individuals in Jordan to read and understand printed text, recognize objects, and interpret their surroundings in real-time. This integration aims to effectively bridge the accessibility gap caused by limited Braille resources and the absence of a comprehensive solution addressing both text and object recognition.

This solution must prioritize concerns related to accuracy, speed, usability, and affordability. The overarching goal is to enhance the quality of life and opportunities for visually impaired individuals in Jordan by providing a versatile tool that not only interprets text but also identifies and communicates information about the surrounding environment through object detection.

1.2 Project and Design Objectives

The primary objective of our smart glasses project is to empower visually impaired individuals by providing a comprehensive solution that combines text recognition and object detection, enhancing their ability to perceive and navigate their surroundings. The key project and design objectives are outlined as follows:

1. Text Recognition:

Enable the smart glasses to recognize and convert printed or written text into audible information, implement advanced text-to-speech technology to provide clear and natural-sounding audio output.

2. Object Detection:

Integrate object detection capabilities to allow the smart glasses to identify and relay information about objects in the user's environment, ensure real-time processing for immediate and accurate feedback.

3. Auditory Feedback:

Utilize sound as the primary mode of conveying information to the user, ensuring an intuitive and non-intrusive user experience, design a system that delivers distinct and easily interpretable auditory cues for both text and object recognition.

4. Affordability:

Strive for a cost-effective design to make the smart glasses accessible to a wide range of users, considering the economic landscape of potential users, explore affordable yet high-performance components to balance functionality and cost-effectiveness.

5. Usability and User Experience:

Prioritize user-friendly interfaces and controls to enhance the usability of the smart glasses, solicit user feedback in the design process to ensure the technology aligns with the needs and preferences of visually impaired individuals.

6. High Performance:

Optimize the performance of the smart glasses, focusing on quick response times for both text and object recognition, explore cutting-edge technologies and algorithms to achieve a high level of accuracy in recognition tasks.

7. Scalability:

Design the smart glasses with the potential for future enhancements and updates, ensuring scalability and adaptability to emerging technologies.

The amalgamation of these objectives aims to create a smart glasses solution that not only aids in recognizing text and objects but also addresses the challenges of affordability, accessibility, and overall user satisfaction. Through this project, we aspire to contribute to the improved quality of life for visually impaired individuals by leveraging technology for their benefit.

1.3 Intended Outcomes and Deliverables

The project aims to develop a smart glasses system with interactive and intelligent features to assist visually impaired individuals. The intended outcomes and deliverables for the smart glasses project include:

- Wide Range of Services: Offer a comprehensive set of services, such as real-time text detection, object recognition, and text-to-speech conversion, to enhance the daily lives of visually impaired individuals.

- Accessibility and Affordability: Design the smart glasses to be accessible to a broad user base by prioritizing affordability without compromising on performance.

- Personalized Assistance: Provide personalized assistance through the smart glasses, addressing specific user needs related to text recognition and object detection.

These intended outcomes and deliverables aim to create a smart glasses solution that not only assists in recognizing texts and objects but also promotes accessibility, affordability, and overall well-being for visually impaired users.

1.4 Summary of the Used Design Process

In our comprehensive design process, key decisions were strategically made to enhance the capabilities of our system. Upon thorough evaluation, we elected to employ a dedicated WebSockets server over a WebRTC-based media streaming server, prioritizing simplicity and speed for real-time communication.

In parallel, we seamlessly integrated powerful external services, such as Google Cloud Vision APIs, for advanced functionalities like object detection and text detection within images. The choice of these services was rooted in their cost-effectiveness, robust features, user-friendly interfaces, and adaptability for future enhancements.

To ensure efficient communication between devices, we meticulously crafted a high-performance client tailored specifically for Raspberry Pi. Employing the Go language for both client and server components, with its concurrent processing model using goroutines and channels, proved instrumental. This architecture empowered our system to handle real-time communication, concurrent processing, and advanced services seamlessly.

Concurrency was adeptly implemented through dedicated goroutines. One goroutine facilitated client communication, while another interacted with Google Cloud API services. These goroutines communicated through channels, enabling the WebSockets server to receive video frames in real-time. Simultaneously, the Google Cloud Vision API processed these frames, incorporating object detection and text detection features.

A notable achievement lies in the successful integration of text-to-speech functionality using Google Cloud's Text-to-Speech API. The API generates sound bytes that the WebSockets server transmits to the client, which then plays the bytes, providing a comprehensive audio output for the text being read.

On the hardware front, a deliberate choice was made to use a microcomputer, specifically the Raspberry Pi Zero W, over a microcontroller. Despite the modest specifications, the Raspberry Pi Zero W's integration of the Debian-based Raspian OS, video camera port, and Wi-Fi connectivity proved instrumental. This decision aimed to streamline development and reduce time investment, contributing to the overall efficiency of our design.

1.5 Summary of Report Structure

This report comprises eight chapters, each focusing on different aspects of the project. The chapter breakdown and content overview are as follows:

The first chapter provides an overview of the report and includes sections such as the Problem Statement and Purpose, Project and Design Objectives, Intended Outcomes and Deliverables, Summary of the Used Design Process, and Summary of Report Structure.

The second chapter focuses on the project's initial design, delving into the proposed preliminary design, the selected design, and the final deliverables. Additionally, it provides insights into the preliminary cost considerations associated with shaping the innovative features and functionalities of the smart glasses.

In the third chapter, the theoretical foundation of the project is explored, concentrating on the theoretical underpinnings of GP-II. The chapter involves an extensive review of pertinent literature, specifically focusing on smart glasses projects akin to ours. The objective is to glean insights, methodologies, and theoretical frameworks from existing research to inform and enrich the theoretical basis of our smart glasses project.

In the fourth chapter, the detailed design of the glasses is elucidated, encompassing the components, project methodology, cloud integration, text detection, object detection, and text-to-speech conversion. This comprehensive section provides an in-depth understanding of the glasses' architecture, functionality, and the intricacies of working with the system. It serves as a practical guide, offering step-by-step explanations for effective implementation and operation.

The five chapter involves the practical implementation of the smart glasses project. This encompasses the execution of the designed system and an emphasis on refining and enhancing its performance. The chapter details the hands-on realization process, optimizations undertaken to improve efficiency, and the overall achievement of project goals through practical implementation and fine-tuning for optimal performance.

The six chapter addresses Economic, Ethical, and Contemporary issues associated with the project. It covers topics such as Project Cost, Relevant Codes of Ethics, Commercializing the Project and Relevance to JORDAN, and Relevant Environmental Analysis and Discussion.

In Chapter seven the emphasis is on Project Management, offering insights into the strategic planning, execution, and control of the smart glasses project. This section covers project timelines, resource allocation and Cost, risk management, Lessons learned, and other crucial aspects of effectively overseeing and coordinating the various elements involved in the project's development and implementation.

In the final chapter, the document synthesizes key findings, achievements, and insights gained throughout the smart glasses project. The "Conclusion" section offers a comprehensive summary of the project's outcomes, highlighting its contributions and addressing the initial problem statement and objectives. Following the conclusion, the "Future Work" section outlines potential avenues for further development, improvement, or expansion of the smart glasses project, providing a roadmap for future research and advancements in the field. Together, these segments encapsulate the project's journey, outcomes, and possibilities for ongoing exploration and innovation.

By organizing the report into these chapters, we aim to provide a comprehensive understanding of the project, covering its introduction, theoretical foundations, detailed design, economic and ethical considerations, evaluation results, and a conclusion that reflects on the overall project outcome while outlining future work.

2 CHAPTER 2: SUMMARY OF ACHEIVMENTS IN SENIOR DESIGN PROJECT-I

2.1 Proposed Preliminary Design

In the initial phase of our project, we embarked on a journey of ideation to address the identified needs of individuals through innovative technological solutions. The fundamental concept that emerged focused on the development of a wearable device equipped with a camera mounted on glasses, leveraging a microprocessor such as the Raspberry Pi Zero W.

The core idea revolves around catering to the requirements of individuals who could benefit from a hands-free, intelligent assistive device. By integrating a camera into everyday eyewear, we aimed to create a versatile tool capable of performing various tasks. The envisioned device was designed to not only capture visual information but also process it in real-time using advanced technologies.

To enhance the functionality of the device, our proposed design included the implementation of a streaming server. This server would facilitate seamless communication and data transfer, enabling users to access information remotely. Additionally, we planned to integrate machine learning into the device by training a model. This model would leverage Optical Character Recognition (OCR) and You Only Look Once (YOLO) object detection algorithms, enhancing the device's ability to interpret and respond to the user's surroundings.

Furthermore, we explored the integration of Google Text-To-Speech technology, aiming to provide auditory feedback to users based on the visual information processed by the device. This feature was intended to improve accessibility and user interaction.

Considering the importance of real-time communication, the incorporation of WebRTC [6] (Web Real-Time Communication) was a key element in our preliminary design. This technology would enable live video and audio communication, fostering a collaborative and interactive user experience.

Briefly, our proposed preliminary design involved the development of a smart, wearable device capable of capturing, processing, and communicating information in real-time. The integration of cutting-edge

technologies, such as OCR, YOLO, Google Text-To-Speech [7], and WebRTC, aimed to create a comprehensive solution catering to the diverse needs of users. This conceptualization laid the foundation for subsequent stages of our project.

2.2 Selected Preliminary Design and Justification

After careful consideration and evaluation of various proposed preliminary designs, we have arrived at a definitive choice for our system architecture. The selected design not only aligns with the core objectives of our project but also demonstrates optimal performance and practicality. The key components of the chosen preliminary design are outlined below, accompanied by the justification for their selection:

1) WebSockets[8] Server for Real-Time Communication:

Justification: The decision to employ a dedicated WebSockets server was based on its demonstrated simplicity and speed for real-time communication. This choice ensures efficient and reliable communication between our client and server components, a critical requirement for our system.

2) Google Cloud Vision APIs for Image Processing:

Justification: We opted for Google Cloud Vision APIs due to their cost-effectiveness, powerful features, simplicity of use, and adaptability for future modifications. These APIs provide advanced capabilities such as object detection and text detection within images, enhancing the overall functionality of our system.

3) Go Language for Server and Client Implementation:

Justification: Choosing Go as the programming language for both the server and client components was motivated by its ability to achieve C-like performance while remaining easy to use and develop with. Go's concurrency model, facilitated by goroutines and channels, enables effective handling of real-time communication and processing without compromising efficiency.

4) Concurrency Model with Goroutines and Channels:

Justification: The incorporation of goroutines and channels in our concurrency model allows for seamless communication between different components of our system. This design choice ensures that tasks such as client communication, interaction with Google Cloud API services, and real-time processing occur concurrently without introducing bottlenecks.

5) Integration of Object Detection and Text Detection Services:

Justification: By integrating Google Cloud Vision APIs for object detection and text detection, our system gains the capability to analyze and interpret visual information within images. This enhances the overall intelligence and utility of our application, allowing it to process and respond to complex visual data.

6) Text-to-Speech Functionality using Google Cloud's Text-to-Speech API:

Justification: The integration of text-to-speech functionality using Google Cloud's Text-to-Speech API adds a valuable auditory dimension to our system. This feature enhances user experience by converting text into sound bytes, providing a comprehensive audio output for the information being processed.

7) Raspberry Pi Zero W for Hardware Implementation:

Justification: Choosing the Raspberry Pi Zero W; As the microcomputer for hardware implementation was driven by the need for simplicity, development efficiency, and time reduction. Despite its limitations, such as modest specifications, the Raspberry Pi Zero W's integration of Raspian OS, video camera port, and Wi-Fi connectivity proved instrumental in achieving our project goals.

Briefly, the selected preliminary design represents a strategic combination of components and technologies that collectively fulfill our project's requirements. Each element was chosen for its specific strengths and contributions to the overall functionality and efficiency of the system.

2.3 Final Deliverables

The final deliverables encompass the tangible outcomes and achievements resulting from our comprehensive design and implementation process. Each element contributes to the overall success of the project, integrating cutting-edge technologies and efficient hardware configurations. Below, the table

provides a concise breakdown of these key deliverables, outlining the components, functionalities, and the associated costs involved in bringing our envisioned system to fruition.

Deliverable	Description
WebSockets Server	Implementation of a dedicated server for efficient real-time communication
High-Performance Raspberry Pi Client	Client application designed for Raspberry Pi, facilitating video frame transmission and audio reception
Integration with Google Cloud Vision APIs	Inclusion of object detection and text detection services for image processing
Text-to-Speech Functionality	Implementation of audio output through Google Cloud's Text-to-Speech API
Hardware Setup	Assembly and configuration of hardware components including Raspberry Pi Zero W, camera, and batteries
Concurrent Processing	Implementation of concurrent processing using goroutines for efficient communication and data handling
System Integration	Successful integration of all components to achieve a cohesive and functional system

Table 2.1 design and implementation process

3 CHAPTER 3: UPDATED BACKGROUND THEORY

3.1 Relevant Literature Search for GP-II

Research on cameras for blind users has gained significant attention in recent years, focusing on developing innovative solutions to enhance their visual perception and access to information. Here are a few relevant literature findings in this field.

- **Deep Learning Assisted Smart Glasses as Educational Aid for Visually Challenged Students. [9]**

[Hawra AlSaid, Lina AlKhatib, Aqeela AlOraidh, Shoaa AlHaidar, Abul Bashar, Computer Engineering Department, Prince Mohammad Bin Fahd University]

This paper presents a novel smart glasses solution that enables blind or visually impaired individuals to "read" images. The system utilizes a Raspberry Pi 3 B+ microcontroller and a webcam integrated into the glasses.

Through the use of OpenCV, Tesseract, and EAST, the system can extract and recognize text from captured images using deep learning techniques.

The recognized text is then converted into audible signals using Google's Text to Speech (gTTS) API. These innovative smart glasses system offers a promising approach to assist individuals with visual challenges in accessing and understanding textual information.

- **Real Time Object Detection and Recognition for Blind People.[10]**

[Mohammad AL-Najjar, Ihab Suliman, Ghazi Al-Hanini, Dr. Ramzi Qawasma, Biomedical Engineering Department, Palestine Polytechnic University]

The Smart Glasses project aims to assist blind and visually impaired individuals in detecting and recognizing office tools using a small camera fixed on the glasses. The system provides job opportunities

by sending voice messages to an earphone, allowing users to independently and easily locate various items.

The project utilizes a Raspberry Pi device that analyzes captured images through neural network algorithms. By comparing image characteristics with a database, the system detects objects and generates audible notifications.

This intelligent system mimics human vision, enhancing accessibility and saving time and effort for the visually impaired.

- **Design of Smart Glasses that Enable Computer Vision for the Improvement of Autonomy of the Visually Impaired. [11]**

[C. C. Spandonidis*, D. Spyropoulos, N. Galiatsatos, F. Giannopoulos and D. Karageorgiou, Prisma Electronics S.A.

Solution	**Developer**	**Conceptual Design**	**Benefits**	**Drawbacks**	**Improvements**
eSight 3	CNETs	High resolution camera for image and video capture for low vision people.	Helps low vision people, avoiding surgery	Does not improve vision as it just an aid.	Waterproof versions are under development.
Oton Glass	Keisuke Shimakage, Japan	Support for dyslexic people. Converts images to words and then to audio.	Symbols to audio conversion, normal looking glasses, supports English and Japanese languages.	For only people with reading difficulty and no support for blind people.	Can be improved to support blind people also by including proximity sensors.
Aira	Suman Kanuganti	Aira uses smart glasses to scan the environment.	Aira agents help users to interpret	Waiting time connected to the Aira agents	To include language translation features.

			their surroundings with smart glasses.	in order to be able to sense.	
Eyesynth	Eyesynth, Spain	Eyesynth is consist 3D cameras, which turn the scene to sound signal (non-verbal) to provide information about of position, size, and shape. It is language independent.	It converts spatial and visual information into audio.	It is expensive (575.78$) and it only recognizes the objects and directions.	Can use verbal audio for better feeling and navigation services.
Google Glasses	Glasses Google Inc.	Google Glasses show information without using hands gestures. Users can communicate with the Internet via normal voice commands.	captures images and videos, get directions, audio calling and real-time translation using word lens app.	It is expensive (1,349.99$) and the glasses are not very helpful for blind people.	Reduce costs to make it more affordable for the consumers.
CCES,PMU Solution	CCES, PMU	Help people who have vision difficulties especially blind people.	Helps blind people to avoid obstacles, to aid in reading and learning and search for information about the words.	English language support only, no use while driving, separate processing unit.	The glasses can support other languages, it also can be smaller and easy to wear.

Table 3.1 Comparison between different smart glasses solutions

3.2 Literature on Potential Ethical and/or Environmental Issue

The literature surrounding the development and deployment of smart glasses acknowledges potential ethical and environmental issues that merit careful consideration. Ethically, concerns may arise regarding privacy and data security, especially when integrating with cloud services. Striking a balance between providing valuable features for users and safeguarding their personal information is a nuanced challenge.

Transparency in data collection practices and robust encryption measures become crucial components to address these concerns. Furthermore, from an environmental standpoint, the production and disposal of electronic devices poses challenges related to electronic waste. Sustainable material choices, responsible manufacturing practices, and the incorporation of eco-friendly disposal options are critical to mitigating the environmental impact of smart glasses.

The literature emphasizes the importance of an ethical and environmentally conscious approach throughout the entire life cycle of the technology, ensuring that its benefits are maximized while minimizing any potential negative consequences.

3.3 Framework for environmental consideration

Developing a comprehensive framework involves a holistic approach that spans the entire lifecycle of the technology. The framework aims to create smart glasses that not only meet the needs of visually impaired users but also adhere to environmentally responsible practices, promoting a sustainable and ethical approach to technology development. By integrating the following considerations into the development process:

1. Material Selection:

- Objective: Choose materials with a focus on sustainability, recyclability, and reduced environmental impact.

- Actions:

- Prioritize materials with a lower ecological footprint.

- Consider the life cycle of materials, from extraction to disposal.

- Explore recycled or upcycled materials for manufacturing.

2. Manufacturing Processes:

- Objective: Minimize the environmental impact of the manufacturing phase.

- Actions:

  - Adopt energy-efficient manufacturing processes.

  - Implement waste reduction and recycling programs in production facilities.

  - Consider local sourcing to reduce transportation-related carbon footprint.

3. Energy Efficiency:

- Objective: Optimize the energy consumption of the smart glasses to reduce overall environmental impact.

- Actions:

  - Implement power-saving features in the design.

  - Utilize energy-efficient components and technologies.

  - Explore renewable energy sources for charging or operation.

4. End-of-Life Considerations:

- Objective: Address the electronic waste issue by planning for responsible disposal or recycling.

- Actions:

  - Design smart glasses with modular components for easier disassembly.

- Provide information on proper disposal methods for users.

- Collaborate with e-waste recycling programs for responsible product disposal.

5. Lifecycle Assessment:

- Objective: Conduct a thorough assessment of the environmental impact throughout the entire life cycle of the product.

- Actions:

- Perform a life cycle analysis to identify environmental hotspots.

- Continuously monitor and improve environmental performance.

6. Regulatory Compliance:

- Objective: Ensure compliance with environmental regulations and standards.

- Actions:

- Stay informed about relevant environmental regulations.

- Obtain certifications such as RoHS (Restriction of Hazardous Substances) for compliance.

7. User Education and Engagement:

- Objective: Inform and involve users in environmentally conscious practices.

- Actions:

- Provide information on sustainable use and maintenance of smart glasses.

- Encourage users to participate in recycling programs.

8. Continuous Improvement:

- Objective: Foster a culture of continuous improvement in environmental practices.

- Actions:

  - Regularly assess and update the environmental framework.

  - Seek and implement innovations for further sustainability gains.

4 CHAPTER 4: DETAILED DESIGN

4.1 Detailed Specifications

Embark on an insightful exploration into the detailed specifications of our groundbreaking smart glasses, where innovation converges with precision. In this segment, we meticulously unravel the intricacies of the hardware components, shedding light on the design, functionality, and integration of each element. Our commitment to excellence extends beyond the tangible, delving into the sophisticated realms of software intricacies that power these cutting-edge glasses. Additionally, we navigate the intricate landscape of cloud services, illuminating the seamless connectivity and collaborative synergy between the smart glasses and the cloud ecosystem. From high-caliber hardware design to the intricacies of software and cloud integration, this exploration is a testament to our dedication to crafting a seamlessly connected and intelligently functioning wearable technology.

Our design process for this project has many steps, those steps being:

1. Hardware specifications and details
2. Accessing the raspberry pi zero w
3. Accessing the desktop environment
4. How to utilize Google Cloud
5. Streaming server
6. Text Detection
7. Object Detection
8. Text-to-Speech
9. Client software initialization

10. Client software Process Loop

4.1.1 HARDWARE SPECIFICATIONS AND DETALILS

A. Raspberry Pi Zero W:

The Raspberry Pi Zero W is a compact and affordable single-board computer developed by the Raspberry Pi Foundation. It is an enhanced version of the original Raspberry Pi Zero, featuring built-in wireless connectivity. Here are some key details about the Raspberry Pi Zero W:

1. **Form Factor:**
    - The Raspberry Pi Zero W has an ultra-compact form factor, measuring 65mm x 30mm x 5mm.
2. **Processor:**
    - It is powered by a Broadcom BCM2835 application processor with a single-core ARM1176JZF-S CPU clocked at 1 GHz.

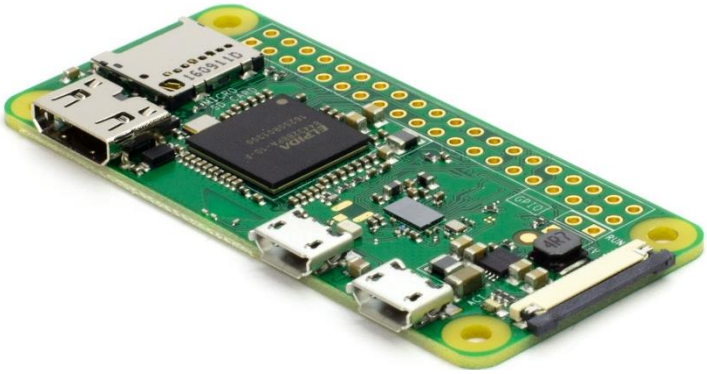


Figure 4.1 Raspberry Pi Zero W

3. **Wireless Connectivity:**
    - The "W" in Raspberry Pi Zero W stands for wireless. It includes built-in wireless LAN (Wi-Fi 802.11n) and Bluetooth 4.1, providing easy connectivity without the need for additional peripherals.
4. **Memory:**
    - The board is equipped with 512MB of RAM, allowing for reasonable performance in various computing tasks.
5. **Ports and Connectors:**
    - Mini HDMI and USB On-The-Go (OTG) ports.
    - Micro USB power input.
    - 40-pin GPIO header, compatible with the Raspberry Pi Model A+/B+/2B/3B.
    - HAT-compatible (Hardware Attached on Top) header.
6. **Operating System:**

- Supports various operating systems, including Raspbian, a Linux distribution based on Debian, designed specifically for Raspberry Pi devices.

7. **Camera and Display Connectors:**
    - The board features a CSI (Camera Serial Interface) connector for connecting the official Raspberry Pi Camera Module.
    - There's also a DSI (Display Serial Interface) connector for connecting the official Raspberry Pi Touchscreen Display.
8. **Power Consumption:**
    - The Raspberry Pi Zero W has low power consumption, making it suitable for projects with power constraints or battery-powered applications.
9. **Headless Operation:**
    - With the built-in wireless capabilities, the Raspberry Pi Zero W is suitable for headless operation, meaning it can be accessed and configured without the need for a dedicated display, keyboard, or mouse.
10. **Projects and Use Cases:**
    - IoT (Internet of Things) projects
    - DIY electronics projects
    - Portable computing projects
    - Low-power embedded systems
    - Educational projects
11. **Availability and Cost:**
    - The Raspberry Pi Zero W is an affordable option, making it accessible for a wide range of hobbyists and developers.
12. **Community and Support:**
    - The Raspberry Pi Zero W benefits from the extensive Raspberry Pi community, providing a wealth of resources, tutorials, and community support.

B. 8 Megapixels Raspberry Pi Camera Board V2:

The official camera module for the Raspberry Pi with an 8-megapixel sensor is referred to as the "Raspberry Pi Camera Module V2." Below is information about this camera module:

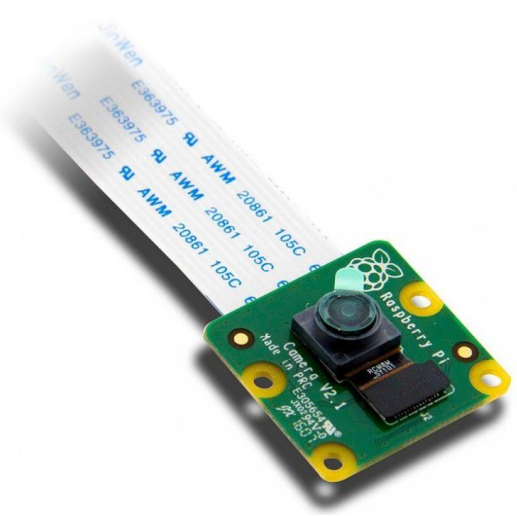


Figure 4.2 Raspberry Pi 8MP Camera

1. **Megapixels:**
    - The Raspberry Pi Camera Module V2 features an 8-megapixel Sony IMX219PQ Exmor sensor.
2. **Sensor Type:**
    - The camera module uses a back-illuminated sensor, which is designed to provide improved performance in low-light conditions.
3. **Fixed Focus:**
    - The lens on the Camera Module V2 is a fixed-focus lens, meaning it does not have a manual or automatic focus adjustment. This makes it suitable for various applications but may require consideration of subject distance.
4. **Field of View (FOV):**
    - The camera module has a horizontal field of view of approximately 62.2 degrees and a vertical field of view of around 48.8 degrees.
5. **Video Capabilities:**
    - It supports high-definition video recording, including 1080p video at 30 frames per second (fps), 720p video at 60fps, and 640x480p video at 90fps.
6. **Connection:**
    - The Camera Module V2 connects directly to the CSI (Camera Serial Interface) port on the Raspberry Pi board.
7. **Ribbon Cable:**
    - It uses a ribbon cable for connection, providing flexibility in camera placement.
8. **Compatibility:**
    - The Camera Module V2 is compatible with various Raspberry Pi models, including the Raspberry Pi 4, Raspberry Pi 3, Raspberry Pi 2, and Raspberry Pi Zero.
9. **Still Image Capture:**
    - The camera module supports still image capture in various formats, including JPEG and RAW.

10. **Night Vision:**
    - While the Camera Module V2 itself does not have infrared capabilities for true night vision, external infrared lighting solutions can be used for low-light conditions.
11. **Projects:**
    - The Camera Module V2 is widely used in projects such as photography, time-lapse recording, computer vision applications, surveillance systems, and robotics.
12. **Software Support:**
    - The official Raspberry Pi operating system, Raspbian (now known as Raspberry Pi OS), includes support for the camera module. Additionally, there are various third-party software libraries and applications available for extended functionality.

C. Dual battery 18650 Dual Output 3.3 V and 5V USB Module charger:

A "Dual Battery 18650 Dual Output 3.3V and 5V USB Module Charger" typically refers to a charging module designed to accommodate two 18650 lithium-ion batteries simultaneously, providing dual output voltages of 3.3V and 5V through USB ports. Here's some general information about such a module:

Figure 4.3 Dual 18650 Dual Output

1. **Purpose:**
    - This module is designed to charge two 18650 lithium-ion batteries and provide dual output voltages for powering or charging devices that require 3.3V or 5V.
2. **Battery Compatibility:**
    - Specifically designed for 18650 lithium-ion batteries, which are widely used due to their compact size and high energy density.
3. **Dual Output:**
    - Offers dual output voltage options:
        - 3.3V: Typically used for powering low-power electronics and microcontrollers.
        - 5V: Commonly used for charging USB-powered devices such as smartphones, tablets, or other USB gadgets.
4. **USB Ports:**
    - Equipped with USB ports for easy connection to various electronic devices using standard USB cables.

5. **Charging Circuit:**
    - Includes a charging circuit that allows the simultaneous charging of two 18650 batteries. The charging process is typically controlled to prevent overcharging and ensure the safety of the batteries.
6. **Voltage Regulation:**
    - The module may include voltage regulation circuits to stabilize the output voltages and ensure a consistent power supply to connected devices.
7. **Protection Features:**
    - May include protection features such as overcurrent protection, overvoltage protection, and short circuit protection to safeguard both the batteries and connected devices.
8. **LED Indicators:**
    - Often equipped with LED indicators to display the charging status and power output status.
9. **Compact Design:**
    - Designed to be compact and portable, making it suitable for various applications, including DIY projects, portable power banks, and electronic prototypes.
10. **Applications:**
    - DIY power banks.
    - Portable electronic projects.
    - Battery-powered devices requiring 3.3V or 5V.
    - Educational projects involving power supply solutions.
11. **Usage Considerations:**
    - Ensure that the module is used with compatible 18650 lithium-ion batteries.
    - Follow the manufacturer's guidelines for charging and usage.
    - Avoid overloading the module beyond its specified current and voltage ratings.

D. Raspberry Pi Zero Case:

Raspberry Pi Zero cases are protective enclosures designed specifically for the Raspberry Pi Zero and its variants. Cases serve to protect the delicate components of the Raspberry Pi Zero, provide access to ports, and offer a platform for additional accessories. Below is general information about Raspberry Pi Zero cases:

1. **Compatibility:**
    - Raspberry Pi Zero cases are specifically designed to accommodate the Raspberry Pi Zero and its variants, including the Raspberry Pi Zero W.
2. **Material:**
    - Cases are commonly made from various materials, including plastic, acrylic, or metal. The choice of material can impact durability, heat dissipation, and the overall aesthetic of the case.

Figure 4.4 Raspberry Pi Zero Case

3. **Design and Form Factor:**
    - Raspberry Pi Zero cases come in a variety of designs and form factors. Some cases are sleek and minimalistic, while others have a more robust design with additional features.
4. **Access to Ports:**
    - A well-designed case provides easy access to the GPIO (General Purpose Input/Output) pins, USB ports, HDMI port (if applicable), microSD card slot, and other essential ports on the Raspberry Pi Zero.
5. **Ventilation:**
    - Many cases include ventilation or slots to help dissipate heat generated by the Raspberry Pi Zero during operation. This is especially important in scenarios where the Raspberry Pi is running resource-intensive tasks.
6. **Assembly:**
    - Raspberry Pi Zero cases are typically designed for easy assembly. They often consist of multiple pieces that snap or screw together, providing a secure fit for the Raspberry Pi.
7. **Camera and Display Compatibility:**
    - Some cases are designed to be compatible with the official Raspberry Pi Camera Module or Display. These cases may include a slot or mount for easy integration.
8. **Mounting Options:**

- Some cases come with mounting options, such as slots for wall mounting or mounting holes for attaching the Raspberry Pi Zero to a surface.

9. **Color Options:**
    - Raspberry Pi Zero cases are available in a variety of colors, allowing users to choose a case that suits their preferences or matches their project aesthetics.
10. **Accessibility Features:**
    - Some cases feature additional ports or slots for specific accessories, such as headers for attaching additional HATs (Hardware Attached on Top) or hats to expand the functionality of the Raspberry Pi Zero.
11. **Brands and Models:**
    - Various companies manufacture Raspberry Pi Zero cases, and there are both official cases from the Raspberry Pi Foundation and third-party options available.

E. Raspberry Pi Zero Camera Cable:

The Raspberry Pi Zero Camera Cable, also known as the Camera Serial Interface (CSI) cable, is a ribbon cable designed to connect the Raspberry Pi Camera Module to the Raspberry Pi Zero or Raspberry Pi Zero W. Here is some information about the Raspberry Pi Zero Camera Cable:

1. **Compatibility:**
    - The camera cable is specifically designed for use with the Raspberry Pi Zero and Raspberry Pi Zero W. It connects to the CSI port on the Raspberry Pi Zero board.
2. **Length:**
    - The cable comes in various lengths to accommodate different setups. Common lengths include 15cm and 30cm.

Figure 4.5 Raspberry Pi Zero Cable

3. **Flexibility:**
    - The cable is flat and flexible, allowing it to be easily routed in various configurations within projects. Its flexibility is essential for accommodating different orientations and arrangements of the camera module.
4. **Connector Types:**

- The cable has a ribbon-like structure with a flat connector at each end. One end is designed to connect to the CSI port on the Raspberry Pi Zero, and the other end connects to the camera module.

5. **CSI Port:**

    - The CSI port (Camera Serial Interface) on the Raspberry Pi Zero is a specific connector for interfacing with the Raspberry Pi Camera Module. It is a ribbon connector with a locking mechanism to ensure a secure connection.

6. **Camera Module Compatibility:**

    - The cable is designed to work with various Raspberry Pi Camera Modules, including the official Raspberry Pi Camera Module V1 and V2. These camera modules have different specifications, but they share compatibility with the CSI interface.

7. **Installation:**

    - To install the camera cable, you need to carefully insert the flat connectors into the corresponding CSI port on the Raspberry Pi Zero and the camera module. The connectors typically have a small latch that needs to be opened before inserting or removing the cable.

8. **Orientation:**

    - Pay attention to the orientation of the connectors when inserting the cable to ensure a proper and secure connection. Misalignment can lead to connection issues or damage to the cable.

Figure 4.6 Raspberry Pi Zero Camera Cable

9. **Usage:**

    - The Raspberry Pi Camera Module, when connected using the camera cable, allows the Raspberry Pi Zero to capture still images and record video. It is commonly used in projects involving photography, video recording, computer vision, and more.

10. **Replacement:**

    - In case of a damaged or malf0unctioning camera cable, it can be replaced with a new one to restore the camera's functionality.

4.1.2 ACCESSING THE RASPBERRY PI ZERO W

During the development process, it was essential to access the Raspberry Pi for the installation of relevant scripts and the client required for server communication. This also involved setting up any necessary interfaces.

It's noteworthy that the Raspberry Pi doesn't come with a pre-installed operating system. To address this, the procedure involves flashing an image onto an SD card. To execute this task, the official RPI Imager [12] is employed. Specifically, a version of Raspberry Pi OS "Buster" Legacy is flashed, accompanied by pre-configuration of certain files to enable wifi-access and SSH functionality.

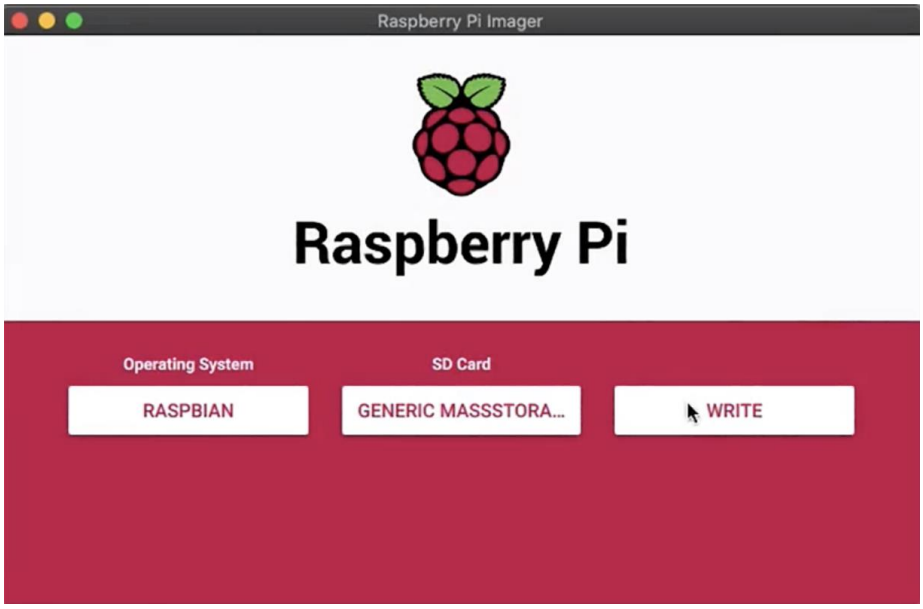


Figure 4.7 Raspberry Pi OS

Once the OS is successfully burned onto the SD card and the device is powered on, patience is required as the OS boots up and establishes a connection with the local Wi-Fi. This connection is crucial for subsequent access via SSH.

To initiate the SSH connection, you'll need to enter the Raspberry Pi's hostname (or IP address) and the associated username into any terminal emulator. This action prompts a password request, where you must provide the user's password to gain access to the device's operating system.

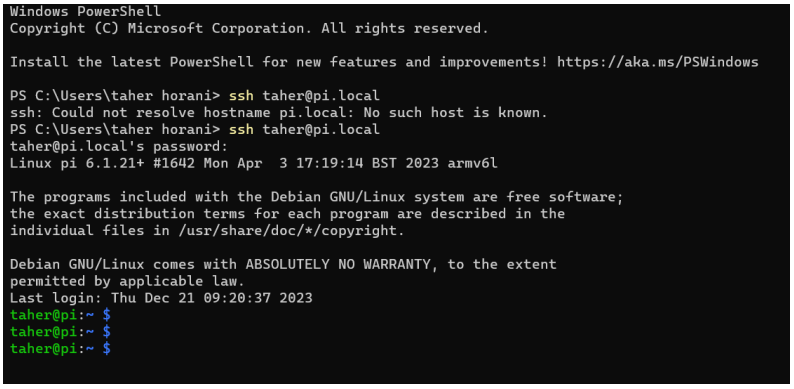


Figure 4.8 Accessing Raspberry pi using SSH via Windows PowerShell

4.1.3 ACCESSING THE DESKTOP ENVIRONMENT

The installed OS version features a user-friendly desktop environment that proves useful for device development. However, to access this environment, it is necessary to enable VNC (Virtual Network Computing) sessions on the Raspberry Pi. This can be accomplished by editing the raspi-config program that is included with the OS.

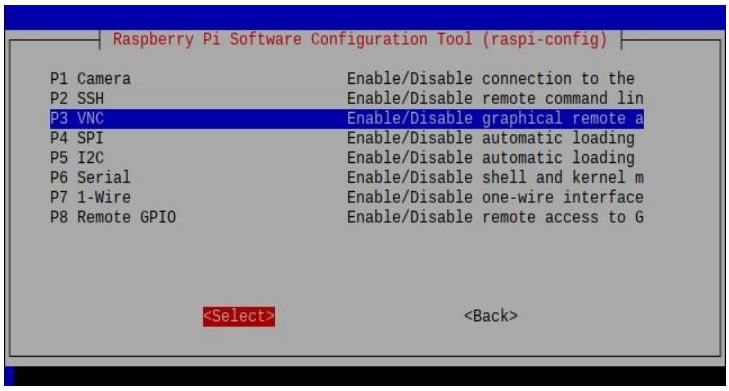


Figure 4.9 Raspberry Pi configuration software

After enabling VNC sessions, the next step involves establishing a remote connection between the laptop and the Raspberry Pi using the RealVNC program [13]. To connect to the Raspberry Pi, you must enter the device's IP address and password into the designated fields. Upon successful completion of these steps, remote access to the Raspberry Pi is established using RealVNC from the laptop.

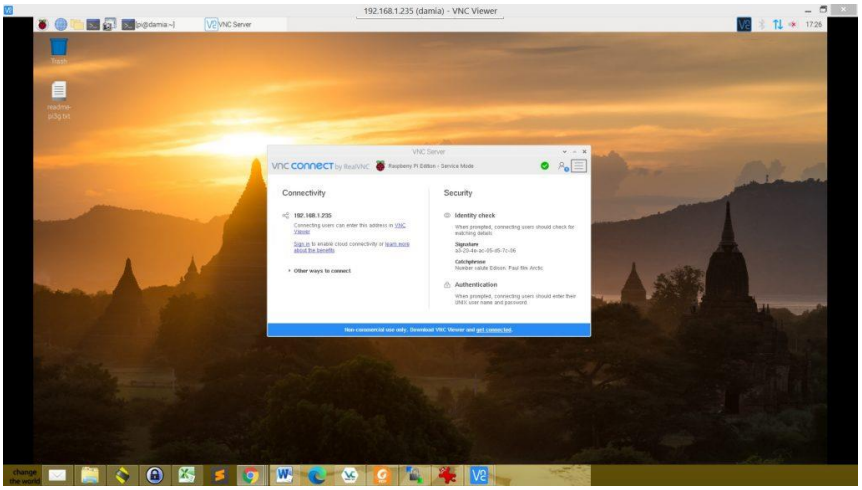


Figure 4.10 RealVNC viewer on desktop

4.1.4 HOW TO UTILIZE GOOGLE CLOUD

In our architecture, Google Cloud plays a pivotal role, serving as a platform for hosting, computing, and providing various services. Leveraging a cloud platform offers the advantage of dynamic server development and ensures compatibility with hardware clients, eliminating the need for frequent updates when the server undergoes changes. A significant asset in our setup is the Google Cloud Vision APIs suite, allowing rapid experimentation and development of computer vision features without the necessity of training models, unless desired in the future.

Accessing the platform commenced with the registration of a Google account, followed by the creation of a project. This project serves as the foundation for enabling and managing all Google Cloud services. Our initial focus was on the Google Cloud Vision service, where we enabled various endpoints, empowering our project to process images as required, particularly for object detection and text detection. Additionally, we activated the Google Cloud Text-to-Speech service [14], granting us the capability to synthesize text into voice using AI-generated voices.

4.1.5 STREAMING SERVER

We need to establish a server that can communicate with both the Raspberry Pi, as well as our Google Cloud platform environment, and we will do all that using Go [15]. Go is a high-level programming language that is statically typed, the language is open source and designed by Google made to have C-like performance while still maintaining memory safety using a garbage collector and achieving high-concurrency using a CSP model [16].

Our server will use the websockets protocol [17] to establish a continuous channel with the client which will be used to read messages sent, as well send messages back to the client. Here are the steps necessary to create a websockets server using Go:

1. Initialize a Go project using the commands "**go mod init <name_of_the_project>**", which will create a new Go project in the current directly and create a new Go module using the name you have set.
2. Start by creating a "**main.go**" (appendix A) file in which we will write the main function that will be the main entry point of our server program execution in which we will create an **HTTP** endpoint that will be accessible to the client.
3. Install the package "**gorilla/websocket**" which is the most widely used package for websockets in Go using the command "**go get github.com/gorilla/websocket".** [18] Then inside the HTTP handler for the endpoint, add a function that will upgrade the connection to a websockets, which we then could read and write to.
4. We also established a goroutine channel named "**readChan**" for exchanging messages between goroutines, this will be used by the websockets connection to send the data to other goroutines.

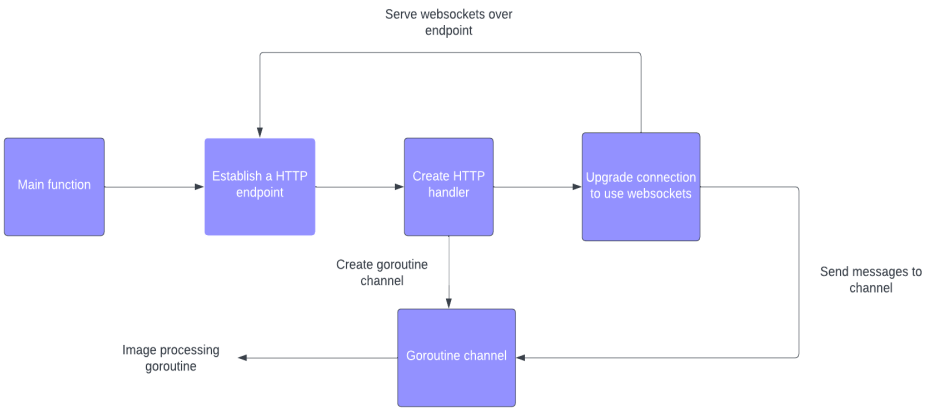


Figure 4.11 Goroutine channel

By following these steps, we successfully implemented a server that can handle websockets, enabling us to send and receive video frames from the client.

In summary, this was the first step in implementing a server that will handle both video frame receiving as well as implementing the first steps in processing using Google Cloud with the usage of Go channels.

4.1.6 TEXT DETECTION

Text detection is the ability for our system to analyze and recognize text from video frames being sent by a client as accurately as possible. We will achieve that functionality by using Optical Character Recognition (OCR) service from Google Cloud, as it will give us high accuracy with low latency at a cheap price compared to training our own model.

Once our system has established a websockets connection with the client, and the client starts sending video frames through our connection, the server creates a separate goroutine to handle sending the requests to the Google Cloud service, while still being able to read new frames at the same time, this approach makes sure that the system remains responsive and is the key to achieving near-real time results. The Go code responsible for this operation is named "text_detection.go" in (appendix A).
Here are the steps to achieve our text detection process:

1. We needed to initialize the Google Cloud client, which will facilitate communication with our Cloud environment, so we downloaded the Google Cloud Go SDK [19] using the command "**go get cloud.google.com/go/vision/apiv1**", this added the library and dependencies necessary to send requests and receive responses from our environment. We initialized the client in our "**main.go**" file in our server code by passing a credentials file that contained secret keys that are used for authentication.

2. The process starts by creating a goroutine which first loops over any messages that have been put in the "**readChan**" channel, if it does not find a message it does nothing but once it reads bytes from the channel, it builds the necessary image struct needed for the request. The code which does that can be found in the (appendix A)

3. Once the request image has been built, we will need to pass a "**imageContext**" struct [20], which contains important contextual information for the client, such as the language to detect which in our case will be arabic and english, and then pass the created image request by using the client's function which is responsible for text detection called "**DetectText**". This function then returns a response struct, which contains all the response information or could return an error value which means the request has failed.

4. Retrieve the response, which consists mainly of a couple of key fields which will be used in the text-to-speech process, those mainly being the textual information, the confidence of the detection, as well as the locale detected (language detect).

Here is a flowchart depicting the process:

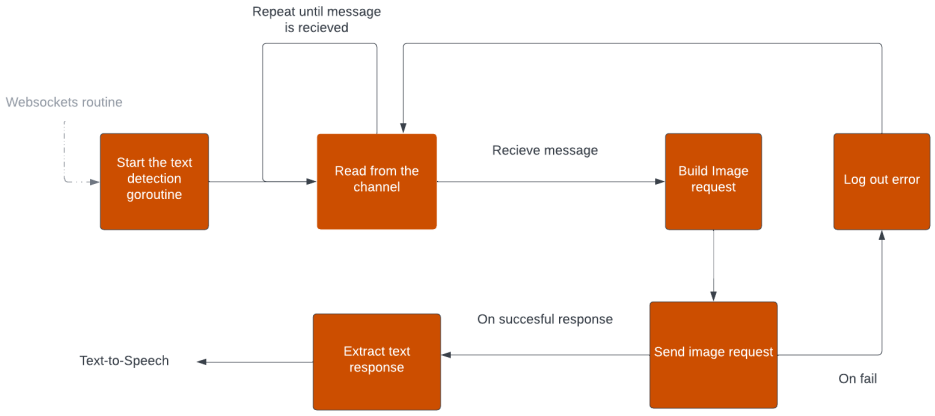


Figure 4.12 Text detection

By following these steps, we were able to add multi-concurrency in our system which allows the system to be extremely responsive, along with the addition of using Google Cloud services which made our system extremely flexible and modular.

4.1.7 OBJECT DETECTION

Object detection is the ability for our system to analyze and recognize objects from video frames being sent by a client as accurately as possible. We will achieve that functionality by using Google Vision API label detection service from Google Cloud [21], as it will give us high accuracy with low latency at a cheap price compared to training our own model.

Once our system has established a websockets connection with the client, and the client starts sending video frames through our connection, the server creates a separate goroutine to handle sending the requests to the Google Cloud service, while still being able to read new frames at the same time, this approach makes sure that the system remains responsive and is the key to achieving near-real time results. The Go code responsible for this operation is named "object_detecion.go" in (appendix A).

While the process of object detection is near identical to text detection, one key difference is that unlike text detection, the object detection sends many responses per object detected in the image, so we took the label information per object and concatenated it into a single text to reduce the requests on the text-to-speech routine.

Here are the steps we took to achieve that:

1. We used the same client that was initialized for both text and object detection, just like the previous step, as well as create the same goroutine with the same functionality as before.

2. Once the request image has been built, we will need to pass a "**imageContext**" struct, which contains important contextual information for the client, such as the language to detect which in our case will be arabic and english, and then pass the created image request by using the client's function which is responsible for text detection called "**LocalizeObject**" [22]. This function then returns a response struct, which contains all the response information or could return an error value which means the request has failed.

3. Retrieve the response, which consists mainly of all the responses for every object detected in the frame, which each have a couple of key fields which will be used in the text-to-speech process, those mainly being the name of the object, and the confidence score of the detection.

4. We looped over the array of response objects that was sent before and checked if the score was 60% or above for each object, if it did succeed, we grabbed the name of the object and concatenated it inside a string variable which will be sent for the text-to-speech routine.

Here's a flowchart demonstrating the process we took:

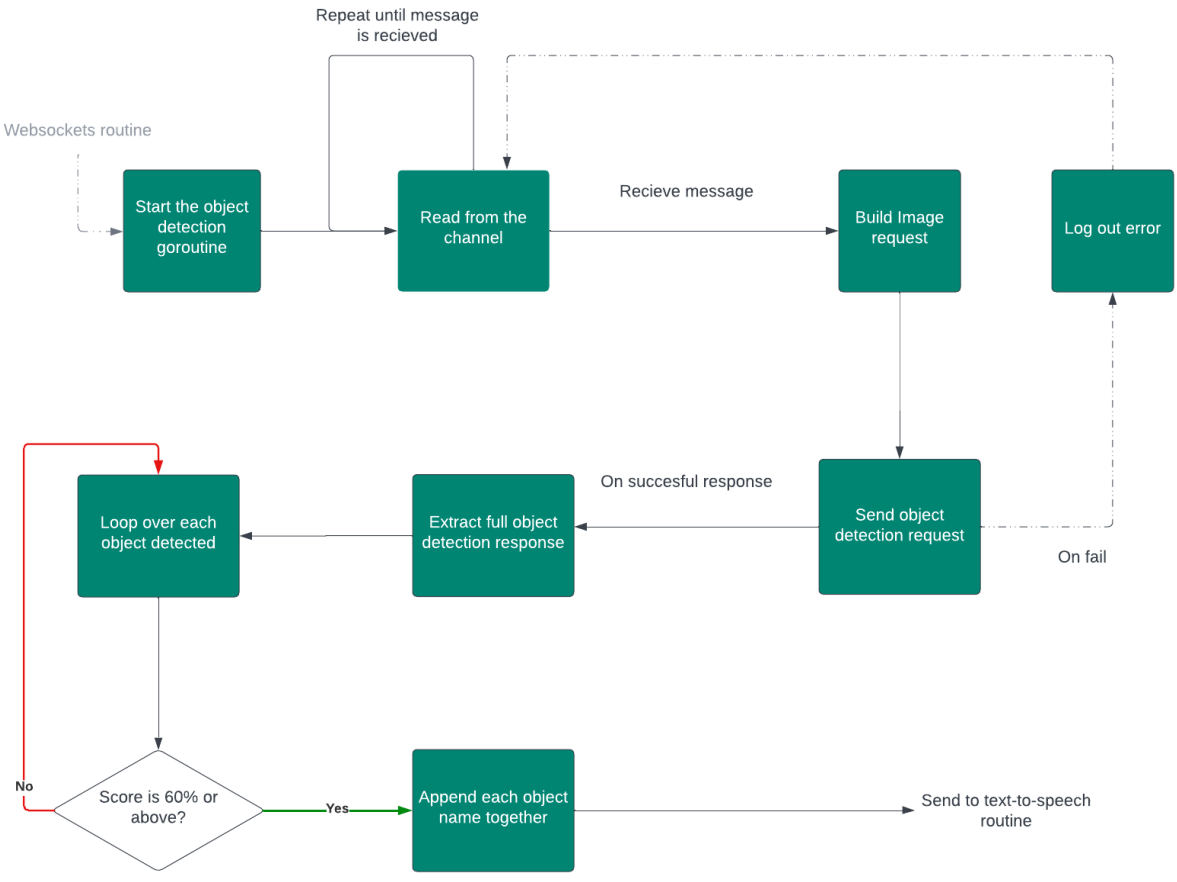


Figure 4.13 Object detection

Once we have successfully implemented these steps, we had a performant and responsive set up as we intended and are now ready to piece the second part of our design which is the text-to-speech aspect.

4.1.8 TEXT-TO-SPEECH

Text-to-Speech allows us to create natural-sounding, synthetic human speech as playable audio. Text-to-Speech is ideal for any application that plays audio of human speech to users. It allows us to convert arbitrary strings, words, and sentences into the sound of a person speaking the same things.

Text-to-Speech creates raw audio data of natural, human speech. Text-to-Speech has a wide selection of custom voices available for us to use. The voices differ by language, gender, and accent (for some languages).

Along with other, traditional synthetic voices, Text-to-Speech also provides premium, WaveNet-generated voices. Users find the Wavenet-generated voices to be more warm and human-like than other synthetic voices.

The key difference to a WaveNet voice is the *WaveNet model* used to generate the voice. WaveNet models have been trained using raw audio samples of actual humans speaking. As a result, these models generate synthetic speech with more human-like emphasis and inflection on syllables, phonemes, and words. [23]

Since WaveNet provides a more human-like voice output, we have decided to use it, but beware that it will incur an additional cost if it does, as it is 4x more expensive than standard voices.

This is the next step of the application is to retrieve from either the object detection process or text process, and build a request that will be sent off to the service, using a different client which facilitates communication between the serve and text-to-speech service, which will be initialized in the "main.go" at the start of execution, along with the previous vision client.

To build a text-to-speech request, we need to utilize a struct which will contain all configuration information necessary, those being:

1. The input text, which will be the output of either detection methods as we have discussed above.

2. The voice selection parameters, which specify the name of the model to use (en-US-Wavenet-A), and the language code (en-US).

3. The audio configuration parameters, which will consist of the audio encoding that can either be MP3 or WAV, in which we chose MP3 due to its compression and smaller size that will reduce bytes sent to the client, the sound profile that will be calibrated to, speaking rate, and pitch

Our full configuration will be in the method "createTTSRequest" in the (appendix A)

Here's the steps we took to implement text-to-speech routine:

1. Once we received the request text output from either routines, we will start by sending it, the necessary text-to-speech client, and a context object to "**DisplayResults**" method which will run in the same goroutine as either detection method to ensure that it will only be run if the detection was successful.
2. The method will take the text input and create a request object to pass to the client, using the configuration information we discussed above.
3. The request struct will be then used by the text-to-speech client along with the context object for tracing, which if successful will return the response containing the audio content bytes that can be retrieved by the method "**GetAudioContent**". If it was not successful it will return an error.
4. Once the process is over, the resulting bytes slice will be written on the websockets channel, using "**WriteMessage**" and will be sent back to the client ready to be played.

Here's a flowchart depicting the operation:

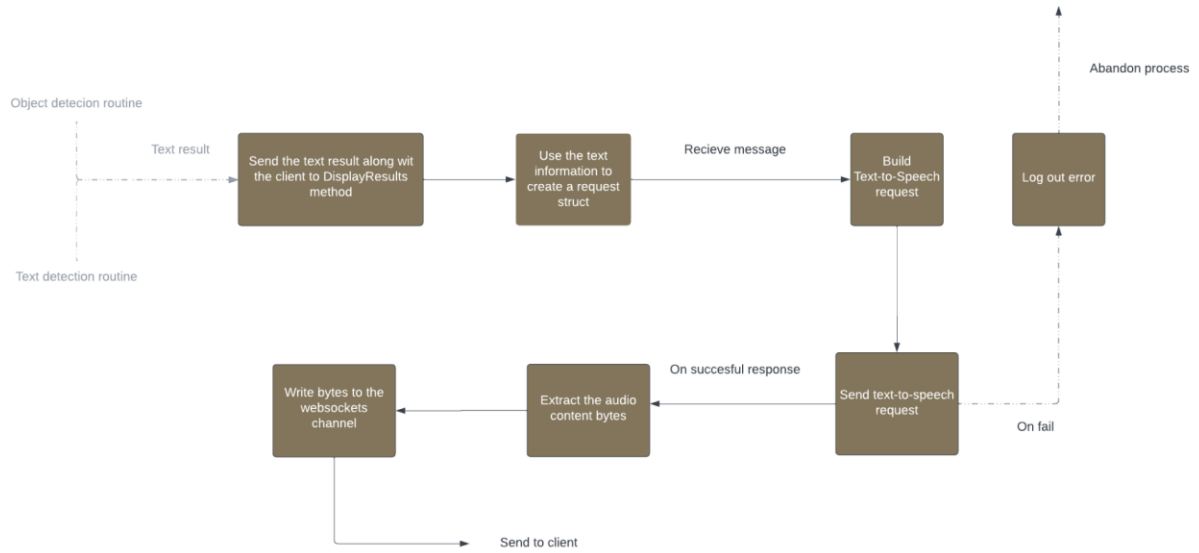


Figure 4.14 text-to- speech flow

With this we are able to complete our server loop, the server is able to do this process many times per second as it is written with concurrency in mind so that none of the steps are a bottleneck to the

operation, as well as being very modular as it can be changed as algorithms develop or requirements change, without affecting the client side in any major way.

In summary, the server reads video frames from the websockets channel that are sent by the client, then puts them into either an object detection or text detection, retrieving the result and transforming it into a text string, then that string is sent off the text-to-speech process, returning the audio bytes and completing the loop by sending the bytes to the websockets channel, so that the client can read the bytes and process them accordingly.

4.1.9 CLIENT SOFTWARE INITIALIZATION

The client side is simple in design, as we have decided to not add any complexities that will involve different hardware requirements, which makes it applicable to any hardware, so long that we have an internet connection and a processor that can handle media decoding and encoding.
We achieve that by defining the client as just an I/O device only, that is not responsible for any computations or heavy processing. This makes sure that no matter how the server changes, the client can still remain backwards compatible with any upgrades, without requiring client software updates.

The first step that the client software does is start the hardware devices necessary for our operation, those mainly being the audio interface, which is done by using the "**oto**" [24] sound library from Go and the video device, which is done by using the "**v4l2**" library native to any Linux OS [25], that can interface with golang using the bindings made by the "**go4vl**" go library. We also have some default configurations and parameters which will be used in the process loop, those being the camera resolution, the websockets server URL, the sound parameters which consist of audio channel count, sample rate, and format. This code snippet is in the (appendix A)

Once the devices are running and operational, the client begins the connection by using the websockets server URL endpoint, which will be the main entry point of any communication from and to the server. The client also creates a goroutine channel, named "**messageChannel**" specifically to facilitate communication between processes so that not only do we achieve multi concurrency on the server size, but as well as the client side, which makes the client responsive and almost real-time without causing any unknown blocking or issues.

The client then creates a separate goroutine process which reads any messages that are sent from the server and does a check, if the bytes read of that channel are above 0, which means that we have a message, it sends those bytes to the "**messageChannel**" to be used by a separate goroutine process which will be responsible for playing the sound bytes.

Here are the steps we took to achieve all that:

1. Begin by downloading the necessary libraries and dependencies, by using the commands "**go get github.com/gorilla/websocket**", "**go get github.com/ebitengine/oto**", and "**go get github.com/vladimirvivien/go4vl**" respectively, and then creating initialization structs for configurations.

2. Initialize the audio device, using the audio configuration options that we specified using the context struct, as well as initializing the video device, providing the encoding format, width and height.

3. Once the devices are ready, begin the websockets connection by using the server URL and waiting for the connection to be ready

4. Start a go routine that will be responsible for reading messages from the websockets connection, as well as sending them to the process channel for further usage.

Here's the flowchart that depicts this operation:

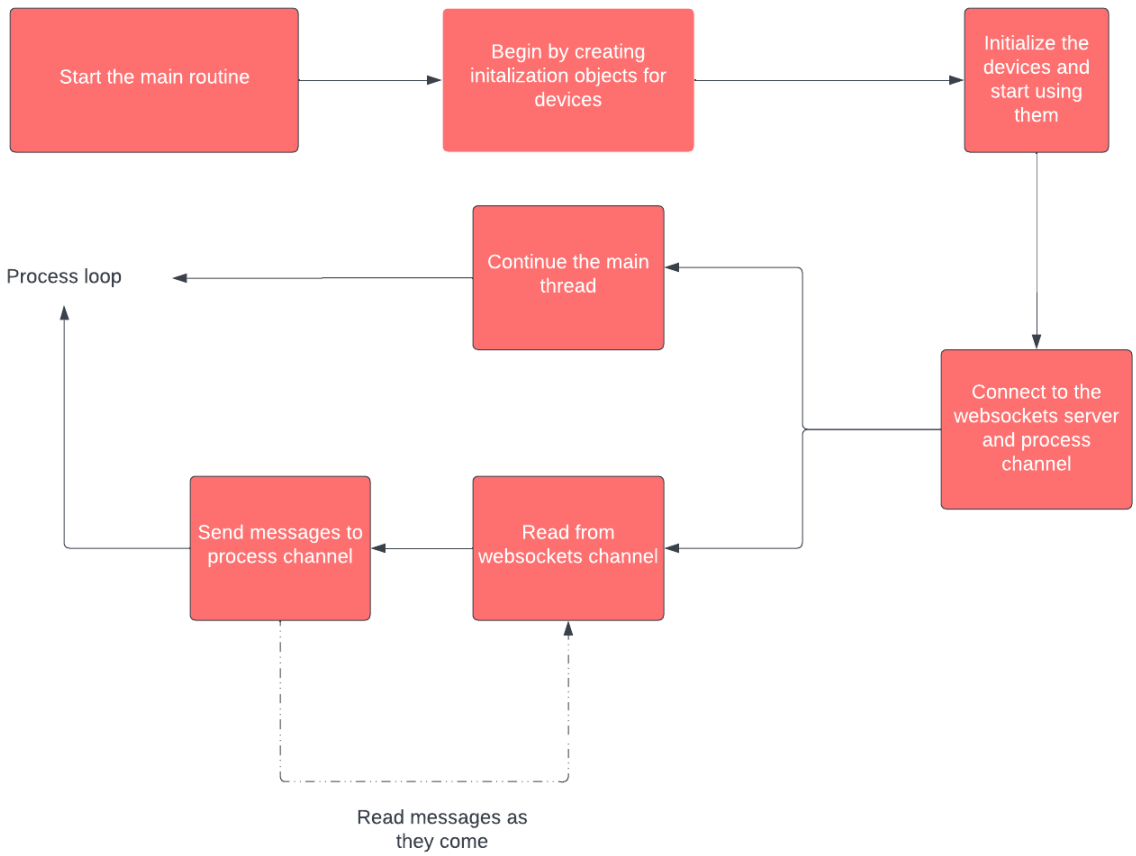


Figure 4.15 Client software initialization

This is the first and initial step in beginning the process loop that will receive video frames from the webcam device, send them to the websockets server, and then receive back audio frames that will be used for the final output.

4.1.10 CLIENT SOFTWARE PROCESS LOOP

The next step of the client process is to begin the process loop, which will be responsible for being the input and output of our system at the same time, and we are able to achieve that using the process channel that we defined earlier, we also heavily use a golang primitive called "**select statements**" which allow us to change the program behaviour depending the message received from a channel, this allows our code to be concise yet streamlined and efficient.

The first part of this loop is that it runs on the main thread, and then reads video frames from the video device library, if it receives a video frame, it immediately sends it to the websockets server for processing.

Second part is to read from the "**messageChannel**" process channel, that receives the websockets message from the separate goroutine, which contains the sound bytes, we then create a byte's reader object that will then be used to create an MP3 decoder, the program can now use that decoder to play the audio. Playing the audio will require us to make the thread sleep for as long as the player is still playing, once it's done, the loop will repeat, and the process will start again.

Here's the steps to achieve our process loop:

1. Initialize a loop that will run until the program is terminated, the loop consists of mainly a "**select statement**" which selects an execution flow depending on the type of message received from a variety of channels and inputs.
2. For the case when we receive a video frame from the video device, we will immediately send the frame bytes to the websockets channel, make the thread sleep for a set amount of time, which is to control the rate at which frames are being sent, to not overload the system, we found that 300ms is a decent sweet spot. After that the loop starts all over again

3. For the case when we receive a websockets message, from the "**messageChannel**", we will start playing audio by:
    a. creating a new byte's reader object from the bytes sent
    b. creating an mp3 decoder object from the byte's reader
    c. using that mp3 decoder to create a new audio player object.
    d. using the player to play the audio, making sure to sleep the thread for as long as the audio is playing for, to continue execution after the audio has finished playing.
4. After the client has finished playing the audio the loop starts all over again.

Here's a flowchart depicting the whole operation:

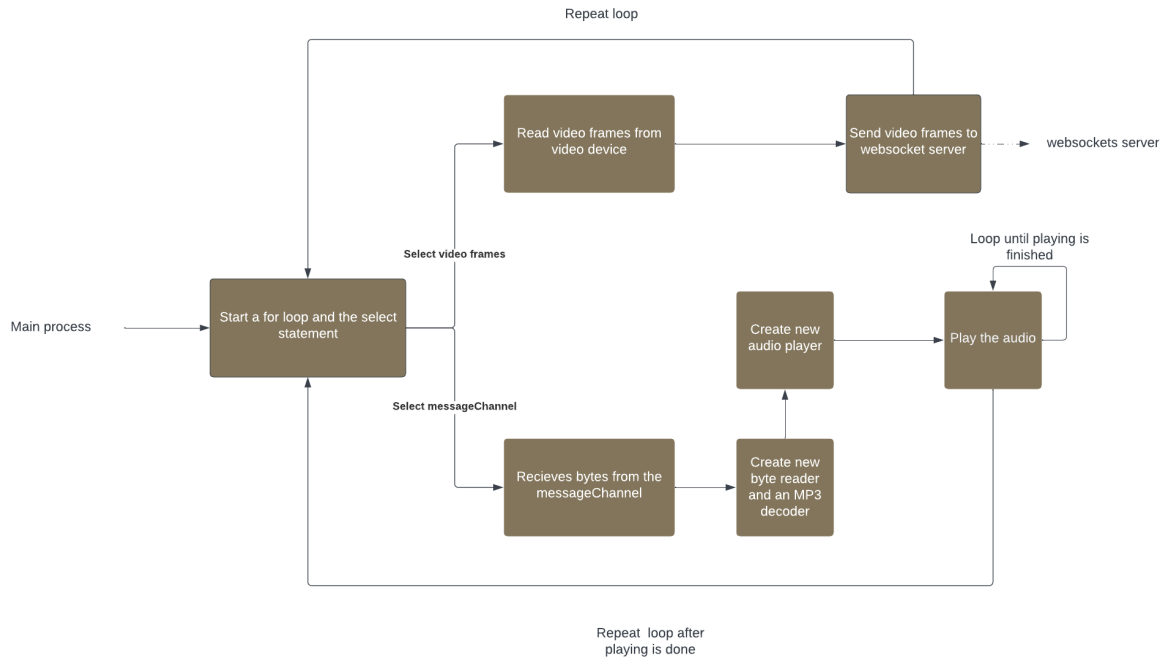


Figure 4.16 Client workflow

This completes the final loop and process of the architecture, and concludes the design in great detail. This design can be subject to change, but for our purposes of demonstration it serves it well.

4.2 Discussion of Detailed Design Alternatives

Converting the functionalities of smart glasses into a smart band is an interesting design alternative with several potential advantages. Here's a breakdown of your points:

Additional enhancements:

Special sensors: Smart bands can accommodate a wider range of sensors than smart glasses due to their larger size and proximity to the body. This opens up the possibility for features like advanced health monitoring (e.g., ECG, blood pressure), environmental sensing (e.g., air quality, UV radiation), and even fall detection for improved safety.

Better battery utilization: Smart bands generally have lower power requirements compared to smart glasses due to the limited display and processing capabilities. This translates to longer battery life, a major pain point for many wearables.

Safety functionalities:

Compact and wearable: Smart bands are lightweight and unobtrusive, making them more comfortable and practical for everyday wear compared to smart glasses. This is especially important for safety features like fall detection or emergency SOS, which need to be readily available.

Other potential benefits:

Cost-effective: Smart band typically require less complex technology and materials compared to smart glasses, potentially leading to lower production costs and wider market reach.

Improved user experience: Smart bands can offer a more intuitive and discreet way to interact with information compared to smart glasses, which can sometimes feel obtrusive or require specific gestures.

However, it's important to consider some potential drawbacks as well:

Limited display capabilities: Smart bands have smaller screens compared to smart glasses, which can hinder information display and interaction. This might be a deal-breaker for tasks that require visual clarity or detailed data visualization.

Reduced interaction and information access: Smart bands rely primarily on touch and voice input, which can be less convenient or accurate compared to the multi-modal interaction options available with smart glasses. Accessing information might also require taking out your phone, breaking the seamless flow of information.

Privacy concerns: Wrist-worn devices raise privacy concerns related to data collection and potential misuse of health and activity information. Addressing these concerns through transparent data practices and user control mechanisms is crucial.

Overall, converting smart glass functionalities into a smart band is a promising design alternative with its own set of advantages and disadvantages. Carefully weighing the trade-offs and considering the specific use case will determine whether this approach is the optimal solution. (not done)

4.3 Relevant Engineering Applications and Calculations

In this section, we explore the diverse engineering applications and calculations integral to the development of our smart glasses project, designed to cater specifically to the visually impaired community. The project encompasses innovative features aimed at addressing crucial challenges faced by individuals with visual impairments.

Improving Reading for Weak Eyesight:

One of the primary objectives of our smart glasses is to enhance the reading experience for individuals with weak eyesight. Leveraging advanced image processing and optical character recognition (OCR) technologies, our system facilitates the seamless conversion of printed or digital text into audible content. The engineering calculations involved in optimizing OCR accuracy and real-time text processing form a critical aspect of this endeavor.

Onsite Quality Control for Workers:

Our smart glasses extend their utility to the industrial sector, serving as a tool for onsite quality control for workers. Equipped with sophisticated object detection algorithms, the glasses can aid in identifying and assessing product defects or irregularities on the manufacturing floor. The engineering calculations here involve precision and accuracy parameters, ensuring reliable real-time feedback to enhance overall quality control processes.

On-Demand Translation of Text:

The incorporation of on-demand translation features elevates the utility of our smart glasses on a global scale. By utilizing advanced language processing algorithms, the glasses enable users to receive instant translations of printed or digital text. This involves intricate engineering calculations related to language recognition, translation accuracy, and real-time linguistic processing.

These diverse engineering applications underscore the versatility and societal impact of our smart glasses project. The calculations involved in optimizing these features are meticulously crafted to ensure both functionality and efficiency, aligning with our mission to provide practical solutions for the visually impaired while pushing the boundaries of engineering innovation.

4.4 Description of the Final Design

The culmination of our smart glasses project results in a refined and comprehensive final design that seamlessly integrates cutting-edge technologies to cater to the unique needs of the visually impaired community.

Figure 4.17 Final Design

The final design encapsulates the following key components and features:

1. Hardware Configuration:

- Compact and lightweight frame to ensure comfort during prolonged use.

- Embedded camera module for capturing real-time visual information.

- Integrated microcontroller, optimized for efficient image processing and computational tasks.

2. Optical Character Recognition (OCR) Technology:

- Advanced OCR algorithms for accurate and rapid text detection from printed or digital sources.

- Real-time processing capabilities to convert detected text into audible content for the user.

3. Object Detection System:

- State-of-the-art object detection algorithms facilitating the identification and interpretation of objects within the user's surroundings.

4. Text-to-Speech Functionality:

- Natural and expressive text-to-speech synthesis for relaying information to the user.

- Customizable voice options and settings to accommodate user preferences.

5. Battery Efficiency:

- Efficient power management strategies to optimize battery life during typical daily use.

- Quick and convenient charging capabilities for sustained and reliable performance.

The final design represents a harmonious amalgamation of engineering precision, user-centric design, and practical functionality. By addressing challenges faced by the visually impaired, our smart glasses aim to empower users with greater independence and an enhanced quality of life. This design embodies our commitment to leveraging technology for positive societal impact.

5 CHAPTER 5: PROJECT REALIZATION AND PERFORMANCE OPTIMIZATION

5.1 Analysis and Optimization

Our software development journey commenced with exploratory proof of concepts in Python. This choice facilitated rapid iteration. Initially, we constructed a basic server employing WebRTC. However, during initial testing, significant delays emerged, rendering it unsuitable for our system. We then experimented with SRT [26], a newer media protocol. Regrettably, due to sparse documentation and lacking libraries, our attempts to implement it were unsuccessful. For video capture and detection, we employed OpenCV [27] and its corresponding Python package. This combination expedited the processing of video footage from webcams.

Our initial design functioned as a proof of concept but required substantial optimization. We observed a predominantly synchronous process that introduced unnecessary delays. Transitioning certain operations, such as sending requests to Google Cloud, to asynchronous methods offered the potential for swiffer execution. Furthermore, we identified an opportunity for optimization by substituting a Websockets connection for a media server. While simpler and real-time, this approach necessitated a robust connection due to the large message sizes. While OpenCV served its purpose in the proof of concept, its heavyweight nature on the client side posed a challenge. Given our reliance on OpenCV solely for video capture functionality, maintaining a lightweight client package was imperative.

Entering the optimization phase, we made a strategic shift from Python to Go for software development. This transition offered multiple advantages. Go's static typing and compiled nature delivered faster computational speeds compared to Python, preventing any potential bottlenecks attributed to Python's reputation for being slower and resource-intensive. Additionally, Go's concurrency model empowered us with built-in concurrency and parallelism capabilities through Goroutines, facilitating rapid development of concurrent features. Subsequently, we opted to replace OpenCV for video capture with a library providing v4l2 bindings. This alternative allowed us to capture video on the client side without the burden of OpenCV's extensive dependencies.

This approach not only yielded high performance and multi-concurrency but also enabled us to keep package usage lightweight while minimizing external dependencies.

5.2 Methods of Realizing Final Design

Our design process for this project comprises several crucial steps, each contributing to the development and realization of the final smart glasses design. Below are the key methods employed in bringing our vision to life:

1. Hardware Specifications and Details:

Define and document the specific hardware components, including camera modules, microcontrollers, and other essential elements crucial for the functioning of the smart glasses.

2. Accessing the Raspberry Pi Zero W:

Outline the steps involved in accessing and configuring the Raspberry Pi Zero W, ensuring seamless integration with the overall system architecture.

3. Accessing the Desktop Environment:

Detail the procedures for accessing the desktop environment on the Raspberry Pi, essential for configuring and managing various aspects of the smart glasses software.

4. Utilizing Google Cloud:

Describe the integration of Google Cloud services, highlighting their role in enhancing functionality, such as language processing, data storage, or other relevant features.

5. Streaming Server:

Explain the setup and configuration of the streaming server, a critical component for real-time visual information processing and feedback.

6. Text Detection:

Detail the methods and algorithms employed for text detection, covering the integration of Optical Character Recognition (OCR) technologies and any customizations made for our specific application.

7. Object Detection:

Outline the steps taken to implement robust object detection capabilities, including the choice of algorithms, training processes, and integration with the overall system.

8. Text-to-Speech:

Specify the methods utilized for implementing text-to-speech functionality, encompassing the selection of synthesis engines, voice options, and the integration process.

9. Client Software Initialization:

Provide insights into the initialization process of the client software on the smart glasses, ensuring a seamless and user-friendly experience for end-users.

By meticulously addressing each of these steps in the realization process, we ensure the cohesiveness and functionality of the final design. This methodology encompasses both hardware and software aspects, emphasizing a holistic approach to creating a technologically advanced and user-centric solution for the visually impaired community.

5.3 Sustainability

Our system was designed with sustainability as a key objective across multiple dimensions. To ensure future compatibility, we prioritized backward compatibility with server changes. This approach allows seamless integration of server updates, including enhanced models, new features, or software redesigns, without necessitating client-side software updates or hardware replacements. Maintaining a singular point of connection to the server through websockets ensures compatibility regardless of the server's behavior, as long as it accurately sends responses.

Another significant sustainability aspect involves our hardware selection, specifically concerning the power source. Our system's power source comprises double lithium-ion batteries held within a dedicated holder. This design allows for the replacement of exhausted batteries and enables recharging using a specific charger, ensuring prolonged usability. Additionally, since the power is supplied via a micro-USB cable, the battery pack can be detached, facilitating the use of common power banks. This design choice significantly minimizes electronic waste associated with battery power usage.

These deliberate measures underscore our commitment to environmental sustainability, ensuring system longevity, adaptability to server upgrades, and minimizing electronic waste generation.

5.4 Testing and Improvement

Throughout our system testing phase, we engaged in iterative evaluations of our architecture's functionalities. These evaluations were aimed at gauging their performance and pinpointing areas necessitating improvement. This meticulous process allowed us to uncover specific aspects within our system that demanded enhancements.

During our testing phase, we discovered that the messages transmitted from the client to the websockets were excessively large to manage effectively. The client, utilizing a 720p resolution for image transmission, resulted in considerable delays. To address this challenge, we made a strategic decision to lower the video capture resolution to 360p. This reduction significantly decreased the overall message size, enabling us to achieve expedited message processing and substantially improved response times.

An optimization strategy we implemented involved addressing the limitations of the OpenCV library for Go. We found that the OpenCV library, being large in size and requiring static building and compilation, was unsuitable for our lightweight client. Since our usage of OpenCV was solely for video capture, we opted for the Go v4l2 library. This library offered native video capture bindings compatible with various Linux operating systems. This transition significantly reduced our bundle size and resulted in faster compile and operation times, aligning better with our requirements.

During our server-side analysis, we observed that the Google Cloud Text Detection API returned a collection of results. The initial element contained the entire detected paragraph or text, followed by subsequent results containing individual words from the detected text. This approach proved inefficient for our requirements, as we did not necessitate processing each word separately, focusing solely on the complete text section.
To address this inefficiency, we explored the request body parameters and identified a solution. By setting a specific parameter that controlled the number of desired results, we configured the request to return only the whole text section in a single response. This optimization significantly enhanced the system's responsiveness and processing speed, eliminating the need to handle individual words from the detected text.

A minor yet impactful optimization on the server-side involved addressing static video frames that remained unchanged for extended periods. Recognizing that processing identical frames was inefficient and diminished the user experience, we implemented a solution. Our approach involved comparing the current video frame with the previous one. If they were identical, we optimized the process by skipping iteration, thus bypassing unnecessary frame processing.

This optimization effectively mitigated the redundant processing of static frames, enhancing the overall efficiency of our server's performance. With these server-side optimizations in place, we achieved a satisfactory level of performance for our system.

The initial optimizations implemented in our system provided a satisfactory level of performance. However, our overarching goal was to emphasize the adaptability and flexibility of our architecture. We aimed to create a system that remained open to change and upgrades without necessitating alterations to its fundamental core. This adaptability was a cornerstone of our architectural approach, enabling us to easily accommodate future optimizations. Our focus on ensuring the system's readiness for change

reaffirms our commitment to continuous improvement and flexibility, ensuring that we can readily introduce further optimizations as needed in the future.

5.5 Health, Safety, Quality and Reliability

The health, safety, quality, and reliability considerations are paramount to ensure a robust and trustworthy assistive technology solution. From a health perspective, the system is designed with user well-being in mind, incorporating ergonomic principles to minimize discomfort during prolonged usage. Strict safety measures are implemented to prevent potential hazards, such as ensuring that the device does not impede the user's natural awareness of their surroundings. Quality assurance is ingrained in every aspect of the system, from the selection of durable materials to stringent manufacturing processes, guaranteeing a high-quality end product. The reliability of the smart glasses is upheld through rigorous testing protocols, including stress testing of hardware components and continuous monitoring of software functionalities. Fail-safes and redundant systems are integrated to mitigate potential risks and enhance overall system reliability. Additionally, the system adheres to industry standards and regulatory requirements to ensure compliance with safety and quality benchmarks. By prioritizing health, safety, quality, and reliability, the smart glasses system not only provides an innovative solution for the visually impaired but also establishes a foundation of trust and dependability for users relying on this assistive technology in their daily lives.

5.6 Performance Evaluation

After completing the design process, it was essential to evaluate the system's performance. This evaluation helped us identify strengths and areas for improvement and optimization. This will showcase some measurements and key areas where the system could be further improved upon.

There are multiple criterias that we were determined to test with this system, being a key goal with our design is to have great performance while still remaining cost-effective and scalable, we had to test the latency of request, the accuracy of the results, and the overall system metrics.

5.6.1 INTRODUCTION TO THE TESTING ENVERONMENT AND SUITE

In our initial server evaluation, we analyzed response times for cloud-based tasks such as text detection, object detection, and text-to-speech. Additionally, we closely monitored CPU and memory utilization to

gauge resource efficiency. These evaluations served as a foundation to understand responsiveness and resource consumption, guiding our optimization strategies.

Our evaluation was conducted within our local environment, utilizing a laptop as the server to obtain an initial estimation. A 100 Mb/s fiber internet connection facilitated these tests. Notably, outcomes may vary based on server specifications, hosting environment, and internet connectivity. The server specifications are detailed below:

1. **Model**: ThinkPad E470
2. **CPU**: 3.1GHz Intel i7-7500U 7th Gen processor
3. **RAM**: 20 GB DDR4
4. **Storage**: 500 GB Kingston SATA SSD
5. **OS**: Arch Linux distribution
6. **Network**: 100 Mb/s WIFI fiber connection

To emulate the client's behavior, we'll develop a testing client that replicates real client actions. This simulation will involve reading images from a specified directory, selecting a random image during each iteration, converting it into bytes, and transmitting it to the server via the established websockets connection. The server will then undergo its processing routine without alterations. Finally, the client will log the process results.

Testing Client Flow:

1. Initialize connection to the server as usual.
2. Instead of obtaining frames from a video input, frames are sourced from a directory.
3. Randomly select a frame from the directory.
4. Send the selected frame to the server and await completion of the process.
5. Log the results of the process.

Here's a flow chart depicting the testing process:

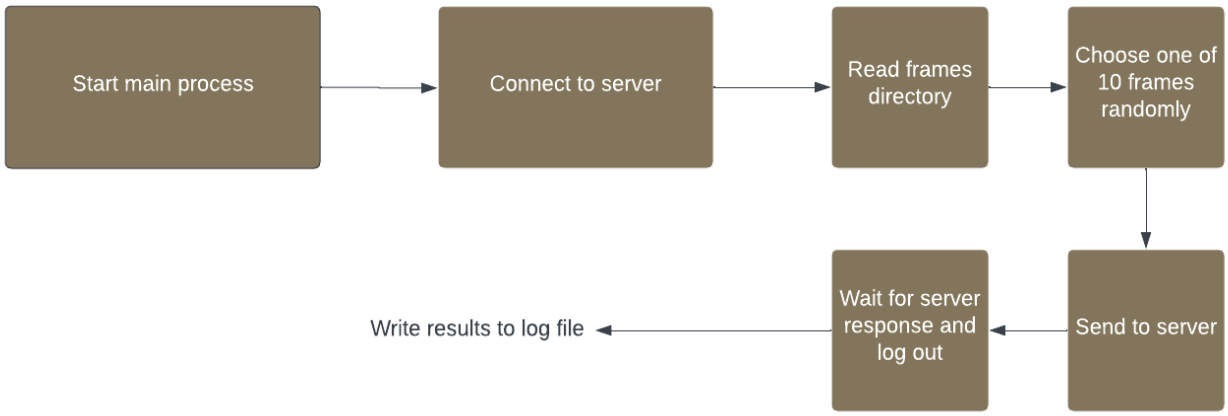


Figure 5.1 Testing Method

With this, we are now able to conduct the initial testing and performance evaluation of our system, and gain insightful metrics that will lead to improvements as we upgrade our system down the line.

5.6.2 SERVER LATENCY EVALUATION

Our initial evaluation focused on text detection performance using a diverse image set comprising 10 images that were not altered and captured randomly and formatted using JPEG format, which is the same format as our video capture. These images encompassed variations in fonts, text colors, and lengths, including five images in Arabic and five in English, simulating real-world scenarios. Small text images are classified by having less than 20 words, and are bigger in font, to simulate store banners or page titles, while large text images are considered to be longer, with smaller fonts, simulating paragraphs in novels or articles.

We iterated over the images in a random fashion, to avoid any unwanted caching, in total each image was tested over 103 times, with the system running for a total of 1040 iterations. The purpose of this test was to check the latency of the requests, it does not take into account latency of the client to server.

With that in mind, here are the results of our tests:

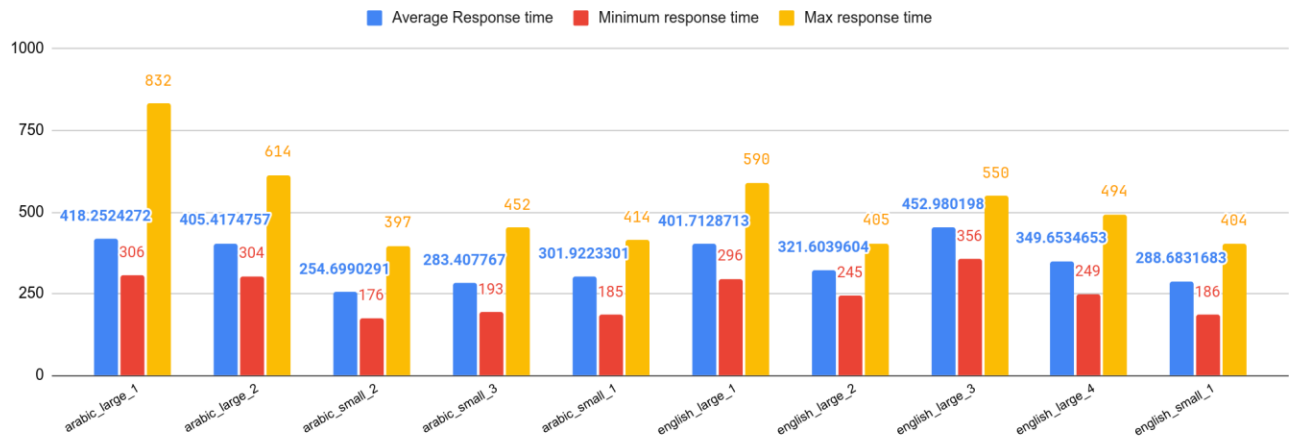


Figure 5.2 Text detection response time

For each image, we measured three crucial parameters:

1. **Average Response Time**: Represents the mean time taken for the server's response across multiple iterations.
2. **Minimum Response Time**: Signifies the shortest duration observed in server response time among the iterations.
3. **Maximum Response Time**: Reflects the longest duration observed in server response time among the iterations.

The test yielded insightful results in the form of response time statistics for individual images. The columns in the provided data correspond directly to the image file names, with each filename retaining its original naming convention, comprising the language and text size information.
Key observations derived from the test outcomes include:

- **Consistent Average Response Time**: Across all images, the average response time consistently ranged between 300 to 400 milliseconds. This consistency was observed irrespective of the language used or the size of the text within the images.

- **Predictability in Response Time**: The analysis indicates that the system maintains a stable and predictable response time for text detection. This predictability is beneficial as it suggests that the system's performance remains constant and reliable, regardless of variations in client operations or video frame complexities.

Similar to the previous test iterations, a series of requests were made, focusing explicitly on the text-to-speech functionality. This involved initiating the calculation timing immediately before transmitting the request to the cloud service.

The primary goal of this test was twofold:

1. **Latency Assessmen**t: Accurately quantify and analyze the latency associated with the text-to-speech process.
2. **Language Influence Analysis**: Determine if the selection of different languages had a noticeable impact on the calculated latency metrics.

Here are the results of the arabic text-to-speech:

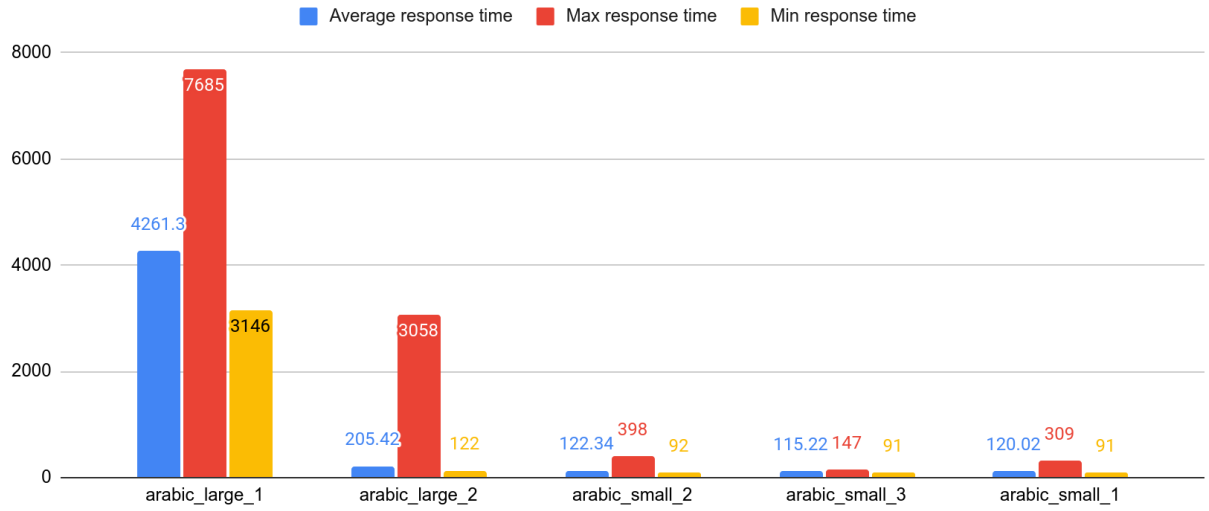


Figure 5.3 Arabic text to speech response time

And here are the results of the english text-to-speech:

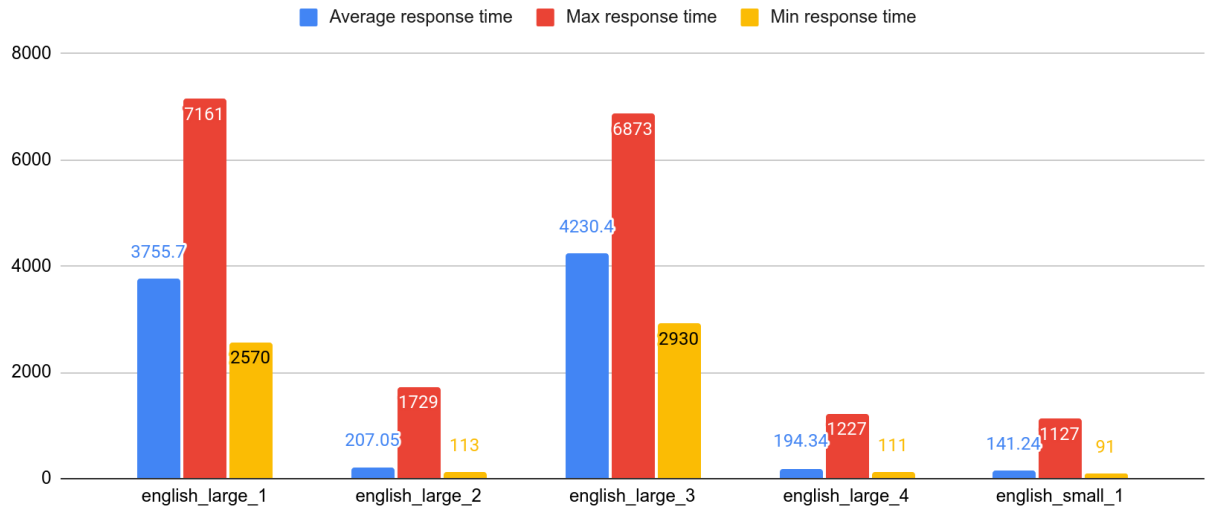


Figure 5.4 English text to speech response time

Our tests have shown that there's a relation between the amount of text and the response time, we found this to be quite predictable as the text-to-speech ai needs to analyze each word of the text and create a sound for it, the larger the text, the higher the response time. We also found that latency does not differ based on the language, to our surprise, we thought that Arabic language would be harder to synthesize but the results have shown that they are more or less quite the same.

5.6.3 SERVER ACCURACY EVALUATION

Our accuracy evaluation focused on the output of the text detection performance, using a diverse image set comprising 10 images that were not altered and captured randomly and formatted using JPEG format, which is the same format as our video capture. These images encompassed variations in fonts, text colors, and lengths, including five images in Arabic and five in English, simulating real-world scenarios. Small text images are classified by having less than 20 words, and are bigger in font, to simulate store banners or page titles, while large text images are considered to be longer, with smaller fonts, simulating paragraphs in novels or articles.

We then examined the text these images consisted off and ran a single round of detection and examined the boundary boxes of the detect text, as well the detected text content by eye, as there was no score or accuracy parameter that we could use, we determined that as long as the detected text was close to the original material and the box perimeter encompassed the text, then we could verify the accuracy.

Here are side-by-side comparisons to the original image and text output respectively:

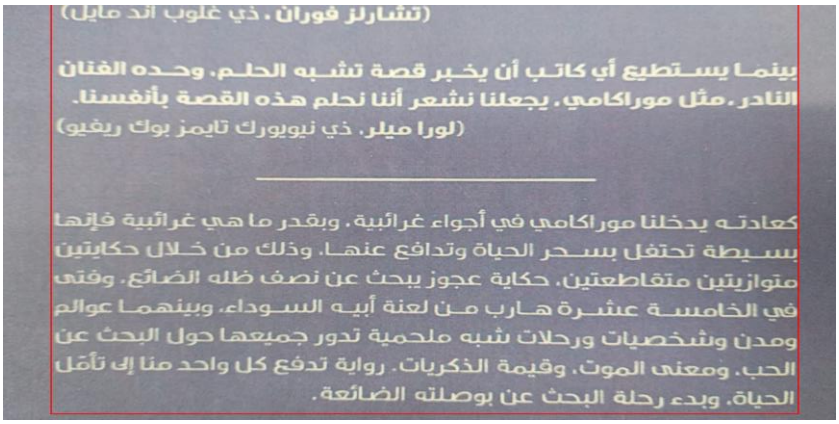


Figure 5.5 The selected text is in arabic

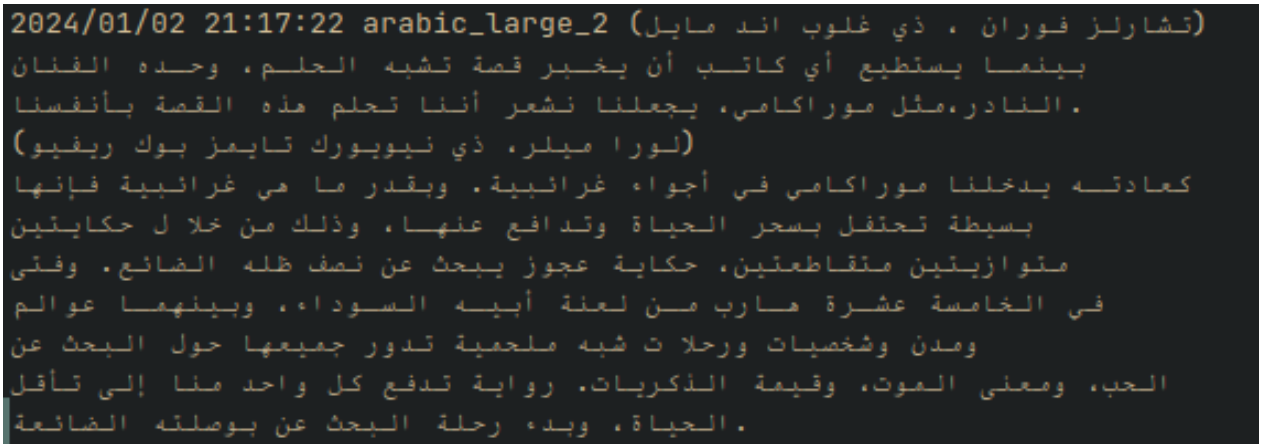


Figure 5.6 Output Arabic text detection

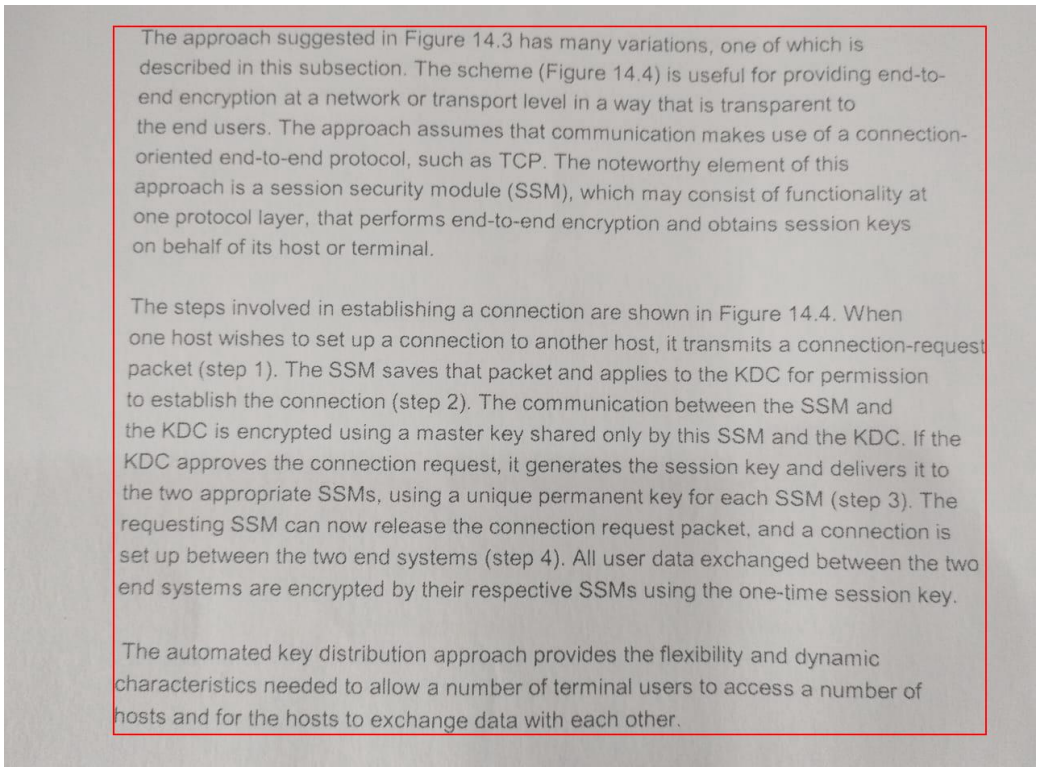


Figure 5.7 The selected text is in english

```
2024/01/02 21:17:33 english_large_3 The approach suggested in Figure 14.3 has many variations, one of which is
described in this subsection. The scheme (Figure 14.4) is useful for providing end-to-
end encryption at a network or transport level in a way that is transparent to
the end users. The approach assumes that communication makes use of a connection-
oriented end-to-end protocol, such as TCP. The noteworthy element of this
approach is a session security module (SSM), which may consist of functionality at
one protocol layer, that performs end-to-end encryption and obtains session keys
on behalf of its host or terminal.
The steps involved in establishing a connection are shown in Figure 14.4. When
one host wishes to set up a connection to another host, it transmits a connection-request
packet (step 1). The SSM saves that packet and applies to the KDC for permission
to establish the connection (step 2). The communication between the SSM and
the KDC is encrypted using a master key shared only by this SSM and the KDC. If the
KDC approves the connection request, it generates the session key and delivers it to
the two appropriate SSMS, using a unique permanent key for each SSM (step 3). The
requesting SSM can now release the connection request packet, and a connection is
set up between the two end systems (step 4). All user data exchanged between the two
end systems are encrypted by their respective SSMS using the one-time session key.
The automated key distribution approach provides the flexibility and dynamic
characteristics needed to allow a number of terminal users to access a number of
hosts and for the hosts to exchange data with each other.
```

Figure 5.8 Output English text detection

We conducted testing on several images for object detection. Subsequently, we reviewed the official accuracy results of the response, prompting us to reassess the accuracy of the detection to confirm its reliability:

Figure 5.9 Sample image

```
2024/01/02 22:41:16 Couch 0.88961065
2024/01/02 22:41:16 Chair 0.87312675
2024/01/02 22:41:16 Table 0.84815454
2024/01/02 22:41:16 Cat 0.64617825
2024/01/02 22:41:16 Packaged goods 0.56476885
```

Figure 5.10 accuracy result of the image

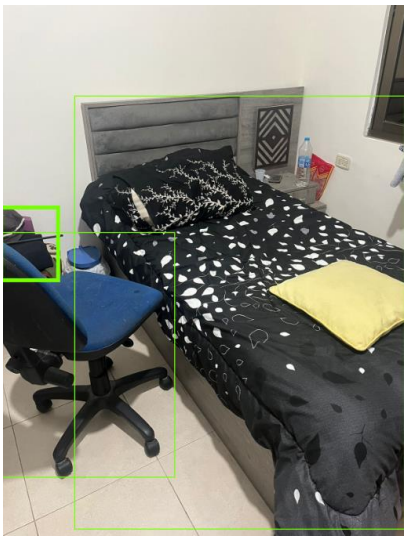


Figure 5.11 Sample image depicting a bedroom

```
2024/01/02 22:41:14 Time taken: 167
2024/01/02 22:41:15 Bed 0.86493665
2024/01/02 22:41:15 Chair 0.81867874
2024/01/02 22:41:15 Luggage & bags 0.52208406
```

Figure 5.12 detected accuracy

5.6.4 CLIENT METRICS

One important metric that was necessary to test for was the heat of our client system. After all, we do not want our hardware to overheat, causing discomfort, maybe causing injuries, and or cause a fatal error that will render the hardware useless.

My testing methodology tried to find out whether or not a bare Pi Zero will run at 100% continuously. We used PiCockpit's PiStats [28] to track temperature data and PiDoctor [29] to tell me if we have been throttled. If your Pi is throttled, you will also see a thermometer icon on your display.

There are three sections to this test. The first is completely idle (1% CPU load) which should show you what the baseline temperature is when you are doing nothing.
Then, there's the 35% CPU load test, which was done by running a Python script in the background.
Then, we ran stress-ng –cpu 4 –cpu-method fft to get CPU load to 100%. This command is one of the most power-hungry methods out there. The room temperature was about 22°C.

Here are the results:

CPU load=~1%	CPU load=~35%	CPU load= ~100%	Throttling?
42°C	57°C	74°C	No

Table 5.1 CPU Performance and Throttling Status at Different Loads

It appeared that the heat of the raspberry pi is manageable at low to mid temperatures but can reach unmanageable heat levels at higher cpu temperatures, but since it the raspberry pi is covered it won't have direct skin contact with the users.

6 CHAPTER 6: ECONOMIC, ETHICAL, AND CONTEMPORARY ISSUES

6.1 Final Cost Analysis

We divided the project cost into two parts:

6.1.1 HARDWARE COSTS

Our design process involved meticulous consideration of hardware components crucial to the functionality of our innovative wearable device. The following hardware cost breakdown details the financial investment required for each essential component:

Hardware Item	Cost (JOD)
Glasses	10
Raspberry Pi Zero W	31
8 Megapixels Raspberry Pi Camera Board V2	40
Dual Battery 18650 Dual Output Charger	8
Two 18650 3.7 LI-ION Rechargeable Batteries	9
Raspberry Pi Zero Case	5
Raspberry Pi Zero Camera Cable	2

Table 6.1 Hardware costs

This breakdown offers insights into the tangible expenses associated with the hardware components essential for the successful implementation of our project. The total hardware cost amounts to 105 JOD, providing a clear understanding of the financial investment in the tangible elements of our project.

6.1.2 GOOGLE CLOUD VISION API AND GOOGLE TEXT-TO-SPEECH PRICING

In addition to hardware considerations, our design process incorporates the utilization of cloud services, specifically Google Cloud Vision API and Google Text-to-Speech. Understanding the pricing structure for these cloud services is essential for accurate budgeting. The following table provides a comprehensive overview of the pricing details for various features:

Feature	First 1000 Units/Month	Units 1001 - 5,000,000/Month	Units 5,000,001 and Higher/Month
Object Detection	Free	$1.50	$1.00
Text Detection	Free	$1.50	$0.60

Table 6.2 Pricing structure of Google Vision API

Additional Pricing:

- Google Cloud Vision API (Word Detection): $1.50 per 1000 words
- Google Text-to-Speech: $4.00 per 1 million characters

This breakdown provides a comprehensive view of the project's financial considerations, encompassing both hardware and cloud service costs.

6.2 Commercializing the Project and Relevance to JORDAN and the Region

In our endeavor to commercialize the project, the primary focus is on creating a solution that makes a meaningful impact on the lives of visually impaired individuals. The relevance of our project to Jordan and the broader region lies in its potential to address challenges faced by blind people, enhancing their independence and accessibility.

- **Tailoring the Solution for Blind Individuals:**

The wearable device is specifically designed to cater to the unique needs of blind individuals. Through features such as object detection, text-to-speech capabilities, and real-time information access, our solution empowers visually impaired users to navigate their surroundings, read printed text, and access crucial information independently.

- **Accessibility and Affordability:**

We understand the importance of making assistive technology accessible to a wider audience. Our commercialization strategy prioritizes affordability, aiming to collaborate with local manufacturing partners and explore cost-effective production methods. This approach ensures that the technology reaches a broader segment of the blind population, including those in underserved communities.

- **Addressing Specific Challenges:**

Blind individuals often face challenges in mobility, education, and accessing information. Our wearable device directly addresses these challenges by providing real-time information about the environment, enabling educational support through text-to-speech capabilities, and facilitating greater independence in daily activities.

- **Collaboration with Blind Community Organizations:**

To enhance the relevance and impact of our project, we plan to collaborate closely with organizations dedicated to supporting the blind community. Engaging with these local stakeholders ensures that the technology aligns with the actual needs and preferences of blind individuals, fostering a user-centric approach.

- **Social Impact and Empowerment:**

The commercialization of our project is not solely about technology; it is about empowering blind individuals to lead more independent lives. By providing a solution that enhances accessibility and inclusivity, we aim to contribute to social development and positively impact the lives of visually impaired individuals in Jordan and the wider region.

In summary, our commercialization efforts are driven by a commitment to creating a solution that directly addresses the challenges faced by blind individuals. Through cultural sensitivity, affordability, and strategic collaboration, we aspire to make a lasting and meaningful impact on the lives of visually impaired individuals in Jordan and the broader region.

6.3 Relevant Code of Ethics and Moral Framework

Our project is underpinned by a comprehensive code of ethics and a robust moral framework that reflects our unwavering commitment to responsible and inclusive technology. At its core, our ethical code emphasizes inclusivity and accessibility, ensuring that our innovation benefits all individuals, regardless of their abilities or socio-economic background.

Cultural sensitivity is ingrained in our ethical practices. We aim to design technology that respects and aligns with the cultural norms and values of the communities we serve. Collaboration with local stakeholders, community leaders, and organizations is central to our approach, ensuring cultural relevance and respect.

Our commitment extends to responsible innovation with an emphasis on positive social impact. We recognize the profound responsibility that comes with technological advancement and strive to minimize environmental impact. Sustainable practices are integrated into our processes from design to manufacturing.

Briefly, our code of ethics and moral framework are foundational elements of our project, guiding our decisions and actions. We prioritize inclusivity, privacy, cultural sensitivity, and responsible innovation to ensure that our technology not only meets the highest ethical standards but also contributes positively to the well-being of individuals and communities.

7 CHAPTER 7: PROJECT MANAGEMENT

7.1 Task and Schedule

Software development life cycle (SDLC) models serve as navigational tools to guide teams through the intricate and demanding process of software development. In today's dynamic landscape, there exist over 50 recognized SDLC models, each with their own advantages and disadvantages tailored to specific software projects and teams. The success of a project in terms of quality, timelines, budget, and stakeholder satisfaction significantly relies on the chosen SDLC model. In our project, we have enthusiastically embraced the Agile Methodology.

Agile methodologies represent a set of principles and practices for effective project management that prioritize flexibility, collaboration, and rapid iteration. These methodologies prove particularly well-suited for projects involving intricate or rapidly evolving requirements. By adopting Agile, we are empowered to tackle the challenges of our project in a more dynamic, engaging, and efficient manner.

With Agile, we embark on a journey that embraces adaptability and embraces change as an inherent part of the software development process. Instead of rigidly adhering to a predefined plan, Agile allows us to flexibly respond to evolving client needs and market dynamics. Collaboration is at the heart of Agile, fostering a sense of unity within our team as we work closely together to deliver exceptional results.

Agile's iterative approach encourages us to continuously refine and enhance our software through frequent feedback loops. This iterative nature enables us to quickly incorporate valuable insights and make necessary adjustments to deliver a product that not only meets but exceeds stakeholder expectations. By embracing regular feedback and iteration, we strive for continuous improvement and are better equipped to adapt to the ever-changing demands of the project.

In our pursuit of Agile, we bring a spirit of enthusiasm, professionalism, and a dedication to creating software that truly adds value. We are excited to embark on this journey, confident that Agile will enable us to navigate the complexities of our project effectively and efficiently. With Agile as our guiding light, we look forward to delivering outstanding results that propel our project to new heights of success.

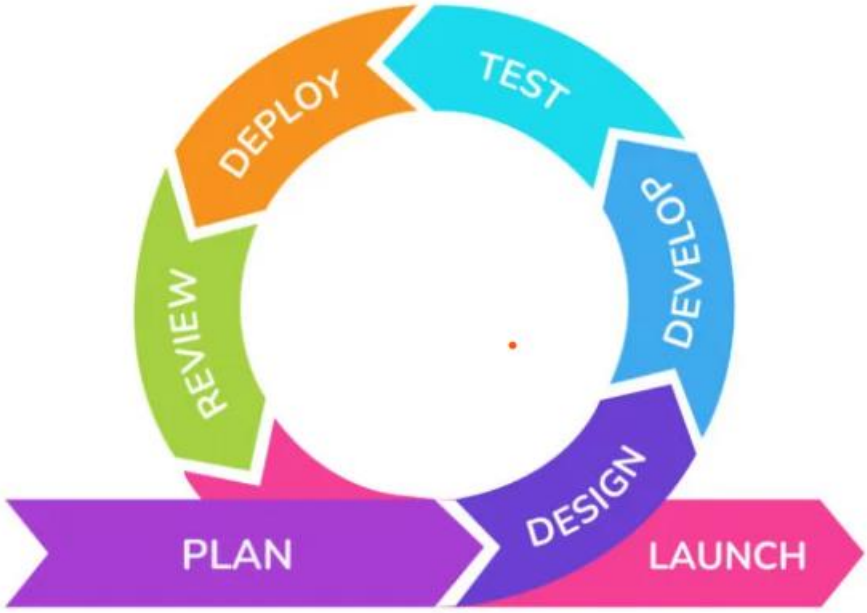


Figure 7.1: Agile Methodology

In the realm of modern IT projects, Agile approaches have become the norm, with over 70% of organizations embracing its methodologies [30]. These Agile iterations typically span a few weeks (around 2-4 weeks) and showcase a delightful twist—they deliver a fully functional version of the software.

In the Agile realm, the spotlight shines brightly on delivering working software swiftly. The emphasis shifts from drowning in detailed documentation to diving headfirst into software testing activities. It's a dance of priorities, ensuring that our codebase is robust and reliable while keeping our focus on delivering tangible results.

To foster effective collaboration and harmonious code orchestration, we have chosen GitHub as our trusted code hosting platform. GitHub's magical realm of version control empowers us to track and revert changes with ease. It becomes our virtual sanctuary, where we weave together the threads of our project, keeping everything in sync and ensuring the integrity of our codebase.

Within our GitHub sanctuary, a repository has been set up, granting all team members the power of contribution. We now have the ability to push our changes to GitHub, enabling others to stay up to date by pulling from this digital haven. This check-in and check-out system allows us to collaborate remotely, seamlessly working on the project as if we were all in the same room. And should the need arise, we possess

the mystical ability to revert to previous versions of our code, ensuring that no misstep becomes an insurmountable obstacle.

So let us continue this Agile adventure, armed with the collaborative power of GitHub. Together, we shall push the boundaries of innovation, writing our code in harmony and dancing to the rhythm of successful iterations. With GitHub as our ally, we embrace the spirit of efficiency, flexibility, and the joy of seamless code collaboration.

The functioning parts delivered by the team during the semester:

Week No.	Tasks that are Assigned to the team0
First week	Study relevant hardware decisions and choices
Second week	Learn about Google Cloud and its relevant services
Third week	Learn about Go and multi-concurrency
Fourth week	Learn about Google Cloud SDKs for object detection
Fifth week	Learn about Google Cloud SDKs for text detection
Sixth week	Learn about Google Cloud Text-to-Speech SDK
Seventh week	Creating a websockets server using go for multi-concurrency
Eighth week	Establishing a connection with our Google Cloud environment
Ninth week	Create the relevant client software using Go
Tenth week	Implementing the client and installing it on our hardware
Eleventh week	Conducting performance testing and analysis
Twelfth week	Writing the final documentation
Thirteen week	Presentation work

Table 7.1 Weekly work

7.2 Quality and Risk Management

In the context of developing smart glasses to aid visually impaired individuals with object and text detection converted into speech, our approach to quality and risk management is central to project success.

- **Quality Management:**

Ensuring the accuracy of object and text detection functionalities through rigorous testing is our priority. Continuous refinement, guided by user feedback, guarantees a user-centric experience.

- **Risk Management:**

Customized risk assessments are undertaken to address challenges unique to our project, including environmental factors affecting object detection. Focused mitigation strategies are implemented for seamless text-to-speech conversion.

In the middle of our development journey, our commitment to quality and risk management ensures that the smart glasses project remains focused on delivering a reliable and impactful solution for the visually impaired.

7.3 Lessons Learned

Throughout the course of this project, we have gained valuable skills and knowledge that will be beneficial in our future endeavors.

1. Developing the ability to read and understand research papers in order to extract relevant information and apply it in practice.

2. Establishing effective time management skills to prioritize tasks and meet deadlines.

3. Working collaboratively with a team and utilizing version control tools like Git to contribute to group projects.

4. Familiarizing oneself with the functions and documentation of external packages in order to effectively utilize them in projects.

5. Communicating effectively with team members and committing to deadlines to ensure project success.

6. Staying current with developments in the field through continuous learning and staying informed about new technologies and techniques.

These skills collectively position the team for success in future endeavors, showcasing adaptability, collaboration, and a commitment to ongoing professional development.

8 CHAPTER 8: CONCLUSION AND WAY FORWARD

8.1 Restatement of Purpose of Report and Objectives

In this section, we revisit the primary aim of our report, centered on the development of smart glasses tailored to aid individuals with visual impairments. Our objectives encompass:

- Enabling the detection and interpretation of text through advanced OCR technologies.
- Implementing sophisticated algorithms for object detection, enhancing the glasses' ability to recognize and convey information about the user's surroundings.
- Creating a seamless text-to-speech functionality, converting detected text into audible speech for real-time user awareness.

These objectives collectively drive our overarching goal of delivering a practical and impactful solution, ultimately enhancing the daily experiences and independence of those with visual impairments through our smart glasses project.

8.2 Restatement of Proposed Deliverables

This section revisits and reaffirms the anticipated deliverables outlined for our smart glasses project, emphasizing the tangible outcomes we aim to achieve. The proposed deliverables include:

- Completed Smart Glasses Prototype:

  A fully functional prototype integrating text detection, object detection, and text-to-speech capabilities.

- Comprehensive Documentation:

  Thorough documentation detailing the design, development, and implementation processes of the smart glasses, aiding in understanding and replication.

- User Interface Demo:

  A demonstration of the user interface showcasing the intuitive controls and seamless interaction mechanisms incorporated into the smart glasses.

- Performance Analysis Report:

  An in-depth report evaluating the performance metrics, including accuracy rates in text and object detection, as well as the efficiency of the text-to-speech conversion.

These proposed deliverables serve as tangible markers of our project's progress and success, aligning with our commitment to delivering a functional, well-documented, and user-friendly solution for individuals with visual impairments.

8.3 Summary of How Each Objective has been Met

This section provides a consolidated overview of the fulfillment of each project objective, offering a comprehensive account of how the initial goals have been successfully addressed. The summary highlights the methodologies, technologies, and innovations employed to meet each specific objective:

- Text Detection Objective:

Implementation of advanced Optical Character Recognition (OCR) technologies for accurate and efficient text detection.

Integration of real-time processing capabilities ensuring prompt conversion of detected text into audible content.

- Object Detection Objective:

Adoption of sophisticated object detection algorithms to enable seamless identification of objects within the user's surroundings.

Incorporation of intuitive feedback mechanisms, enhancing user interaction and situational awareness.

- Text-to-Speech Conversion Objective:

Development of a text-to-speech functionality with natural and expressive synthesis.

Customizable voice options and settings to accommodate user preferences.

This summary serves to underscore the successful realization of each project objective, emphasizing the practical outcomes achieved in creating a technologically advanced and user-centric solution for individuals with visual impairments.

8.4 New Skills Learnt

Throughout this project, we've gained a plethora of valuable skills encompassing both software and hardware domains.

On the software front, we delved into a new programming language, Go, mastering its application for concurrent programming. Our journey also involved a deep dive into media servers, exploring their applications and distinctions. Additionally, we acquired insights into websocket servers.

In the hardware realm, our focus extended to diverse microcontrollers and embedded platforms, notably including the Raspberry Pi. We honed our abilities in developing software tailored for these platforms.

In essence, this project served as an enriching experience in the realm of computer engineering, allowing us to expand our knowledge and skills significantly.

8.5 Way Forward

We are eager to continue advancing the project, as we firmly believe it carries substantial benefits for the community and has the potential to make a significant contribution.

8.6 Final Discussion and remarks

Overall, we have accomplished many of our goals and discovered a lot of insight regarding our architecture, we have tested the different services belonging to Google and learned how they could accomplish our goal of fast iteration and testing and development regarding this project, as well as discovered how client hardware interacts with our system and how to develop software for it.

Acknowledging the significance of version control and code backup, we made the deliberate choice to upload our package to GitHub [31]. This decision served dual purposes: firstly, it guaranteed the secure storage and convenient access of our code for future revisions; secondly, it aimed to establish a valuable reference for fellow individuals embarking on similar projects, enabling them to leverage and build upon our work effectively. By sharing our code on GitHub, we aimed to foster collaboration and knowledge exchange within the community, ultimately contributing to the collective advancement of such endeavors. please refer to Appendix B for the GitHub project link.

REFERENCES

[1] Optical Character Recognition: Ravina Mithe, Supriya Indalkar, Nilam Divekar

document (psu.edu)

[2] Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video

Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video - NASA/ADS (harvard.edu)

[3] simple high-performance bioinformatics toolkit for the Go language

bíogo: a simple high-performance bioinformatics toolkit for the Go language | bioRxiv

[4] Cloud Vision API

Vision AI | Google Cloud

[5] The Reality of Disability "Functional Difficulties" in

Jordan, based on the data of The General Population and Housing Census 2015

dos.gov.jo/dos_home_e/main/population/census2015/Disability 2021.pdf

[6] WebRTC technology overview and signaling solution design and implementation

WebRTC technology overview and signaling solution design and implementation | IEEE Conference Publication | IEEE Xplore

[7] P. N. Durette, "gTTS: gTTS (Google Text-to-Speech), a Python library and CLI

tool to interface with Google Translate text-to-speech API."

GitHub - pndurette/gTTS: Python library and CLI tool to interface with Google Translate's text-to-speech API

[8] Communicating and Displaying Real-Time Data with WebSocket

Communicating and Displaying Real-Time Data with WebSocket | IEEE Journals & Magazine | IEEE Xplore

[9] Deep Learning Assisted Smart Glasses as Educational

Aid for Visually Challenged Students

Deep Learning Assisted Smart Glasses as Educational Aid for Visually Challenged Students (researchgate.net)

[10] Real Time Object Detection and Recognition for Blind People

PDF المشروع.pdf (ppu.edu)

[11] Design of Smart Glasses that Enable Computer Vision for the Improvement of Autonomy of the Visually Impaired.

C. C. Spandonidis*, D. Spyropoulos, N. Galiatsatos, F.Giannopoulos and D.Karageorgiou

Microsoft Word - 15_113_118_5277_.ok.docx (jestr.org)

[12] Raspberry Pi OS (previously called Raspbian)

Raspberry Pi OS – Raspberry Pi

[13] Streamline your setup experience with our RealVNC Connect Setup app

Download VNC Viewer | VNC® Connect (realvnc.com)

[14] Text-to-Speech AI

Text-to-Speech AI: Lifelike Speech Synthesis | Google Cloud

[15] Using the Go Programming Language in Practice by Erik Westrup & Fredrik Pettersson

Using the Go Programming Language in Practice (lu.se)

[16] Model Checking CSP Revisited: Introducing a Process Analysis Toolkit

Model Checking CSP Revisited: Introducing a Process Analysis Toolkit | SpringerLink

[17] The WebSocket API (WebSockets)

The WebSocket API (WebSockets) - Web APIs | MDN (mozilla.org)

[18] Package gorilla/websocket is a fast, well-tested and widely used WebSocket implementation for Go.

GitHub - gorilla/websocket: Package gorilla/websocket is a fast, well-tested and widely used WebSocket implementation for Go.

[19] Go Cloud Client Libraries

Go Cloud Client Libraries | Google Cloud

[20] FFmpeg ImageContext Struct

FFmpeg: ImageContext Struct Reference

[21] Google Vision API Detect Labels

https://cloud.google.com/vision/docs/labels

[22] Understanding Object Localization with Deep Learning

Understanding Object Localization with Deep Learning (einfochips.com)

[23] WaveNet: A generative model for raw audio

WaveNet: A generative model for raw audio - Google DeepMind

[24] Oto (v2) A low-level library to play sound.

oto package - github.com/hajimehoshi/oto/v2 - Go Packages

[25] Video for Linux API Libv4l User Space Library

6.1. Introduction — The Linux Kernel documentation

[26] Adaptive Rate Control for Live streaming using SRT protocol

Adaptive Rate Control for Live streaming using SRT protocol | IEEE Conference Publication | IEEE Xplore

[27] OpenCV is the world's biggest computer vision library

https://opencv.org/

[28] PiCockpit | Monitor and Control your Raspberry Pi

PiStats | PiCockpit

[29] Using PiDoctor to diagnose your Raspberry Pi

PiDoctor | PiCockpit

[30] Exploring the Latest Trends in Agile Statistics for 2024

https://radixweb.com/blog/agile-statistic

[31] Check out our GitHub project repository:

https://github.com/MustafaMathhar/go_detection

APPENDICES

Appendix A: Codes Used

```
const (
  serverURL      = "ws://localhost:8080/upload"
  frameWidth     = 480
  frameHeight    = 360
  waitKeyDelayMS = 300
)

var upgrader = websocket.Upgrader{
  ReadBufferSize:  128000,
  WriteBufferSize: 128000,
}

var op = &oto.NewContextOptions{
  SampleRate:   44100,
  ChannelCount: 2,
  Format:       oto.FormatSignedInt16LE,
}
```

```
case sound, ok := <-messageChan:
  if !ok {
    fmt.Println("Message channel closed")
    return
  }
  log.Println("Received sound bytes:", len(sound))
  fbReader := bytes.NewReader(sound)
  decodedMp3, err := mp3.NewDecoder(fbReader)
  player := otoCtx.NewPlayer(decodedMp3)
  player.Play()
  for player.IsPlaying() {
    time.Sleep(time.Millisecond)
  }
  if err != nil {
    panic("mp3.NewDecoder failed: " + err.Error())
  }
}
time.Sleep(time.Millisecond * waitKeyDelayMS)
```

```
for {
  select {
  case frame, ok := <-dev.GetOutput():
    if !ok {
      fmt.Println("Frame channel closed")
      return
    }
    log.Printf("Frame is:  %d \n", len(frame))
    if err := conn.WriteMessage(websocket.BinaryMessage, frame); err != nil {
      log.Println("Error sending message:", err)
      return
    }
```

```
go func() {
  for {
    messageType, sound, err := conn.ReadMessage()
    if err != nil {
      log.Println("Error reading message:", err)
      close(messageChan)
      return
    }

    if messageType == websocket.BinaryMessage && len(sound) > 0 {
      select {
      case messageChan <- sound:
      default:
        log.Println("Message dropped: channel full")
      }
    }
  }
}()
```

```
u, _ := url.Parse(serverURL)
conn, _, err := websocket.DefaultDialer.Dial(u.String(), nil)
if err != nil {
  log.Fatal(err)
}
defer conn.Close()

// Initialize image matrix to store webcam frames

// Create a signal channel to handle interruption (Ctrl+C)
interrupt := make(chan os.Signal, 1)
signal.Notify(interrupt, os.Interrupt)

messageChan := make(chan []byte, 128000) // Channel to handle received messages
```

```go
func main() {
  // open sound
  _, readyChan, err := oto.NewContext(op)
  if err != nil {
    panic("oto.NewContext failed: " + err.Error())
  }
  <-readyChan
  // Open the webcam device
  ctx := context.Background()
  dev, err := device.Open("/dev/video0", device.WithPixFormat(
    v4l2.PixFormat{PixelFormat: v4l2.PixelFmtJPEG, Width: 480, Height: 360},
  ))
  if err != nil {
    log.Fatalf("the error: %d ", err)
  }
  defer dev.Close()
  if err := dev.Start(ctx); err != nil {
    log.Fatalf("failed to start stream: %s", err)
  }
```

```go
func displayResults(
    res string,
    tc *texttospeech.Client,
    ctx context.Context,
    //oc *oto.Context,
) []byte {

    req := createTTSRequest(res)
    resp, err := tc.SynthesizeSpeech(ctx, &req)
    if err != nil {
        log.Fatal(err)
    }
    return resp.GetAudioContent()
}
```

```go
func createTTSRequest(res string) texttospeechpb.SynthesizeSpeechRequest {

  return texttospeechpb.SynthesizeSpeechRequest{
    Input: &texttospeechpb.SynthesisInput{
      InputSource: &texttospeechpb.SynthesisInput_Text{Text: res},
    },
    Voice: &texttospeechpb.VoiceSelectionParams{
      Name:         "en-US-Wavenet-A",
      LanguageCode: "en-US",
      //SsmlGender:   texttospeechpb.,
    },
    AudioConfig: &texttospeechpb.AudioConfig{
      AudioEncoding:    texttospeechpb.AudioEncoding_MP3,
      EffectsProfileId: []string{"headphone-class-device"},
      SpeakingRate:     0.6,
      Pitch:            0.5,
    },
  }

}
```

```go
func detectText(

  img *visionpb.Image,
  imageContext *visionpb.ImageContext,
  vc *vision.ImageAnnotatorClient,
  ctx context.Context,
) string {

  res, err := vc.DetectTexts(ctx, img, imageContext, 1)
  if err != nil {
    log.Fatalf("Error sending requests: %v", err)
  }

  if len(res) <= 0 {
    return ""
  }
  text := res[0].GetDescription()
  if len(text) <= 0 {
    return ""
  }

  return text
}
```

```go
func detectObjects(
  ctx context.Context,
  img *visionpb.Image,
  imageContext *visionpb.ImageContext,
  vc *vision.ImageAnnotatorClient,
) (string, error) {
  res, err := vc.LocalizeObjects(ctx, img, imageContext)
  if err != nil {
    log.Fatalf("Error sending requests: %v", err)
  }
  if len(res) <= 0 {
    return "", errors.New("response is 0")

  }
  var finalTxt string
  for _, annotation := range res {
    if annotation.GetScore() >= 0.60 {

      finalTxt = finalTxt + annotation.GetName() + " "
    }
    log.Println(annotation.GetName(),annotation.GetScore())
  }
  fmt.Println()
  return finalTxt, nil
}
```

```go
var imageContext = visionpb.ImageContext{
  LanguageHints: []string{"en", "ar"},
}

func buildImage(image []byte) visionpb.Image {
  return visionpb.Image{
    Content: image,
  }

}
```

```go
  readChan := make(chan []byte, 1)
  conn, err := upgrader.Upgrade(w, r, nil)
  if err != nil {
    log.Fatal(err)
    return
  }
  defer conn.Close()
```

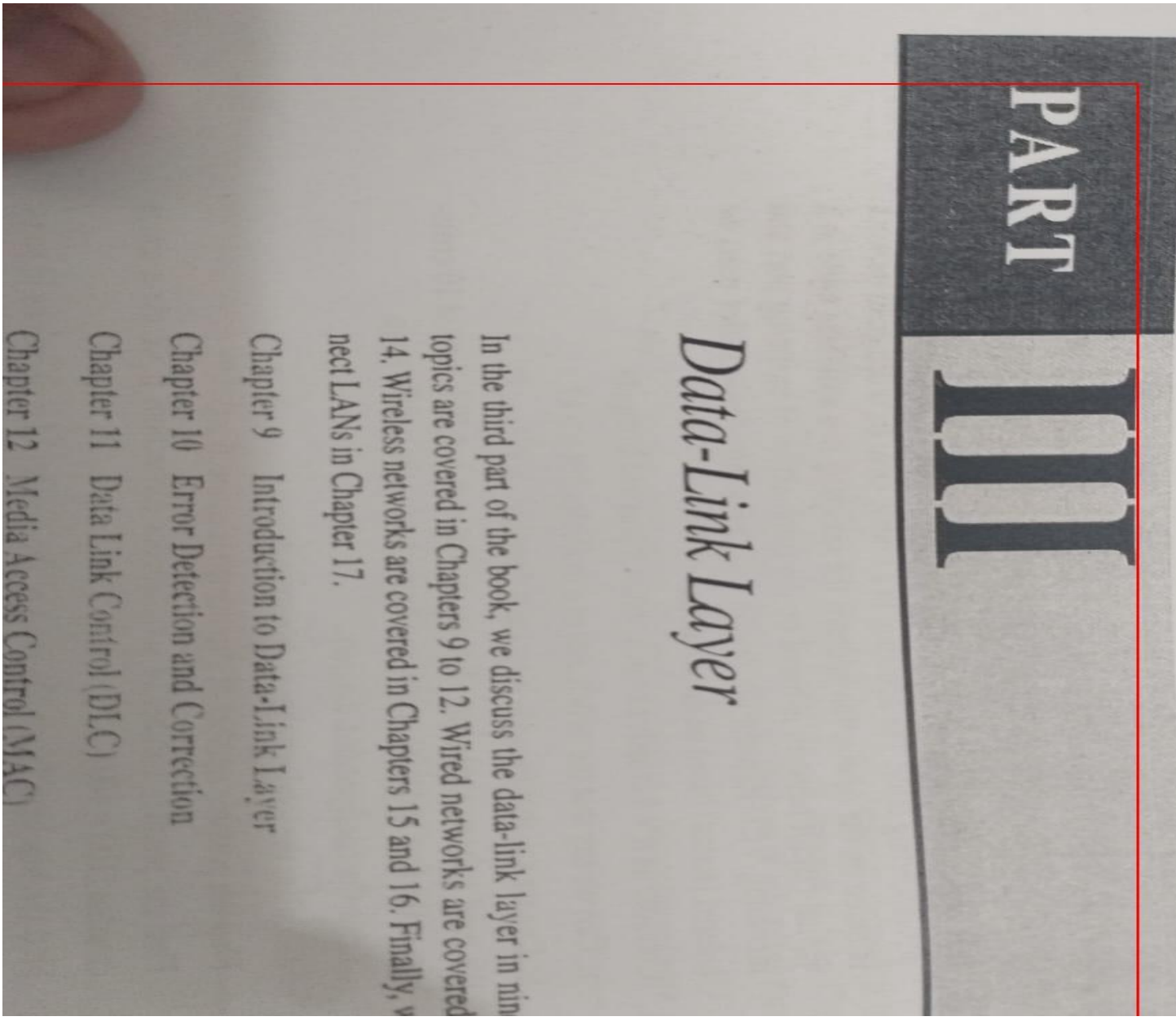
```
    for {
        // Read message from the WebSocket client
        messageType, img, err := conn.ReadMessage()
        if err != nil {
            log.Println("Error reading message:", err)
            break
        }

        if messageType == websocket.BinaryMessage {
            readChan <- img
        }
    }
}
```

```
func main() {
    ctx := context.Background()

    visionClient, err := vision.NewImageAnnotatorClient(ctx, CREDENTIALS)
    if err != nil {
        log.Fatalf("Failed to create vision client: %v", err)
    }
    defer visionClient.Close()

    tc, err := texttospeech.NewClient(
        ctx,
        CREDENTIALS,
    )
    if err != nil {
        log.Fatal(err)
    }
    defer tc.Close()

    http.HandleFunc("/upload", func(w http.ResponseWriter, r *http.Request) {
        handleVideoUpload(w, r, visionClient, tc, ctx)

    })

    fmt.Println("WebSocket server is running on port 8080")
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Appendix B: GitHub Project

Link for our GitHub project repository :

https://github.com/MustafaMathhar/go_detection



PART III

Data-Link Layer

In the third part of the book, we discuss the data-link layer in nin
topics are covered in Chapters 9 to 12. Wired networks are covered
14. Wireless networks are covered in Chapters 15 and 16. Finally, w
nect LANs in Chapter 17.

Chapter 9 Introduction to Data-Link Layer

Chapter 10 Error Detection and Correction

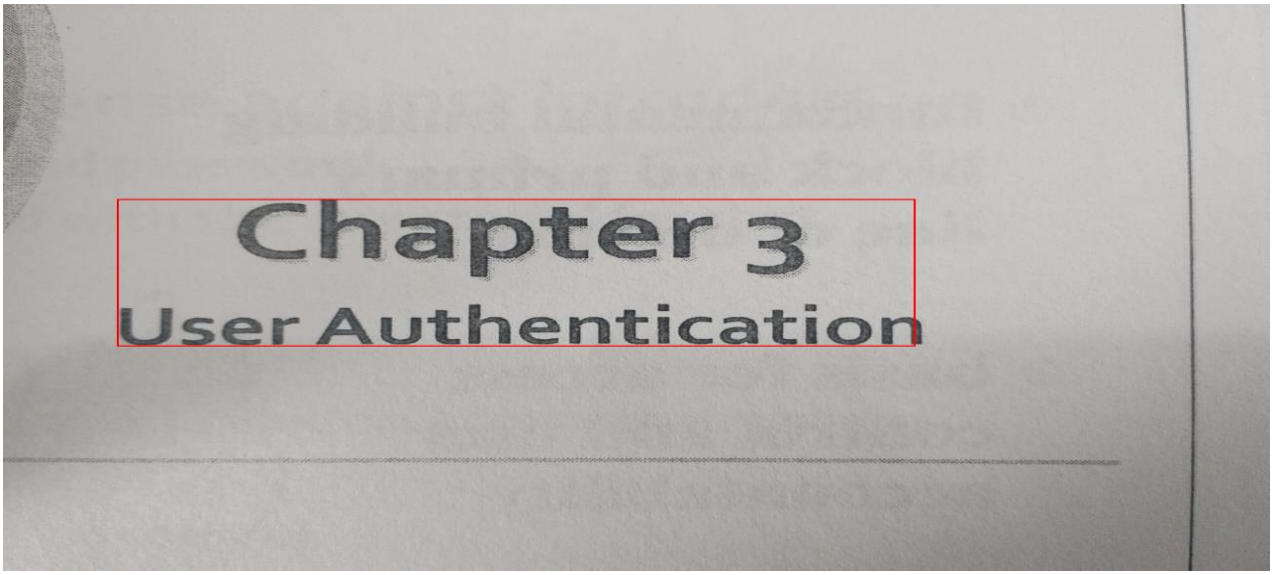
Chapter 11 Data Link Control (DLC)

Chapter 12 Media Access Control (MAC)

The approach suggested in Figure 14.3 has many variations, one of which is described in this subsection. The scheme (Figure 14.4) is useful for providing end-to-end encryption at a network or transport level in a way that is transparent to the end users. The approach assumes that communication makes use of a connection-oriented end-to-end protocol, such as TCP. The noteworthy element of this approach is a session security module (SSM), which may consist of functionality at one protocol layer, that performs end-to-end encryption and obtains session keys on behalf of its host or terminal.

The steps involved in establishing a connection are shown in Figure 14.4. When one host wishes to set up a connection to another host, it transmits a connection-request packet (step 1). The SSM saves that packet and applies to the KDC for permission to establish the connection (step 2). The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC. If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate SSMs, using a unique permanent key for each SSM (step 3). The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems (step 4). All user data exchanged between the two end systems are encrypted by their respective SSMs using the one-time session key.

The automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other.



يذكرني هذا المشهد في كل مرة بفيلم أمريكي قديم بعنوان " المبارزة " هو واحد من الافلام النادرة التي عكست بقوة بشاعة هذه التقاليد الإنسانية اللاإنسانية .

تجرى احداث الفيلم في بلدة صغيرة يقيم فيها رجل اشتهر فيما مضى من الزمن بانه الابرع والاسرع في استخدام المسدس . وهو الآن يعيش حياة ، بسيطة بعد ان " تقاعد" ويعمل في احد المتاجر ويسعى للاستقرار وضمان مستقبله وشيخوخته . وفي يوم من الايام ، يصل إلى البلدة عابر طريق ، يحمل حالياً شهرة الأبرع والأسرع في إستخدام المسدس . وهو في طريقه إلى مكان مابحثا عن عمل مستقر وحياة هادئة . ونتيجة لقدومه إلى البلدة يدور الهمس بين سكانها وتنتشر الشائعات التي تؤكد ان البطل القادم جاء ليباري البطل المتقاعد المقيم وليتحداه على اللقب ويعلو صوت الشائعات إلى درجة ان الجميع يصدقونها ، فتقام المراهنات وتتشكل لجنة تباشر استعدادات لاقامة مباراة تاريخية بين بطلين من نجوم الرمي بالمسدس . ويصبح الهوس عاما إلى درجة ان البطلين يوضعان في موقف لا يحسدان عليه ، فينظران للرضوخ لمطالب جماهير البلدة . وهكذا يتم تنظيم المباراة في حلبة ضخمة شيدت اصلا لاقامة احتفالات مصارعة الثيران التقليدية فيها .

وفي ساعة المباراة يتجمع آلاف المواطنين في مدرجات

KNN

- The K-nearest neighbors (KNN) algorithm is a type of supervised machine learning algorithms.
- KNN is extremely easy to implement in its most basic form, and yet performs quite complex classification tasks.
- It is a lazy learning algorithm since it doesn't have a specialized training phase. Rather, it uses all of the data for training while classifying a new data point or instance.

(تشارلز فوران، ذي غلوب اند مايل)

بينما يستطيع أي كاتب أن يخبر قصة تشبه الحلم، وحده الفنان النادر، مثل موراكامي، يجعلنا نشعر أننا نحلم هذه القصة بأنفسنا.

(لورا ميلر، ذي نيويورك تايمز بوك ريفيو)

كعادته يدخلنا موراكامي في أجواء غرائبية، وبقدر ما هي غرائبية فإنها بسيطة تحتفل بسحر الحياة وتدافع عنها، وذلك من خلال حكايتين متوازيتين متقاطعتين، حكاية عجوز يبحث عن نصف ظله الضائع، وفتى في الخامسة عشرة هارب من لعنة أبيه السوداء، وبينهما عوالم ومدن وشخصيات ورحلات شبه ملحمية تدور جميعها حول البحث عن الحب، ومعنى الموت، وقيمة الذكريات. رواية تدفع كل واحد منا إلى تأمّل الحياة، وبدء رحلة البحث عن بوصلته الضائعة.

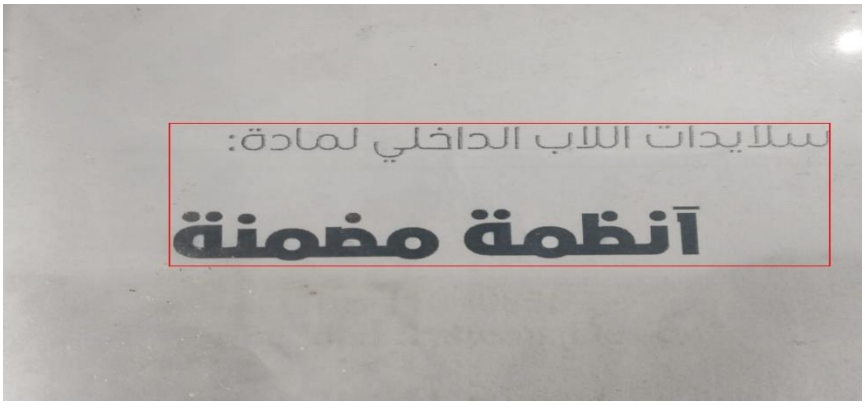
instructions. In the token-passing method, the stations in a network are organized in a logical ring. Each station has a predecessor and a successor. A special packet called a *token* circulates through the ring.

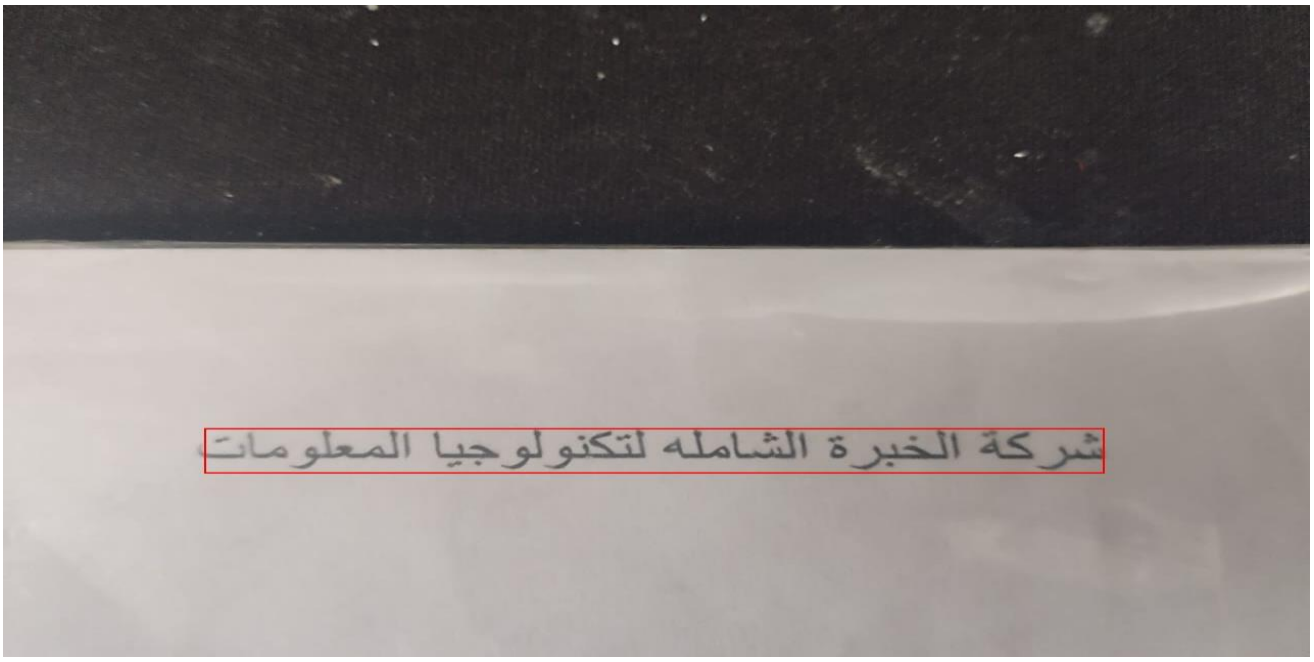
Channelization is a multiple-access method in which the available bandwidth of a link is shared in time, frequency, or through code, between different stations. We discussed three channelization protocols: FDMA, TDMA, and CDMA. In frequency-division multiple access (FDMA), the available bandwidth is divided into frequency bands. Each station is allocated a band to send its data. In other words, each band is reserved for a specific station, and it belongs to the station all the time. In time-division multiple access (TDMA), the stations share the bandwidth of the channel in time. Each station is allocated a time slot during which it can send data. Each station transmits its data in its assigned time slot. In code-division multiple access (CDMA), the stations use different codes to achieve multiple access. CDMA is based on coding theory and uses sequences of numbers called *chips*. The sequences are generated using orthogonal codes such as the Walsh tables.

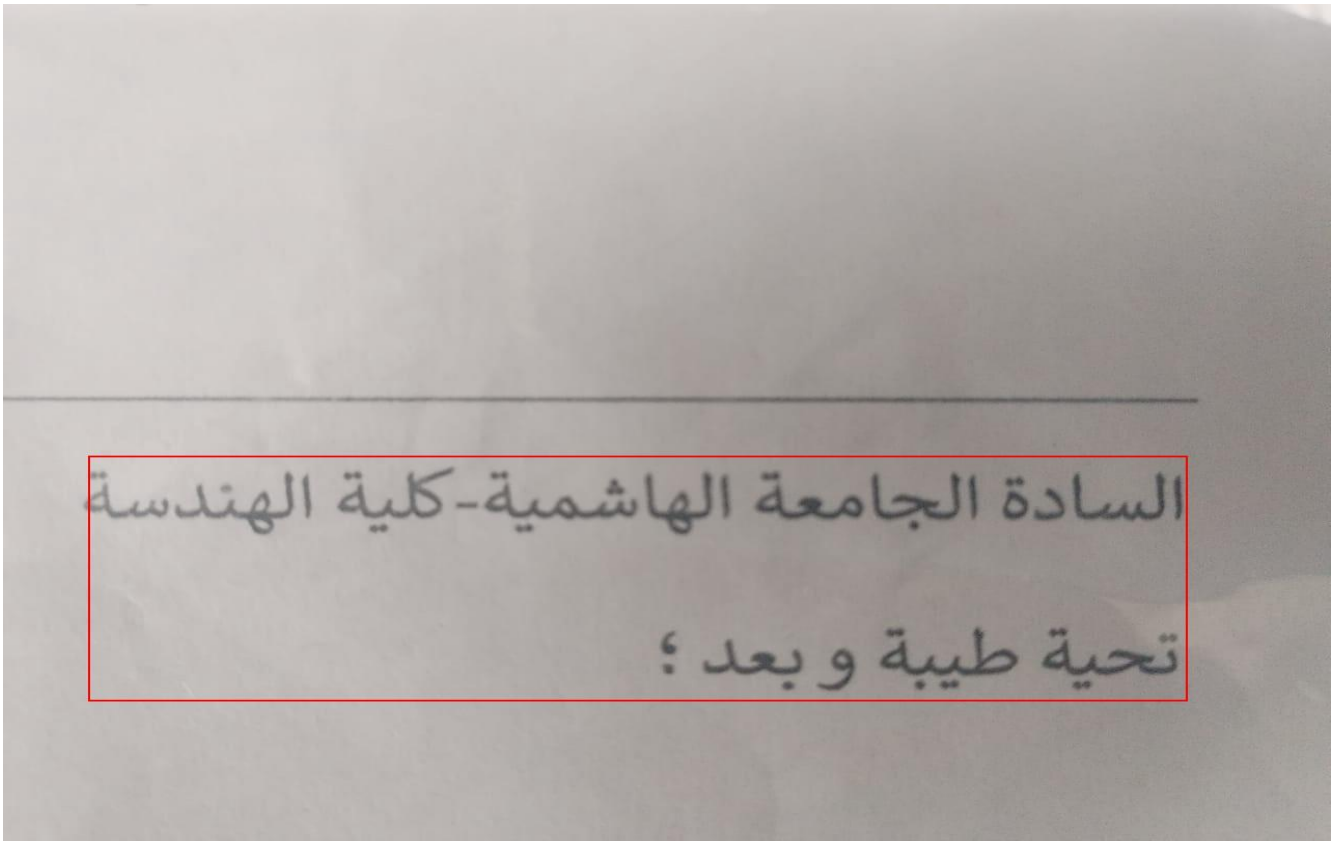
12.5 PRACTICE SET

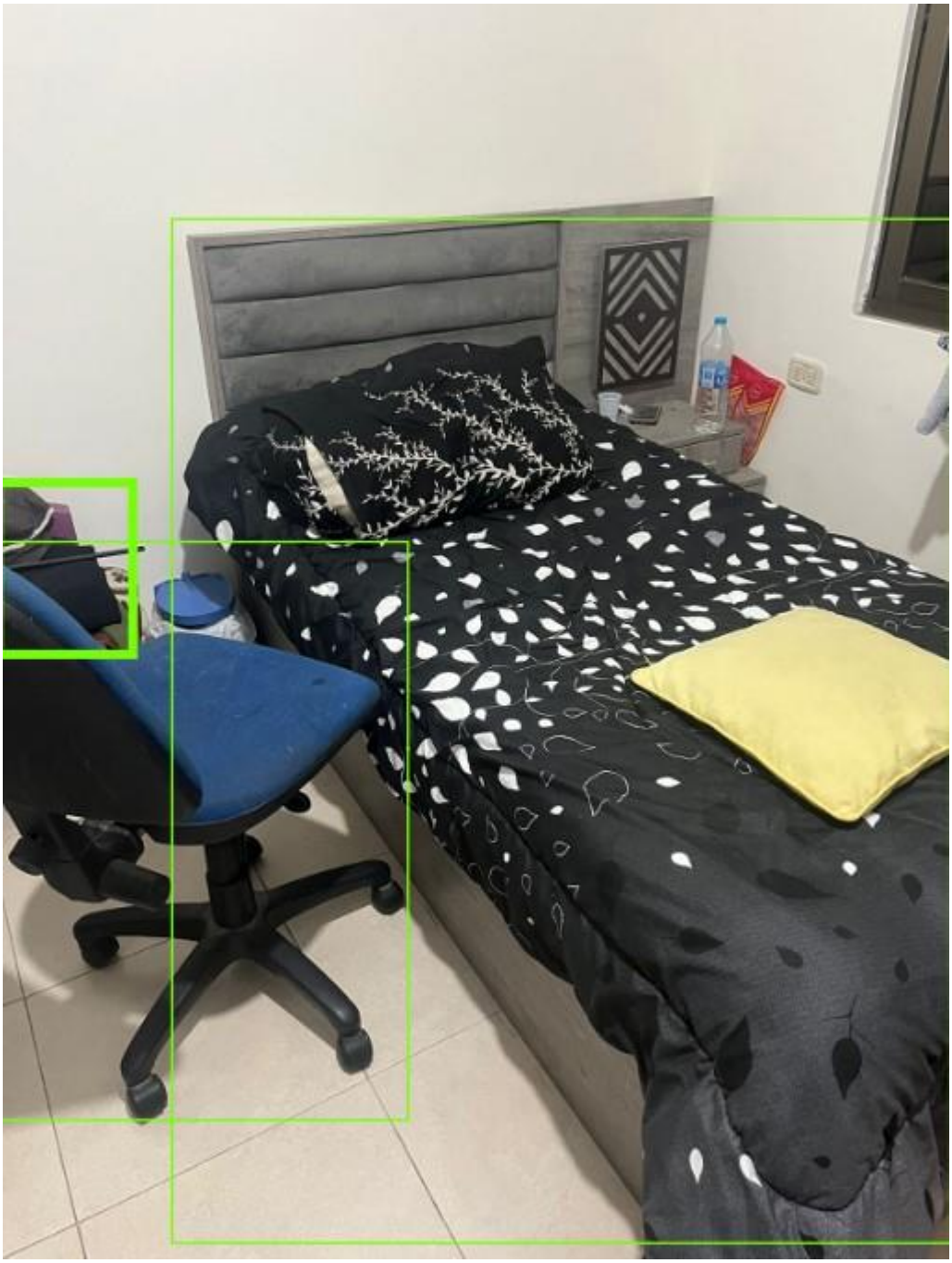
12.5.1 Quizzes

A set of interactive quizzes for this chapter can be found on the book website. It is strongly recommended that the student take the




شركة الخبرة الشامله لتكنولوجيا المعلومات


السادة الجامعة الهاشمية-كلية الهندسة
تحية طيبة و بعد ؛



Appendix D: