UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

A progress report submitted for the award of
MEng Electrical and Electronics Engineering

Supervisor: Dr. Tom Blount
Examiner: TBD

Open-Source Stereo Video Camera System and Virtual Reality (VR) Software Implementation for Stereoscopic Life-logging and Content Creation

by Muhammad Hazimi Bin Yusri

April 30, 2024

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for the award of MEng Electrical and Electronics Engineering

by Muhammad Hazimi Bin Yusri

This project addresses challenges related to the accessibility and cost-effectiveness of stereoscopic video recording for Virtual Reality (VR) Head Mounted Display (HMD) technologies. The main goal is to democratize VR content creation by developing an open-source, modular stereo video camera system, implementing an efficient video processing pipeline, and creating an intuitive VR software tailored for exploring stereoscopic media with focus on life-logging. At its core, the project transforms stereoscopic life-logging into a streamlined process, while taking advantage of bleeding-edge open-source technologies. The system will incorporates accurate scene detection algorithms, allowing custom metadata tagging within captured videos and images. This integration enhances the life-logging experience, allowing for personalised categorisation and efficient retrieval of specific moments, or scenes. Ultimately, the project aims to provide a practical and accessible foundation for stereoscopic life-logging, empowering users to curate and revisit their experiences immersively in VR.

Contents

Acknowledgements vii

1 Introduction 1
- 1.1 Problem Statement . . . 1
- 1.2 Objectives . . . 1
- 1.3 Overview . . . 2

2 Background 4
- 2.1 Literature Review . . . 4
  - 2.1.1 Life-logging . . . 4
  - 2.1.2 Stereo Camera . . . 5
  - 2.1.3 Virtual Reality (VR) . . . 6
  - 2.1.4 Stereo Content Formats . . . 7
  - 2.1.5 User Interface in VR . . . 7
- 2.2 Conceptual Framework . . . 8
- 2.3 Context . . . 9

3 Design 10
- 3.1 Requirements Analysis . . . 10
  - 3.1.1 Functional Requirements (FR) . . . 11
  - 3.1.2 Non-Functional Requirements (NFR) . . . 12
- 3.2 Finalised Design . . . 12
  - 3.2.1 Hardware . . . 12
  - 3.2.2 Stereo Processing Pipeline . . . 14
  - 3.2.3 VR Software Development . . . 14

4 Implementation 16
- 4.1 Development Process . . . 16
- 4.2 Component Details . . . 17
  - 4.2.1 Processing Unit (Raspberry Pi 5 and Pico) . . . 17
  - 4.2.2 Cameras . . . 18
    - 4.2.2.1 ArduCAM 5MP OV5642 . . . 18
    - 4.2.2.2 USB Webcams (Microsoft LifeCAM HD-3000) . . . 18
    - 4.2.2.3 Raspberry Pi Camera Module 3 (Wide) . . . 19
- 4.3 Hardware Development . . . 20

4.3.1 Custom Enclosure and POV Considerations 20
4.3.2 Onboard Life-logging Algorithm 21
4.4 Stereo Processing Pipeline 22
4.4.1 FileZilla (FTP via SSH) 22
4.4.2 Python Scripts and FFMPEG 23
4.4.3 MAX-Scene Classifier (Scene Metadata .json Generation) . . 25
4.5 VR Software 25
4.5.1 Godot Game Engine 25
4.5.2 SBS Video Projection 26
4.5.3 File Browsing 27
4.5.4 2D UI and VR Port 29

5 Testing and Evaluation 32
5.1 Testing Strategy 32
5.2 Test Results 33
5.2.1 Functional Requirements (FR) 33
5.2.2 Non-Functional Requirements (NFR) 35
5.3 Performance Evaluation 36

6 Project Management 39
6.1 Gantt Charts 39
6.2 Reflections 40

7 Conclusion 46
7.1 Summary 46
7.2 Future Work 47
7.3 Finishing thoughts 48

Bibliography 49

Risk Assessment 53

Onboard Algorithm Code 56

Stereo Processing Pipeline Codes 58

Shaders Code 64

Design Archive Files 65

Project Brief 67

List of Figures

2.1 Evolution of wearable camera technology. From left to right: Mann (1998), GoPro (2002), SenseCam (2005), Narrative Clip (2013), reproduced from [11] . . . 4
2.2 Commercially available consumer-level stereo cameras. From left to right: Snapchat Spectacles 3[3], Kandao Qoocam Ego [4], iPhone 15 Pro/Pro Max Spatial Video[18] . . . 5
2.3 Example of VR HMDs. Meta Quest 3[1] and Apple Vision Pro[2] . 6
2.4 Apple's visionOS innovative gesture and eye tracking UI navigation, making full use of 3D space for windows elements with variable sizes.[2] 8

3.1 Final design front and back view. . . . 13
3.2 Godot Game Engine UI example, specific software features can be seen in Implementation section. . . . 14

4.1 Raspberry Pi 5 and Pico . . . 18
4.2 ArduCAM 5MP OV5642 and Microsoft LifeCAM HD-3000 . . . 19
4.3 Rasberry Pi Camera Module 3 (Standard lens) and Rasberry Pi Camera Module 3 (Wide lens) . . . 19
4.4 Onshape public sharing link for USB Webcam version and Raspberry Pi Camera Module 3 version . . . 20
4.5 Designs and POV Example. Left to right: Glasses, Head-Mounted and Chest-Mounted . . . 21
4.6 Examples of Mono (left) and Stereo (right) images that are stitched from two mono stills . . . 24
4.7 Example of image and the prediction api .json response. . . . 26
4.8 Images of App Running . . . 27
4.9 The simple main menu and file explorer (with thumbnails) 2D app screenshots . . . 28
4.10 Example filter with prediction of 'supermarket' and 'residential neighbourhood' . . . 29
4.11 Example of folder directories and search result of '1513' ie 3:13PM. 29
4.12 Additional buttons added to aid user browsing experience in image and video viewing screen. . . . 30
4.13 Example of bigger UI buttons and option to select by clicking on thumbnails including visual feedback. . . . 30

4.14 To change from smooth locomotion, simply drag and replace LeftMovementDirect with function_teleport. To change from smooth turning to snap turning, just simply choose it from the dropdown. . 31

6.1 Planned Gantt Chart. Legend: Orange - Report or writing related task, Blue - Software related task, Green - Hardware related task. . 41

6.2 Combined Gantt Chart before Progress Report submission. Legend: Orange - Report or writing related task, Blue - Software related task, Green - Hardware related task. Softer color represents tasks over expected time or unexpected tasks compared to the Planned Gantt chart. 42

6.3 Combined Gantt Chart after Progress Report submission. Legend: Orange - Report or writing related task, Blue - Software related task, Green - Hardware related task. Softer color represents tasks over expected time or unexpected tasks compared to the Planned Gantt chart. 43

List of Tables

3.1 Functional Requirements 11
3.2 Non-Functional Requirements 12

4.1 Cost Breakdown of Components 17

5.1 Comparison between USB-Webcam, MIPI CSI (Raspberry Pi Camera Module 3 Wide) versions and Kandao QooCam Ego 3D Camera 37

1 Risk Management Table (Part 1) 54
2 Risk Management Table (Part 2) 55

Acknowledgements

I would like to thanks Dr. Tom Blount for his tremendous support, ideas and approval for this project, Dr. Thomas Andritsch for his critical feedback and also to Godot XR community especially Bastiaan Olij and Malcolm Nixon for help with shaders, stereo video example and guidance.

Chapter 1

Introduction

1.1 Problem Statement

The surge in Virtual Reality (VR) Head Mounted Display (HMD) technologies has opened new creative avenues, offering users immersive experiences [1], [2]. However, the challenges of exclusivity and high costs in existing solutions for recording stereoscopic video limit their potential [3]–[5]. This holds significance for VR's broader adoption, as low user retention is often linked to a lack of exclusive content maximizing the medium's advantages [6]. With the hassles of wearing VR HMDs compared to watching television or using mobile phones, there needs to be a more compelling reason to use VR HMDs for media consumption, thus underlines the need to empower users to effortlessly create their stereoscopic content, enhancing the appeal of VR [7]. Most VR contents that focus only on factors like Field of View (FOV) found on 180 or 360 degree media often results in visually uninteresting videos lacking depth and resolution [7], [8]. The exclusive depth perception feature on VR HMDs and certain autostereoscopic displays has the potential to revive life-logging, offering individuals an immersive experience to relive memories [9].

1.2 Objectives

This project endeavours to surmount these challenges by:

1. Developing an open-source, cost-efficient stereo video camera hardware system to record stereoscopic content with focus on life-logging.

2. Implementing a stereo processing pipeline to automatically transform captured content to correct format and further processing.

3. Creating intuitive VR software for browsing and viewing captured stereoscopic content.

This project not only addresses the critical gaps in hardware accessibility and the video processing pipeline but also represents a significant enhancement to existing stereoscopic video VR software. By fostering an open-source, modular, and cost-effective approach, this initiative strives to democratize VR content creation, making it more widely accessible and fostering innovation in the field.

1.3 Overview

1. **Introduction**: The introduction chapter sets the stage for the report by outlining the problem statement, project objectives, and providing a concise overview of the project's scope and significance.

2. **Background**: The background chapter delves into relevant literature reviews, exploring life-logging, stereo camera technology, virtual reality (VR), stereo content formats, and user interface considerations in VR. It also presents the conceptual framework and contextual information essential for understanding the project's foundation.

3. **Design**: In the design chapter, the focus is on requirements analysis, detailing both functional and non-functional requirements. The finalised design decisions are discussed, covering hardware specifications, the stereo processing pipeline, and the development of VR software.

4. **Implementation**: Here, the implementation chapter delves into the details of each part of the project, discussing how the design decisions were put into practice. It covers the development process, component details, hardware development, stereo processing pipeline, and VR software. This chapter provides a comprehensive look at the technical aspects of bringing the project to life.

5. **Testing and Evaluation**: This chapter discusses the testing strategy employed, presents test results for both functional and non-functional requirements, and evaluates the performance of the project.

6. **Project Management**: The project management section discusses the use of Gantt charts and reflects on the process, highlighting both successes and areas for improvement.

7. **Conclusion**: The conclusion chapter summarises the key findings, achievements, and contributions of the project. It also outlines potential future work and provides finishing thoughts on the project's implications and significance.

Chapter 2

Background

2.1 Literature Review

2.1.1 Life-logging

Life-logging, a contemporary term encapsulating habitual documentation of one life akin to social media practices, distinguishes itself through its methodical and routine nature. The motivation for lifelogging extends beyond the sporadic capturing of moments to a deliberate effort to systematically record and preserve lifetime memories [10]. From the definition itself, the Life-log data can be in any format such as texts from diaries, health sensor data or even digital footprint from application use. However, in this project, I will be focusing on visual life-logging which consists of images and videos [11].

FIGURE 2.1: Evolution of wearable camera technology. From left to right: Mann (1998), GoPro (2002), SenseCam (2005), Narrative Clip (2013), reproduced from [11]

Examples of existing wearable cameras specifics that are used for visual life-logging are shown in Fig 2.1. However, in recent years with rapid advancement in the miniaturization of consumer products such as smartphones, camera sensors and

storage devices, companies like Meta [12] and Snapchat also released their versions of smart glasses, similar to Google Glass [13] released years prior. Although the main appeal is the smart assistance features for convenience, this form factor and placement of the camera is perfect for visual life-logging aimed for VR user-base especially as already proven by Snapchat Spectacles 3 [3] which can capture 3D photos and videos at 60fps.

2.1.2 Stereo Camera

Stereo camera refers to a camera with stereoscopic imaging capabilities, this is simply achieved using 2 cameras aligned side-by-side at a distance approximately the same as average human inter-pupillary distance (IPD) to mimic human vision [14], [15]. This gives the content viewed proper depth definition as in real life. This idea can also be extrapolated to higher Field Of View (FOV) content such as 180/360 degrees content but requires either a more complicated setup, computation or both [16], [17]. A higher resolution is also needed as the content is stretched into a larger area [7], this in turn increase storage and processing requirement.

FIGURE 2.2: Commercially available consumer-level stereo cameras. From left to right: Snapchat Spectacles 3[3], Kandao Qoocam Ego [4], iPhone 15 Pro/Pro Max Spatial Video[18]

Examples of consumer-level stereo cameras commercially available to buy right now as shown in Fig 2.2 are Snapchat Spectacles 3 [3], Kandao QooCam Ego [4] which includes an integrated stereo/VR viewer and technically, iPhone 15 Pro/Pro Max [5] with version iOS 17.2 and above [18] which had Spatial Video recording enabled. This move is obviously to entice consumers using Apple products to buy their upcoming Apple Vision Pro VR HMD [2] which will make stereo camera and content more mainstream.

2.1.3 Virtual Reality (VR)

Virtual Reality (VR) transcends the conventional by offering users a virtual 3D environment, or an immersive emulation of real life overlaid with digital objects which usually referred to as Mixed Reality (MR). The emotional attachment fostered by VR content stems from its heightened realism thanks to proper 3D depth perception, creating a sense of physical presence within the virtual space [7]. This emotional resonance distinguishes VR content from traditional media, providing a compelling reason for its adoption in content consumption including lifelog data.

FIGURE 2.3: Example of VR HMDs. Meta Quest 3[1] and Apple Vision Pro[2]

Contrary to fully immersive Free Viewpoint Video (FVV) [19] or scene reconstruction [20], the adoption of the SBS format is grounded in the current limitations of VR Head-Mounted Display (HMD) hardware. While FVV offers unparalleled immersion and depth perception through 6DoF [21], practical constraints dictate a compromise. This decision is further supported by the spatial video capabilities of the iPhone 15 Pro, which, despite being 1080p at 60fps, aligns with the current standards for windowed style viewing. The emphasis here is not on resolution but on perceived Pixel Per Degree (PPD) which is higher as the virtual window/screen is further, and lower FOV needed to stretch out across [7]. Choosing windowed viewing over panoramic alternatives (180/360 degrees) acknowledges the balance between realism and current technological constraints.

A poignant illustration of this concept is evident in the Black Mirror episode which explores the immersive preservation of memories in an eye-camera format [22]. This format aligns with human visual perception, eliminating the necessity for constant 3 Degrees of Freedom (3DoF) as the human gaze is not constantly omnidirectional. Although a higher FOV would enhance the life-logging experience, it remains cost-prohibitive at present (full human FOV is approximately 200 to 220 degrees [23]).

2.1.4 Stereo Content Formats

In the realm of stereo content, various formats exist, each offering its unique set of advantages and limitations. One such format is the Full Side-by-Side (Full-SBS)format, which has been selected for implementation within this project due to its simplicity and practicality. Full-SBS involves the straightforward process of stitching together two stereo pairs, making it relatively easy to generate and process stereo content. This simplicity of implementation aligns well with the project's constraints and technical requirements, allowing for efficient development and utilization within the targeted VR framework.

However, it's essential to acknowledge a drawback associated with Full-SBS: on non-VR devices, Full-SBS content appears as a stereo pair. This limitation may affect the compatibility and usability of FSBS content outside of VR environments, potentially restricting its accessibility to a broader audience. Despite this drawback, Full-SBS remains a suitable choice for the project's objectives, offering a balance between implementation simplicity and functionality within the targeted VR framework.

While other stereo content formats were considered, they were ultimately deemed less suitable for the project's scope and limitations. For example, the Multi-Picture Object (MPO) format offers better compatibility with non-VR platforms but is more complex to implement, exceeding the project's current scope. Therefore, FSBS stands out as the most practical and effective choice for fulfilling the project's stereo content requirements.

2.1.5 User Interface in VR

Integrating VR software for content browsing amplifies the life-logging experience. Leveraging the full potential of 6DoF through innovative UI design, expanded screen space [24], and interactive features such as timeline shelves, this approach redefines how users engage with their memories. The incorporation of hand tracking and eye tracking further enhances the intuitive and immersive nature of content navigation within the VR environment as shown in visionOS demo of Apple Vision Pro in Figure 2.4.

Other than that, the existence of a window or virtual screen to host the content will cause less motion sickness as users have virtual anchors to the virtual space

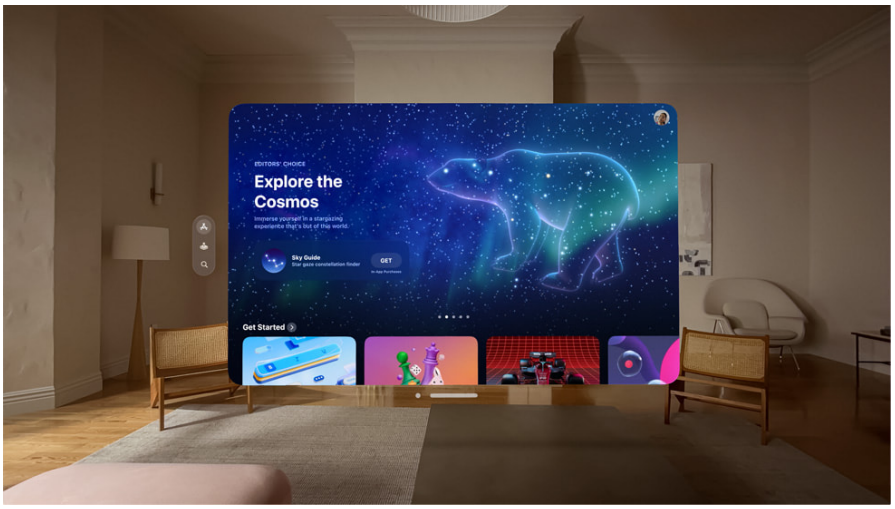

FIGURE 2.4: Apple's visionOS innovative gesture and eye tracking UI navigation, making full use of 3D space for windows elements with variable sizes.[2]

[25], [26]. Lower FOV helps as the experience will be just like watching movies in 3D cinema[27]–[29].

The combination of life-logging and VR represents a novel and niche subset within both domains. As technology advances and user preferences evolve, this integrated approach is poised to become more mainstream, particularly in the realm of personal videos, memories, and photos. The synergy between life-logging and VR anticipates a future where individuals seamlessly capture, relive, and share their most cherished moments in a more immersive and engaging manner.

2.2 Conceptual Framework

The project sits within a conceptual framework that addresses two key areas: the prohibitive nature of VR stereo content creation and the overlooked potential of life-logging in immersive format. Firstly, the creation of VR stereo content has been hindered by the high cost and complexity of hardware, leading to stagnation in research and development. This limitation has resulted in a lack of accessible tools and streamlined pipelines for creating stereo content, particularly in formats suitable for consumer adoption.

Secondly, life-logging, once a burgeoning field of research, has been overshadowed by the widespread adoption of social media platforms and smartphones for documenting daily experiences. However, there remains untapped potential for

life-logging in the context of VR, especially in windowed stereo formats that offer a simpler and more approachable experience for consumers. Despite existing studies on life-logging in VR, such as in 360-degree formats, there is a gap in development focusing on more consumer-friendly stereo content formats such as windowed Full-SBS stereo content.

Therefore, the project aims to address these gaps by developing a full streamlined pipeline for creating windowed stereo content, with a particular focus on accessibility and open-source principles. By providing accessible tools and resources, the project aims to empower individuals to engage in life-logging activities using VR technology, fostering innovation and exploration in this field.

2.3 Context

In recent years, there has been a growing demand for stereo content creation, fueled by advancements in technology and changing consumer preferences. Products like the Apple Vision Pro and iPhone 15 Pro have introduced new spatial video features, highlighting the increasing interest in immersive content experiences. Additionally, the proliferation of affordable VR headsets, such as the Meta Quest 3 (£480) and Quest 2 (£200), has made VR technology more accessible to the average consumer.

The capabilities of VR headsets have also improved significantly, with many now equipped with standalone modes and built-in cameras for pass-through functionality. This technological advancement opens up new possibilities for hands-free content creation, extending beyond traditional VR HMDs to devices like Meta Ray-Ban (mono camera) and Snapchat (stereo camera) glasses. With the ability to capture moments in stereo, users can experience more immersive and emotionally gripping content, enhancing their overall media consumption experience.

However, alongside these advancements, there are also concerns regarding privacy and data sensitivity. As life-logging becomes more prevalent, there is a need to address these concerns and ensure that users have control over their personal data. By developing open-source tools and promoting transparency in data handling, the project aims to mitigate these concerns and promote responsible life-logging practices within the VR community.

Chapter 3

Design

3.1 Requirements Analysis

The process of gathering requirements is carried out through the User Stories format, involving the following stakeholders: **VR Users**, **Stereo Content Producers**, **Developer** and **Researchers** specializing in the fields of life-logging and VR:

- **User Story 1:** As VR life-logger **developers**, we need access to an open-source stereo video camera hardware system built using off-the-shelf components. It should be cost-effective and readily available, providing a reliable solution for capturing stereoscopic content.
- **User Story 2:** As **stereo content producers**, we require an automated stereo processing pipeline tool for stitching two mono stills and videos into stereoscopic Side-By-Side (SBS) format for seamless sharing. The image and video quality should be at least in HD (720p) at 30 fps minimum to avoid blurry and nauseating footage.
- **User Story 3:** As **VR users**, we seek an intuitive VR software application for easy file browsing and content viewing. It should leverage the VR medium, allowing us to relive and interact with our stereoscopic memories effortlessly.
- **User Story 4:** As VR life-logger **researchers**, we require access to Free and Open-Source Software (FOSS) processing tools capable of automatically tagging scenes and objects within captured stereoscopic images and

videos. These tools should facilitate efficient organization and categorization of large datasets, enabling us to conduct in-depth analysis and research on life-logging experiences in virtual reality environments.

These requirements are then rewritten in the form of functional and non-functional requirements grouped in 3 separate category (**Hardware Development**, **Stereo Processing**, and **VR Software** for better planning and evaluation at the end. MoSCoW type are M (Must have), S (Should have), C (Could have) and W (Wont have).

3.1.1 Functional Requirements (FR)

Functional requirements specify what a system should do. They describe the specific functions or features that the system must provide to fulfill its intended purpose. These requirements are usually expressed as actions or behaviors that the system should exhibit under specific conditions. Functional requirements are typically detailed and specific, outlining the system's capabilities in terms of inputs, outputs, and processing logic.

Category	**Requirements**	**Type**
Hardware Development	At-least 720p (HD) Full-SBS resolution at 30fps	M
	Low-cost components total (less than £150)	M
	Audio recording support	S
	Additional sensors and peripherals	C
	Complex formats such as 180 degree	W
Stereo Processing	Automated stereo stitching	M
	Correct format conversion	M
	Scene detection metadata generation	S
	Immersive spatial audio mixing	C
	3D depth analysis for FVV	W
VR Software	Stereo Full-SBS format support	M
	File browsing and searching	M
	Scene filtering	S
	Hand and eye tracking support	C
	Compatibility with devices other than VR-HMDs	W

Table 3.1: Functional Requirements

3.1.2 Non-Functional Requirements (NFR)

Non-functional requirements, on the other hand, specify how a system should behave. They define the quality attributes or characteristics that the system must exhibit, such as performance, reliability, usability, and security. Unlike functional requirements, which focus on what the system does, non-functional requirements focus on how well the system performs its functions. These requirements are often more qualitative and subjective, describing the overall qualities and constraints that the system must adhere to in order to meet user expectations and regulatory standards.

Category	**Requirements**	**Type**
Hardware Development	Storage efficiency	M
	Ease of assembly and maintenance	M
	Compatibility with standard interfaces (USB, HDMI)	S
	Physical durability and robustness	C
	Energy consumption optimization	W
Stereo Processing	Computational efficiency	M
	Accuracy and reliability of scene detection	M
	Scalability for handling large datasets	S
	Flexibility for integrating with different camera setups	C
	Real-time processing capability	W
VR Software	Does not cause motion sickness	M
	Intuitive and user-friendly interface design	M
	Compatibility with popular VR hardware platforms	S
	Interesting 3D UI control elements	C
	Security measures to protect user data	W

Table 3.2: Non-Functional Requirements

3.2 Finalised Design

The full system architecture is comprised of three core components: **hardware**, **stereo processing pipeline**, and **VR software application**. These components work in tandem to capture, process, and present stereo media, offering users a seamless stereo life-logging experience.

3.2.1 Hardware

The hardware selection is critical to ensure effective capture of stereo images and videos. After careful evaluation, the Raspberry Pi 5 8GB, equipped with Raspbian

Lite OS, was chosen as the processing unit. Following are key design decisions that makes the capturing process seamless:

- Paired with two Raspberry Pi Camera Modules 3 (Wide) and a USB microphone, this configuration enables Full-HD resolution stereo capture at 30fps.
- Custom enclosures, designed and 3D printed, ensure precise alignment of stereo pairs.
- RTC clock is added for accurate filename timestamp even without being connected to the internet for a long time during boot.
- A python script is used to configure the behaviour of the capturing algorithm so systemCTL[1] functionalities were incorporated for seamless execution on startup.
- Developers can easily access and modify the script's operations, such as starting or stopping it, through SSH, making it a straightforward and user-friendly interface.
- The Raspberry Pi is powered with power-bank capable of at least 3A/5V Power Delivery via USB-C.
- To adhere to the budget constraint of under £150, audio quality was sacrificed by opting for a single, inexpensive low sensitivity USB microphone, resulting in the absence of stereo audio.

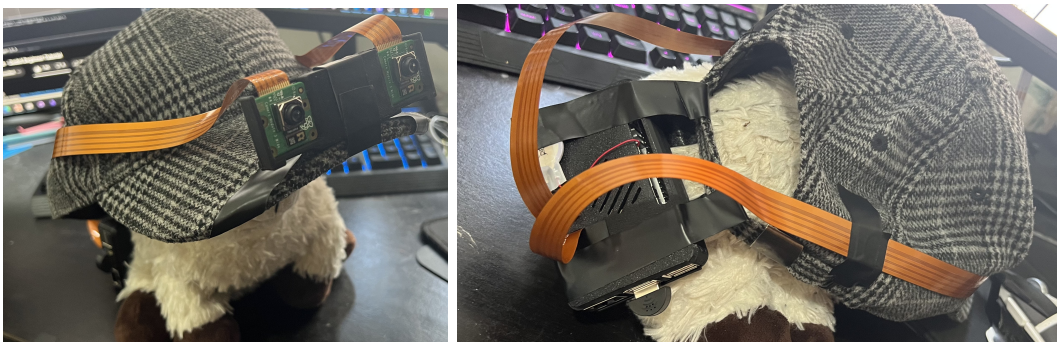

FIGURE 3.1: Final design front and back view.

[1] systemctl is a controlling interface and inspection tool for the widely-adopted init system and service manager systemd.

3.2.2 Stereo Processing Pipeline

Stereo processing is performed on the primary computer, utilizing its substantial computational capabilities. The captured stereo files follows this sequence:

1. The use of FTP over SSH with FileZilla facilitates file transfers, streamlining the process and ensuring dependability.

2. Image stitching and video manipulation tasks are managed by Python scripts that employ PILLOW and subprocess libraries, that call on FFMPEG.

3. Additionally, the use of the MAX-SCENE classifier API for automated metadata generation boosts organizational productivity by creating JSON formatted predictions for scene analysis. This enhances the efficient handling and organization of stereo media.

3.2.3 VR Software Development

The VR software application is developed using the Godot game engine, chosen for its adherence to open-source principles and robust scripting capabilities with GDScript. Custom shaders enable accurate depth effects, enhancing emotional engagement with stereo media. Additionally, VR controls offer intuitive interaction, enhancing user immersion and interface effectiveness. Despite not being as feature-rich as other engines, Godot meets project requirements and fosters community-driven development, aligning with the project's open-source ethos.

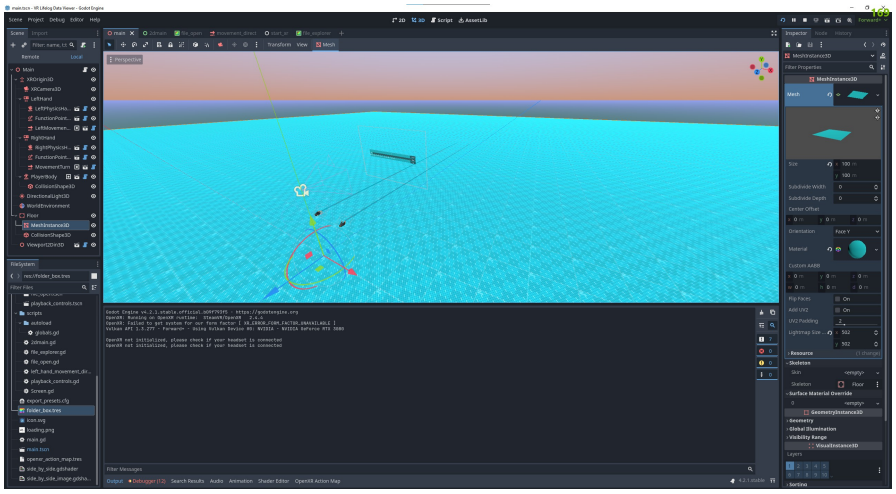

FIGURE 3.2: Godot Game Engine UI example, specific software features can be seen in Implementation section.

By integrating these components and design decisions, the stereo life-logging system offers users a compelling and immersive experience while providing flexibility and scalability for future advancements.

Chapter 4

Implementation

4.1 Development Process

The development process commenced with hardware development, concurrently with a preliminary feasibility study of Godot. This involved experimenting with custom shaders for side-by-side (SBS) viewing and implementing basic stereo screens within a 3D game world. Once hardware development reached completion and data acquisition was feasible, focus shifted to perfecting the stereo processing pipelines for immediate testing with captured media.

Stereo processing tasks included automating file crawling, stereo pair stitching, audio mixing, and metadata tagging for JSON generation. The resulting output was systematically transferred to designated folders for categorization based on the week, month, and year.

Subsequently after basic testing of stereo media successful, further software development for the VR application commenced. Initially, a simple 2D UI was developed before transitioning to a VR version. This involved integrating scene filters and refining UI/UX elements, including interactive 3D elements for enhanced user engagement. Throughout the development process, iterative improvements were made to ensure the functionality and usability of the VR application.

4.2 Component Details

Following is total cost breakdown of final prototype for crucial electronics components from ThePiHut as supplier:

No.	Item	Cost (£)	Quantity
1	Raspberry Pi 5	57.60	1
2	Raspberry Pi Camera Module 3	33.60	2
3	128GB MicroSD Card	23.00	1
		Total	147.80

Table 4.1: Cost Breakdown of Components

4.2.1 Processing Unit (Raspberry Pi 5 and Pico)

The Raspberry Pi 5 is a single-board computer, while Raspberry Pi Pico is a microcontroller board, both are developed by the Raspberry Pi Foundation. While they share the Raspberry Pi branding and are designed for a range of applications, they serve different purposes within the system architecture.

The Raspberry Pi 5, the latest iteration of the Raspberry Pi series, is a powerful and versatile single-board computer equipped with a quad-core ARM Cortex-A72 processor and up to 8GB of RAM. It offers significant improvements in performance compared to its predecessors, making it well-suited for tasks requiring intensive computation, such as image and video processing. With its Full-HD resolution image and video capture capabilities at 30 frames per second (fps) thanks to dual MIPI CSI Camera connection port, the Raspberry Pi 5 provides the processing power and interfaces necessary for capturing high-quality stereo footage.

In contrast, the Raspberry Pi Pico is a microcontroller board based on the Raspberry Pi Foundation's RP2040 microcontroller chip. It is designed for embedded applications and projects requiring real-time control and low-power operation. While the Raspberry Pi Pico is more affordable and energy-efficient than the Raspberry Pi 5, it has limitations in terms of processing power and connectivity. The SPI connection bandwidth and processing capabilities of the Raspberry Pi Pico are insufficient for handling the demands of stereo image and video capture at Full-HD resolution.

Ultimately, the Raspberry Pi 5 was selected over the Raspberry Pi Pico for its superior performance and capability to meet the requirements of the stereo processing pipeline. Its higher processing power and Full-HD resolution image and

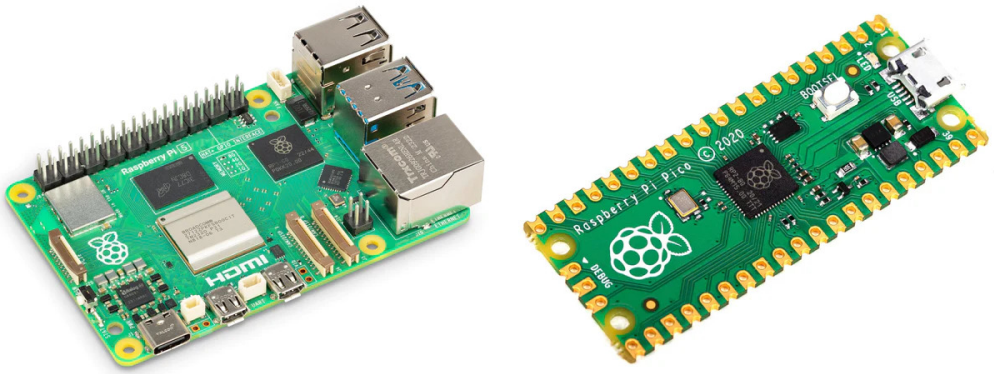

FIGURE 4.1: Raspberry Pi 5 and Pico

video capture capabilities ensure smooth operation and high-quality output for the system's stereo footage capture and processing tasks.

4.2.2 Cameras

4.2.2.1 ArduCAM 5MP OV5642

The ArduCAM 5MP OV5642 camera module features a 5-megapixel image sensor and is capable of capturing high-resolution images. It offers a wide range of configuration options for adjusting settings such as exposure, white balance, and focus. While the OV5642 module provides excellent image quality, compatibility issues arose when attempting to integrate it with the Raspberry Pi Pico. These compatibility challenges, coupled with concerns about SPI connection bandwidth limitations, led to the exploration of alternative camera options.

4.2.2.2 USB Webcams (Microsoft LifeCAM HD-3000)

USB webcams, such as the Microsoft LifeCAM HD-3000, were considered for their plug-and-play compatibility with Raspberry Pi and affordability. USB webcams are widely available and offer ease of use with a simple plug-and-play setup. However, while USB webcams provide convenience, they may lack the image quality and configurability compared to dedicated camera modules. Furthermore, they are bigger and heavier thus more cumbersome to wear. Although might be a cheaper option by procuring from used market or using existing webcam, most people only

have one webcam lying around, and this project requires two identical camera ideally for synchronised and quality stereo pair capture.

FIGURE 4.2: ArduCAM 5MP OV5642 and Microsoft LifeCAM HD-3000

4.2.2.3 Raspberry Pi Camera Module 3 (Wide)

The Raspberry Pi Camera Module 3 (Wide) is a dedicated camera module designed specifically for use with Raspberry Pi single-board computers. It features a wide-angle lens and is capable of capturing Full-HD resolution stereo footage at 30 frames per second. The Camera Module 3 offers seamless integration with the Raspberry Pi 5, providing compatibility and optimal performance for the stereo processing pipeline. Its compact form factor and flexible mounting options make it well-suited for embedded applications requiring stereo image and video capture.

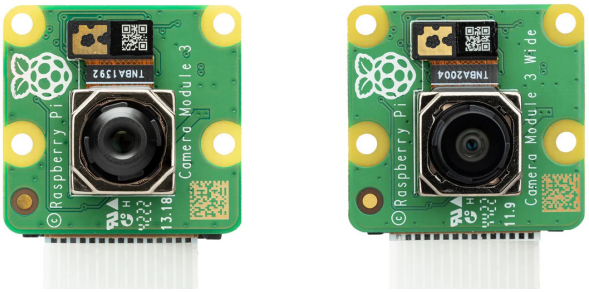

FIGURE 4.3: Rasberry Pi Camera Module 3 (Standard lens) and Rasberry Pi Camera Module 3 (Wide lens)

Ultimately, the Raspberry Pi Camera Module 3 (Wide) was chosen as the preferred camera option for its compatibility with the Raspberry Pi 5 and ability to meet the requirements of the stereo footage capture task.

4.3 Hardware Development

Following the selection of components, further refinement was essential to ensure acceptable quality and streamline the process. This involved the creation of a custom enclosure to house the components effectively, with careful consideration given to the mounting location for the camera. Additionally, the development of an onboard script was necessary to implement the lifelogging algorithm.

The Raspberry Pi 5 case is designed by @pyrho from Printables. The Raspberry Pi is powered with a power-bank that capable of atleast 3A/5V USB-C power delivery.

4.3.1 Custom Enclosure and POV Considerations

Achieving precise alignment between the stereo camera modules is crucial for generating accurate stereo images and videos. To ensure proper alignment, a custom 3D-printed enclosure was designed using Onshape. This enclosure allows for the precise mounting of the camera modules at a fixed distance and orientation relative to each other, minimizing any discrepancies in the stereo pair.

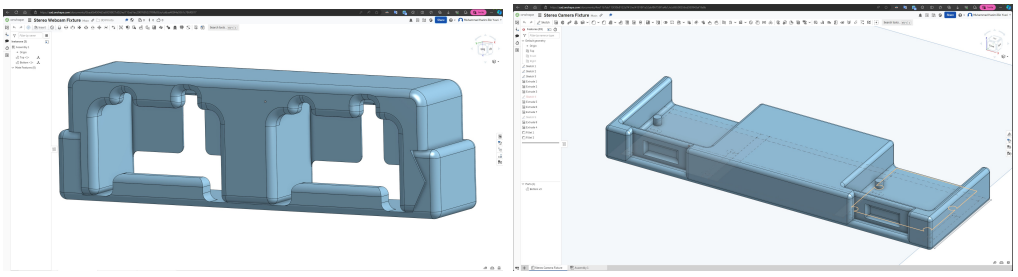

FIGURE 4.4: Onshape public sharing link for USB Webcam version and Raspberry Pi Camera Module 3 version

Additionally, considerations regarding the point-of-view (POV) were taken into account during the design process. Initially, a design resembling Ray-Ban Meta and Snapchat Spectacles glasses was considered to capture footage closely resembling human vision. However, practical constraints led to the adoption of a head-mounted design. Although the POV is slightly higher than eye level in this configuration, the movement of the cameras still follows the wearer's head movement, resulting in a realistic POV perspective as seen in Figure 4.5. This compromise between POV and practicality ensures an immersive viewing experience while maintaining ease of use and wearability.

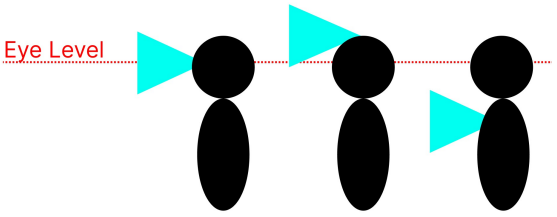

FIGURE 4.5: Designs and POV Example. Left to right: Glasses, Head-Mounted and Chest-Mounted

4.3.2 Onboard Life-logging Algorithm

The onboard life-logging algorithm script orchestrates the frequency and method of image and video capture within the system. It incorporates optimizations such as utilizing a Real-Time Clock (RTC) to ensure accurate timestamping even when not connected to the internet, and employing systemctl to automatically execute the script on boot. This ensures seamless operation and consistent data capture regardless of network connectivity.

The script, which runs on the Raspbian operating system, allows for remote access via SSH, enabling control and configuration adjustments without the need for additional interfaces or servers. By configuring the script parameters, users can easily modify the frequency and duration of image and video capture sessions to suit their specific requirements. This simplicity and flexibility make it well-suited for the scope of the project, facilitating efficient life-logging operations.

To further enhance efficiency, the script leverages optimized FFMPEG[30] and rpicam[1] commands for video recording and image capture. It also includes conditional logic to select between USB webcams and the Raspberry Pi camera module based on user preferences, ensuring compatibility and optimal performance.

Full onboard algorithm code is on Appendix 2, here is example of using FFMPEG an rpicam commands:

```
if use_webcam:
    # Define the FFmpeg commands for each USB webcam with optimized settings
    command_vid0 = f"ffmpeg -f v4l2 -framerate 24 -ss 1 -video_size 1280x720
    -input_format mjpeg -i /dev/video0 -preset ultrafast -pix_fmt yuv420p
    -t 30 {filename_vid0}.mkv"
    command_vid2 = f"ffmpeg -f v4l2 -framerate 24 -ss 1 -video_size 1280x720
    -input_format mjpeg -i /dev/video2 -preset ultrafast -pix_fmt yuv420p
    -t 30 {filename_vid2}.mkv"
else:
    # Define the libcamera/rpicam commands for the Raspberry Pi camera
```

1

```
command_vid0 = f"rpicam-vid --camera 0 --width 1920 --height 1080
--framerate 30 -t 30000 --codec mjpeg -o  {filename_vid0}.mjpeg"
command_vid2 = f"rpicam-vid --camera 1 --width 1920 --height 1080
--framerate 30 -t 30000 --codec mjpeg -o  {filename_vid2}.mjpeg"
```

When using FFMPEG and USB webcam combination, an issue arises with the Raspberry Pi overheating during successive video recordings, potentially leading to performance degradation. It is observed that despite the overheating, there are no noticeable frame drops. To address this, a trial is planned for using the "very fast" preset instead of "ultrafast" to assess if it mitigates heat throttling. However, uncertainties remain regarding whether the choice of preset, or indoor/low-light conditions contribute to stuttering and flickering, warranting further investigation. Additionally, the "-ss 1" parameter, implemented to skip the first frame for video thumbnails, is noted as necessary for better thumbnail quality.

Considering the continuous capturing of images and videos during the user's waking hours from 0800 to 2400, which spans approximately 16 hours each day, the estimated data usage can be calculated. Assuming an infinite battery life, the system is expected to capture 12 images and record two 30-second videos every hour.

This results in a data consumption of approximately 12MB for images and 720MB for videos per day, totaling to approximately 11532 MB (11.532GB) over the 16-hour period. Extrapolating this usage over a week, the cumulative data requirement amounts to approximately 80.724GB.

Given this estimated data consumption, implementing a weekly backup and data transfer strategy becomes imperative to ensure data integrity and manage storage capacity effectively. This approach aligns with the project's objectives of maintaining a consistent and reliable lifelogging system while managing data storage efficiently.

4.4 Stereo Processing Pipeline

4.4.1 FileZilla (FTP via SSH)

FileZilla[2] software is utilized for transferring media capture from the Raspberry Pi 5 to the main/host PC for processing and storage. It uses File Transfer Protocol

[2]

(FTP) via Secure Shell (SSH). This method provides a secure and efficient means of transferring files over a network connection. By leveraging FileZilla, the system minimizes manual intervention and simplifies the data transfer process, improving overall workflow efficiency. Manual data transfer from SD card is not recommended as it will increases the likelihood of data corruption which can jeopardise both captured content and Raspberry Pi OS installed, prompting a reinstall.

4.4.2 Python Scripts and FFMPEG

Python scripts play a crucial role in the stereo processing pipeline, facilitating automation and manipulation of captured media. FFMPEG, a powerful multimedia processing tool, is integrated into the pipeline for tasks such as format conversion, audio mixing and stereo pair stitching. For example, both stereo pair images and video need to be stitched together according to the camera left and right order, which is denoted with 0 and 2.

The utilization of FFmpeg [30] for mono to stereo stitching serves as a cornerstone in the pre-processing pipeline. FFMPEG, a powerful multimedia processing tool, efficiently transforms mono/2D videos and images into the desired stereoscopic Side-By-Side (SBS)/3D format as seen in Fig 4.6. This process is instrumental in creating a lifelike and immersive visual experience for VR content, aligning with the project's goal of democratizing VR content creation.

Snippets of stitching codes, full codes at Appendix 2:

```
    # Define the ffmpeg command to stitch the videos side by side
ffmpeg_command = [
    "ffmpeg",
    "-i", vid2_path, #raspi cam use 2,0 instead of 0,2
    "-i", vid0_path,
    "-filter_complex", "[0:v][1:v]hstack=inputs=2",
    "-c:v", "libtheora",  # Use Theora codec for .ogv
    "-qscale:v", "7",  # Set quality scale for Theora codec
    "-an",  # Disable audio
    "-map_metadata", "0:g:0",  # Copy metadata from the first input file
    "-y",  # Overwrite output file without asking
    output_path.replace(".mkv", ".ogv")  # Change output file extension to .ogv
]

# Run ffmpeg command
try:
    subprocess.run(ffmpeg_command, check=True)
    print(f"Videos stitched successfully: {output_path}")
except subprocess.CalledProcessError as e:
    print(f"Error stitching videos: {e}")
```

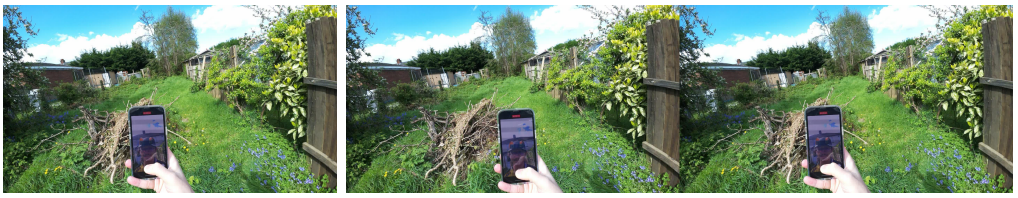

FIGURE 4.6: Examples of Mono (left) and Stereo (right) images that are stitched from two mono stills

Other than that, the format to display video are strict as Godot videostream-player node only support .ogv files, thus conversion from .mkv to .ogv is needed, furthermore from .mjpeg to .mkv first when using Raspberry Pi Camera as that configuration resulted into fastest and smoothest video capture due to no encoding on the Raspbery Pi itself. Important thing to note is that FFMPEG can't overwrite directly to file being converted, instead a temporary file is generated, renamed and old one removed. It is also then rotated 180 as the camera is mounted upside down.

Code snippet, with full codes in Appendix 2:

```
# Define the ffmpeg command to convert the video
ffmpeg_command = [
    "ffmpeg",
    "-i", input_path,  # Input video file
    "-c:v", "copy",  # Copy video stream without re-encoding
    "-r", "30",  # Set the output framerate to 30 fps
    "-f", "matroska",  # Output format: Matroska (MKV)
    "-y",  # Overwrite output file without asking
    output_path
]

# Run ffmpeg command
try:
    subprocess.run(ffmpeg_command, check=True)
    print(f"Video converted successfully: {output_path}")
    # Remove the original .mjpeg file
    os.remove(input_path)
    print(f"Removed original file: {input_path}")
except subprocess.CalledProcessError as e:
    print(f"Error converting video: {e}")
```

To streamline it further, .bat file is created to run all the required scripts sequentially, refer Appendix 2. Together, these tools enable seamless processing and preparation of stereo footage for viewing in the VR software application.

4.4.3 MAX-Scene Classifier (Scene Metadata .json Generation)

The MAX-Scene Classifier[31] by IBM is instrumental in our stereo processing pipeline, enabling scene recognition and metadata tagging. To integrate this functionality, a Docker image provided by MAX-Scene Classifier was utilized. By deploying this Docker image, a service environment was set up to call the MAX-Scene Classifier API and obtain scene predictions in JSON format. This approach facilitated the generation of metadata for captured images and videos, with the JSON response files saved under the same name as the corresponding media files.

Code snippet of API request, full codes in Appendix 2:

```
def make_prediction_request(file_path):
    url = "http://localhost:5000/model/predict"
    files = {'image': open(file_path, 'rb')}
    headers = {'accept': 'application/json'}
    response = requests.post(url, files=files, headers=headers)
    print(response.json)
        # Write prediction to JSON file
    prediction_json_path = os.path.splitext(file_path)[0] + "_prediction.json"
    with open(prediction_json_path, 'w') as json_file:
        json.dump(response.json(), json_file, indent=4)

    return response.json()
```

For images, the process is straightforward as only a single frame is analyzed. However, for videos, five evenly spaced frames are extracted and used for scene prediction analysis. Consequently, this results in the generation of five separate .json files for each video file. These .json files contain the scene predictions derived from the sampled frames and are utilized for filtering within the VR software application.

4.5 VR Software

4.5.1 Godot Game Engine

The choice of the Godot game engine[32] for developing the video player VR app stems from the project's commitment to Free and Open Source Software (FOSS) principles. Unlike more established game engines such as Unity or Unreal Engine, Godot aligns with the FOSS ethos, making it the ideal choice for this project.

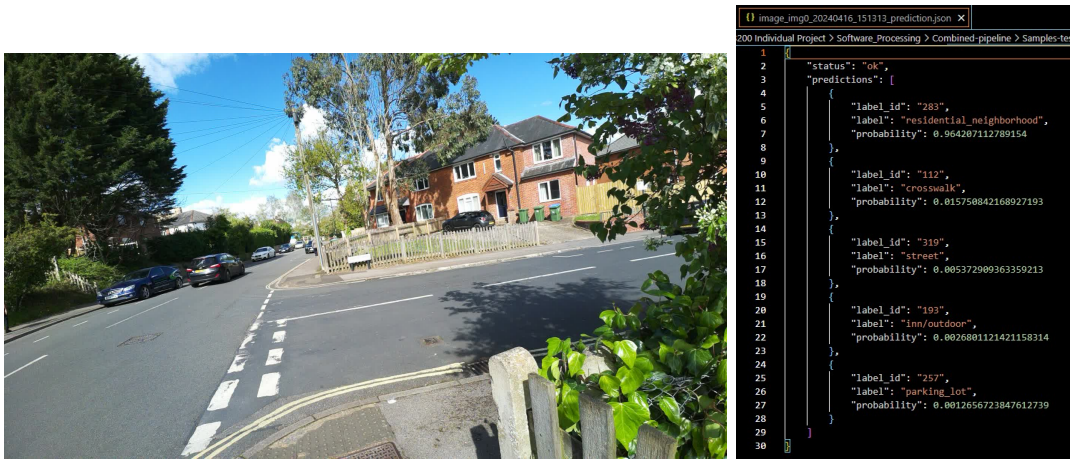

FIGURE 4.7: Example of image and the prediction api .json response.

This is also beneficial for our open-source project as it promotes community development and ensures project longevity, as it does not rely on other vendors and is fully customizable. This is evident from the multitude of different kinds of apps created in Godot, which are not exclusive to games but also include apps such as Pixelorama and Godello. Additionally, its lightweight nature makes development and prototyping easier due to faster compile times and hot reload capabilities. Furthermore, its custom language, GDScript, is similar to Python, making it easy to learn and reducing time spent grappling with language intricacies.

The VR software application is built upon the robust foundation of the Godot version 4.2.1 game engine. Godot's intuitive interface and extensive documentation provide a conducive environment for rapid development, facilitating the creation of engaging VR software.

4.5.2 SBS Video Projection

Consultation with the Godot community, particularly through their Discord channel, guided the decision-making process, revealing that utilizing shaders is the most straightforward approach for rendering stereo Side-By-Side (SBS) video playback. In the implementation of stereo video playback using the Godot 4.2 game engine, shaders play a crucial role in rendering. The process is straightforward, aligning with the Side-By-Side (SBS) format of the video. The left half is rendered for the left camera eye, and the right half for the right eye. The gdshaders code in Appendix 2 utilized adheres to this logic, ensuring an efficient rendering process.

However, a challenge arises as the video player operates as a 2D screen within this 3D environment. Initial attempts to integrate the shader script within the same scene as the videostreamplayer node, following a tutorial by Malcolm Nixon on YouTube, faced difficulties. Subsequent experimentation and development efforts proved inconclusive.

Fortunately, after seeking guidance from the Discord community, Malcolm Nixon, the tutorial's author, provided a pivotal solution [33]. The revised method involves instantiating a 2D screen videostreamplayer node in the main scene, where the shader is then applied. This modification successfully addresses the challenges encountered during the development process. An example of the app running is depicted in Figure 4.8.

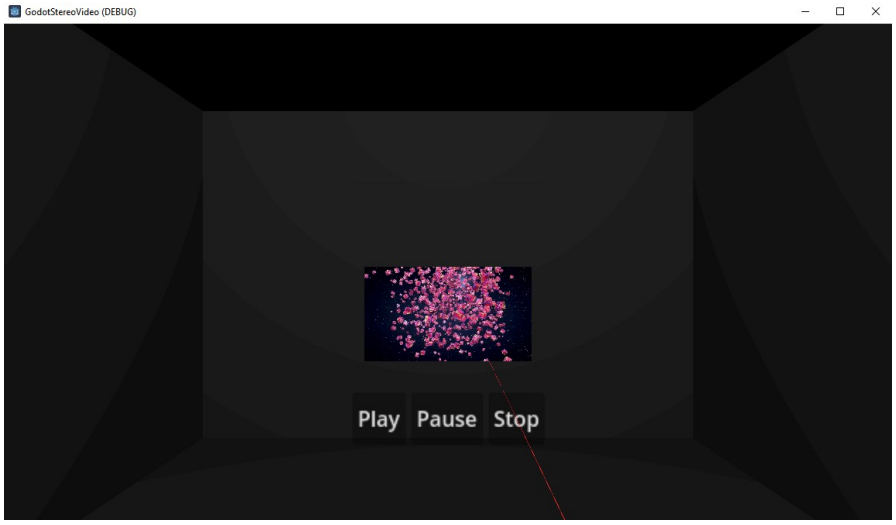

FIGURE 4.8: Images of App Running

4.5.3 File Browsing

REDO, mention using tutorial video as base, then added lots more functionality to fit, such as metadata fitlers etc

Proposed ideas include leveraging metadata tagging for specific searches and employing bookshelves or 3D objects as interfaces for navigating between months or weeks, deviating from the conventional 2D screen approach to enhance interactivity. To optimize browsing efficiency, the screen space may be expanded to resemble an ultra-wide monitor or more, enabling users to view a greater number of files in a single window. This approach capitalizes on the immersive capabilities of VR, utilizing the 360-degree view and 6 Degrees of Freedom (6DoF). Additionally, the

implementation of the depth axis remains contingent on contextual considerations and future developments [24].

However, this is too complex at the moment and need a good basic foundation first. Thus, following this tutorial by WorldBuildingBert to create a solid 2D file browser app as a foundational base in which more features are added.

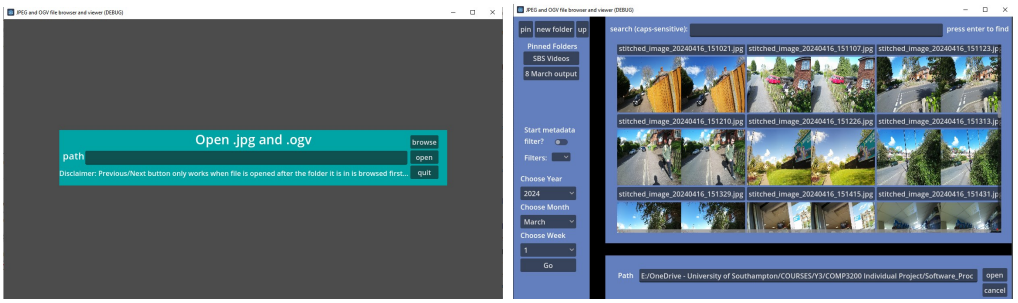

FIGURE 4.9: The simple main menu and file explorer (with thumbnails) 2D app screenshots

One of the critical features added specific to this project is adding thumbnails for the .jpg and .ogv files. This is done by checking the file type then instantiating a textureRect node for each file. This works however causes a lag as the software needs to load all files in the folder to generate the thumbnails. Thus, in future revisions, optimisation on this regard is needed, either by parallelization or only generating thumbnail as needed (scrolling).

Another important features is the filtering based on the scene detected using the .json metadata generated. First iteration to use embedded metadata using piexif and ffmpeg proved difficult as there is no simple way to read those data in Godot using GDScript, thus why .json metadata file with same name corresponding to file it predicted is used. First, by reading through *_json_analysis.txt file that is generated from stereo processing pipeline beforehand, the dropdown menu is populated. Then, according to filter selected, only content that match selected filter is shown. This however includes all 5 predicted response, regardless of probability. Future work should refine on this aspect more to give more control and refinement to users.

When dealing with large datasets such as life-logging, proper organisation is needed, thus to properly manage and index them, a system to use folder by separating them into Years¿Month¿Week is used. As the file browser functionality is also bounder with OS conventional directory system, this means a simple UI to go to specific

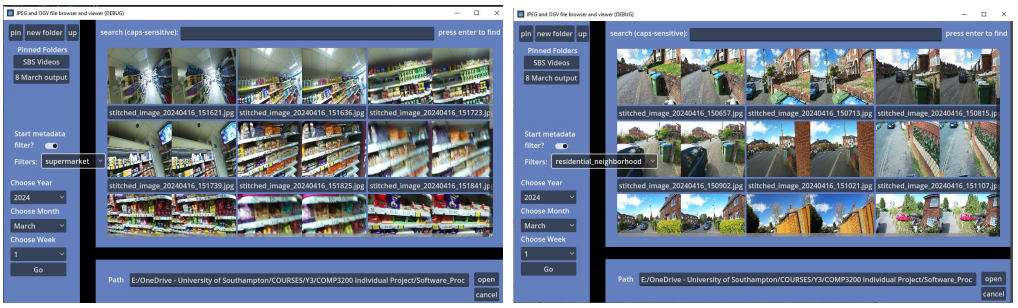

FIGURE 4.10: Example filter with prediction of 'supermarket' and 'residential neighbourhood'

date range can be implemented easily as shown. Further more, as the file are name with timestamp, specific time and date can be search easily as well.

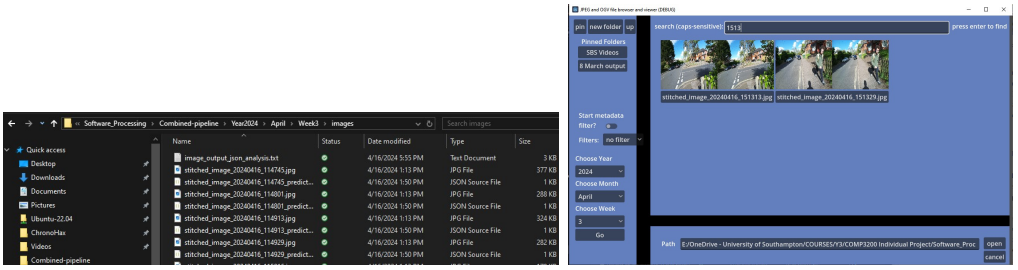

FIGURE 4.11: Example of folder directories and search result of '1513' ie 3:13PM.

4.5.4 2D UI and VR Port

During the development process, initial emphasis was placed on creating a user-friendly interface (UI) that seamlessly transitions between 2D UI to VR environments, this is important to make sure the barebones functionality of browsing and viewing works before making it more complicated. Other innovative UI design concepts, such as using 3D game elements like bookshelves and tapes for browsing images and videos, to enhance user engagement and interaction are also easier to implement with this strong foundation. Thus, the VR software was meticulously ported from a 2D UI application, ensuring continuity in functionality and user experience across different platforms. This strategic approach allowed for the integration of VR-specific features while maintaining familiarity for users accustomed to traditional 2D interfaces. However, unfortunately as Godot videostreamplayer node only supports playback of .ogv files, the superior .mpo file format can't be used, thus 2D APP will see Full-SBS image and video as well, it is possible to solve this by cropping but it is left out for now as not the main focus of this project.

The bulk of the work are made easier thanks to the Godot-XR-tools addons developed by Godot XR team. Particularly, the port from 2D UI app to 2D screen inside 3D VR game is done with the use of included Viewport2Din3D packed scene. However, as the screen to display the image and video now should be a 3D node object instead of 2D, major rework on that part needed to be done. Thankfully, as preliminary work regarding this had been done using similar technique with Viewport2Din3D and SBS shaders, it was completed flawlessly. Next, and Previous buttons are also added when image is viewed, together with Play, Pause and Stop button for videos.

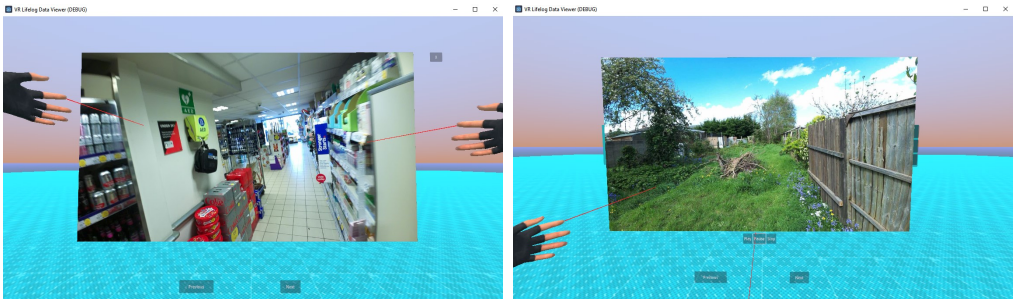

FIGURE 4.12: Additional buttons added to aid user browsing experience in image and video viewing screen.

Other than making sure the 3D depth effect work properly, certain UI elements and size are also adjusted, prominently, bigger buttons, and selectable thumbnails instead of only the button. This makes it easier for VR users to navigate through the menu as pointer selection is implemented. Virtual keyboards are also added so search bar can be used.

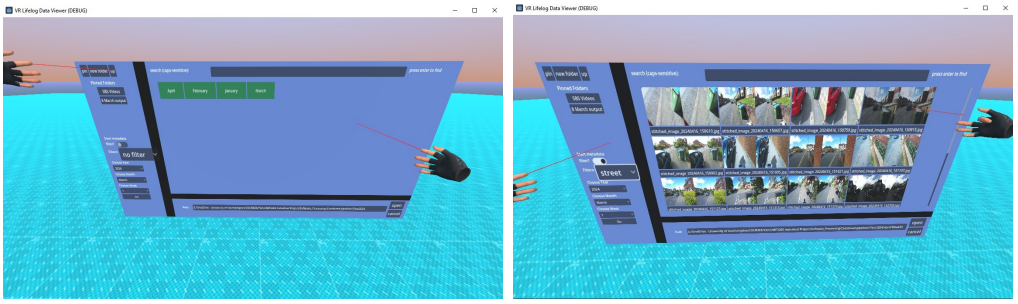

FIGURE 4.13: Example of bigger UI buttons and option to select by clicking on thumbnails including visual feedback.

User locomotive is set to default smooth locomotion with strafing on left controller and smooth turning on right controller. The choice of smooth locomotion is deliberate as it gives user a more refined control to adjust their position compared to screen. In the future, a static user with grab-able and movable screen menu should be implemented as well as it allows more freedom. Although the choice

might seems counter-intuitive to combat motion sickness at first, it is not as impactful as movement is only in slight adjustment, and as the screen is static, with adjustable distance from user control via movement, a virtual anchor is set to prevent motion sickness[25]. In worst case, as the project is open-source and the movement is using XR-tools addons, to change the locomotion setting is simply by changing the dropdown selection in the node attribute.

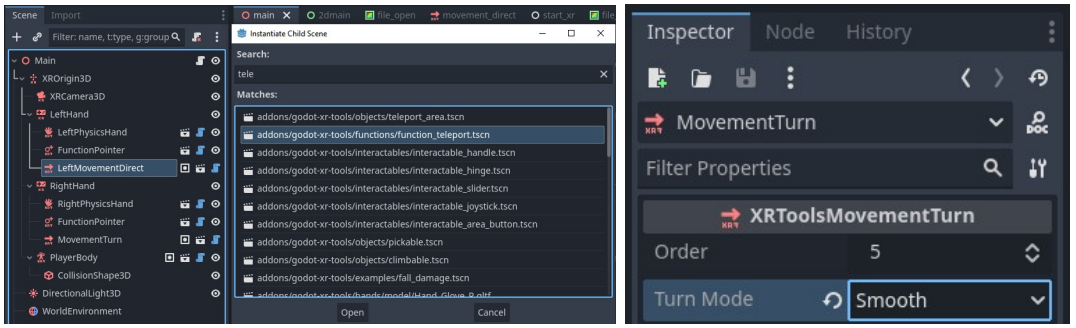

FIGURE 4.14: To change from smooth locomotion, simply drag and replace LeftMovementDirect with function_teleport. To change from smooth turning to snap turning, just simply choose it from the dropdown.

The app is then exported as .exe using the recommended settings and presets, however .apk export doesnt seem to work properly as the file system in android is different than Windows thus messing up the core working of file browser. Even after a lot of experimentation, videos are managed to be exported locally together with the .apk, however the images .jpg are missing, and the video playback stutters and have lower resolution so suffice to say it only works for PCVR/SteamVR for now.

Chapter 5

Testing and Evaluation

5.1 Testing Strategy

The testing strategy is structured around the previously defined functional and non-functional requirements, serving as the basis for unit testing, integration testing, and acceptance testing.

- **Unit Testing:** Each individual component of the system undergoes unit testing based on the functional and non-functional requirements assigned to it. This ensures that each component performs its intended function effectively and meets the specified criteria.
- **Integration Testing:** Integration testing involves running the entire hardware setup for an extended period indoors and outdoors to capture footage. The captured footage then undergoes seamless processing through the stereo processing pipeline. Finally, the processed media is viewed using the VR software application to ensure compatibility and functionality. Integration testing validates the interoperability of all components and the overall system performance.
- **Acceptance Testing:** The results of unit and integration testing inform the acceptance criteria. Further modifications and refinements are made iteratively until the testing results meet the predefined acceptance criteria. This iterative process ensures that the final design aligns with the project's objectives and requirements.

5.2 Test Results

The testing phase revealed insights into the performance and functionality of each system component, highlighting both successes and challenges encountered throughout the process.

Integration Testing: Initial integration testing encountered challenges with the Raspberry Pi Pico and OV5640 camera combination due to limitations in SPI connection bandwidth, resulting in low frame rates and latency issues. To address this, a Raspberry Pi 5 was used instead, albeit with a larger form factor. Subsequent testing with a USB camera exhibited subpar resolution and frame rate performance, leading to further exploration with the CSI-MIPI connection camera (Raspberry Pi Camera 3 Wide angle). This camera proved successful, providing Full-HD resolution at 30fps with smooth performance, thus enabling successful integration testing. Challenges met on other components are less critical and addressed in unit testing part.

Unit Testing: Unit testing results were evaluated based on the predefined functional and non-functional requirements.

5.2.1 Functional Requirements (FR)

M: Must-haves, S: Should-haves, C: Could-haves, W: Wont-haves

Hardware Development:

- [**M1**] Atleast 720p (HD) Full-SBS resolution at 30fps: **PASSED** as the final design manages to get Full-SBS 1080p at 30fps.
- [**M2**] Low-cost components total (less than £150): **PASSED** as the final total components cost is £147.8 which is less than £150
- [**S1**] Audio recording support: **PARTIALLY** as it have single audio channel recording but the quality is underwhelming due to low sensitivity and quality microphone.
- [**C1**] Additional sensors and peripherals: **FAILED** as there is currently no additional sensors and peripherals like Inertial Measurement Unit (IMU), Global Positioning System (GPS) module, etc.

- [**W1**] Complex formats such as 180 degree: **PASSED** as the current lens is only 102 degree, which is still suitable for Full-SBS windowed viewing format although a bit squished.

Stereo Processing:

- [**M3**] Automated stereo stitching: **PASSED** as images and videos stereo pair stitched to Full-SBS format successfully.
- [**M4**] Correct format conversion: **PASSED** as videos are converted from mjpeg to mkv to ogv.
- [**S2**] Scene detection metadata generation: **PASSED** as the predicted scene is very accurate thanks to very good model.
- [**C2**] Immersive spatial audio mixing: **FAILED** as there is no stereo audio channel for immersive spatial audio mixing.
- [**W2**] 3D depth analysis for FVV: **PASSED** as no work is done on 3D depth analysis and reconstruction for FVV.

VR Software:

- [**M5**] Stereo Full-SBS format support: **PASSED** as the software playback both images and videos on 3D format as intended.
- [**M6**] File browsing and searching: **PASSED** as it supports the basic file browser functionality with enhancements specific for VR use.
- [**S3**] Scene filtering: **PASSED** as the software makes use of predicted results for further filtering based on scenes.
- [**C3**] Hand and eye tracking support: **FAILED** as experimentation with eye and hand tracking in Godot proved implementation is too difficult to achieve.
- [**W3**] Compatibility with devices other than VR-HMDs: **PASSED** as the focus and scope is only towards VR HMDs, the program can be exported as 2D UI app, but the content would still be in Full-SBS format.

All must-haves and should-have passed except 1 partially passed, while all could-haves failed due to lack of time. All wont-have also passed which shows correct prioritization.

5.2.2 Non-Functional Requirements (NFR)

M: Must-haves, S: Should-haves, C: Could-haves, W: Wont-haves

Hardware Development

- [**M1**] Storage efficiency: **PASSED** as based on calculations, a 128GB SD CARD would take around a week to fill up, necessitating weekly backup/file transfer.
- [**M2**] Ease of assembly and maintenance: **PASSED** as the enclosure is designed with 3D printing in mind, and the code and .stl 3D design is open source and modularized for easy maintenance.
- [**S1**] Compatibility with standard interfaces (USB, HDMI): **PASSED** as Raspberry Pi 5 adheres to these standards.
- [**C1**] Physical durability and robustness: **PARTIALLY** as it is made from PLA, but due to fashion choices (to not be discreet) and to prevent component overheating, full enclosures for both Pi and cameras are not designed, meaning it's not water-resistant.
- [**W1**] Energy consumption optimization: **PASSED** as there is no explicit power consumption optimization on both the Pi, and a power bank is used to power it, so no hardware-side optimization is implemented.

Stereo Processing

- [**M1**] Computational efficiency: **PASSED** as the stereo processing pipeline utilizes optimized algorithms to ensure efficient use of computational resources.
- [**M2**] Accuracy and reliability of scene detection: **PASSED** as the MAX-SCENE Classifier API provides accurate scene recognition predictions.
- [**S1**] Scalability for handling large datasets: **PASSED** as the system architecture is designed to scale with increased data volume, leveraging the processing power of the main PC.
- [**C1**] Flexibility for integrating with different camera setups: **PASSED** as the stereo processing pipeline is modular and can accommodate various camera configurations with minimal adjustments.

- [**W1**] Real-time processing capability: **PASSED** as the stereo processing pipeline requires time to process especially videos proportional to amount of captured content.

VR Software

- [**M1**] Does not cause motion sickness: **PASSED** as the VR software application employs techniques to minimize motion sickness-inducing effects.
- [**M2**] Intuitive and user-friendly interface design: **PASSED** as the VR software application is designed with user experience in mind, featuring a straightforward and intuitive interface.
- [**S1**] Compatibility with popular VR hardware platforms: **PARTIALLY** as the VR software application is compatible with widely used VR hardware platforms such as SteamVR but does not work fully on Meta Quest's Horizon OS.
- [**C1**] Interesting 3D UI control elements: **FAILED** as there is no interesting 3D UI control elements implemented.
- [**W1**] Security measures to protect user data: **PASSED** as the VR software application does not implements any encryption and authentication protocols yet.

Most non functional requirements set passed to certain degree except 2 partially passed in Should-haves and 1 failed in Could-haves.

5.3 Performance Evaluation

The statistical perfomance evaluation of the system through user evaluation should be based on full experience of the system to draw reliable and conclusive evidence. However that is difficult to achieve due to nature of the project (need to lend the hardware to lifelog which poses risks). User evaluation of only the VR software is considered, however, this would be less meaningful in terms of user emotion as the footage shown is not capture by themself. Therefore, the performance evaluation relies on technical specification comparisons with alternative system configurations and commercially available products to see price to features ratio.

Performance comparisons were made against alternative system configurations such as using a USB webcam versus the CSI-MIPI connection camera (Raspberry Pi Camera 3 Wide angle). Additionally, comparisons were made with commercially available products like the Kandao Qoocam Ego to assess performance against established benchmarks.

Criteria	USB-Webcam	MIPI CSI Camera	QooCam Ego
Resolution	1280x720p	3840x1080	3840x1080
FOV (horizontal)	60	102	65
FPS	Choppy 30	30	60
Audio	Mono	Mono	Stereo
Storage	microSD	microSD	microSD
Design	Handsfree	Handsfree	Handheld
Ecosystem	Open Source	Open Source	Closed System
Price	£130	£150	£319

TABLE 5.1: Comparison between USB-Webcam, MIPI CSI (Raspberry Pi Camera Module 3 Wide) versions and Kandao QooCam Ego 3D Camera

As seen from the table, the comparison highlights distinct advantages and trade-offs among the USB-Webcam, MIPI CSI Camera (Raspberry Pi Camera Module 3 Wide), and the Kandao QooCam Ego 3D Camera.

The MIPI CSI Camera emerges as the best bang for the buck, offering superior resolution and a wider horizontal field of view (FOV) at £150. Its affordability combined with high-quality performance makes it an attractive option for users seeking cost-effective solutions without compromising on image quality. Furthermore, being an open-source ecosystem, it provides users with the most freedom and customizability to tailor the system according to their specific needs and preferences.

Both the USB-Webcam and MIPI CSI Camera feature a hands-free design, making them well-suited for lifeloggers who require seamless and hassle-free operation during their daily activities. This design choice ensures convenience and ease of use, allowing lifeloggers to focus on capturing moments without being encumbered by handheld devices.

In contrast, the Kandao QooCam Ego boasts better specification such as stereo audio capture and a higher frame per second (fps) capture, catering to users who prioritize enhanced audiovisual experiences. However, its closed system may limit customization options compared to the open-source counterparts.

Ultimately, the choice between these options depends on the user's priorities, balancing factors such as budget, image quality, customization potential, and design preferences. While the Kandao QooCam Ego excels in certain aspects, the MIPI CSI Camera stands out as a versatile and cost-effective solution, offering both high performance and customization possibilities for stereoscopic lifelogging and content creation needs.

Chapter 6

Project Management

The development process for implementing the system involved a combination of methodology and tools to ensure efficiency and organization. As mentioned before in testing chapter specifically integration and acceptance testing, evolving project management is applied while Git version control was utilized for managing project files and ensuring backups, providing a structured approach to development. Additionally, Gantt charts were employed for project planning and tracking progress over time. This ensures the project to be on track while being flexible towards the scope and achievements.

Project files were organized into categorized folders to maintain clarity and ease of access. For example, the "Hardware_mount" folder contained STL files exported from the 3D enclosure designed using Onshape for 3D printing. Similarly, the "Software_Algorithm" folder housed the "run_me.py" code containing algorithms for lifelogger behavior, while the "Software_Game" directory stored all source code for the Godot project files. The "Software_Processing" folder contained Python scripts related to stereo processing tasks. Every single one of the folders is stored on both OneDrive and GitHub to ensure data redundancy and expedite synchronization.

6.1 Gantt Charts

This project consists of three distinct components, simplified to their basic elements to demonstrate a minimum viable product prototype. Therefore, meticulous

planning and initial design drafting are essential, and this has been executed thoroughly, as evidenced. However, the challenging phase lies in translating the bulk of the design into a functional solution, which is shown with how the final design changes varily from initial design.

As observed in the Gantt charts in Figure 6.1, Figure 6.2 and Figure 6.3, the proposed timeline has encountered some deviations, primarily due to unforeseen risks such as the Raspberry Pi Pico microcontroller breaking due to a short circuit during user testing and instability in the connections of the wires provided by manufacturers, thereby causing delays in testing. This had been recorded in the risk assessment in the Appendix 2. Nevertheless, all three main components underwent preliminary testing, and initial work was successfully accomplished during first phase of work before interim progress submission.

Despite efforts, the latency issue persisted, rendering the video unusable. Further investigation specifically on manufacturer's Github Issue page revealed that the combination of the Raspberry Pi Pico microcontroller and ArduCAM OV5642 SPI camera was not suitable for smooth video recording as mentioned in implementation section, despite claims of 720p30fps capability on the manufacturer's specifications and website. This issue delayed the hardware development schedule. However, a contingency plan involving a more bulkier but simpler setup using USB webcams and a Raspberry Pi was implemented, which expedited the hardware setup process. Ultimately, to enhance resolution and video performance, MIPI CSI cameras was employed.

Figure 6.2 displays the executed Gantt chart timeline before progress report submission alongside the planned one, distinguished by varying colour densities. Figure 6.3 display same graph but after progress report. It can be seen how unexpected issues throws me off the planned schedule, however it is still useful as it helps me gauge my capabilities with time remaining and tells me which part to focus on first.

6.2 Reflections

Throughout the project, various aspects of project planning and management have been instrumental in shaping the course of development. Reflecting on the process, several positive and negative aspects emerge, offering valuable insights for future endeavors.

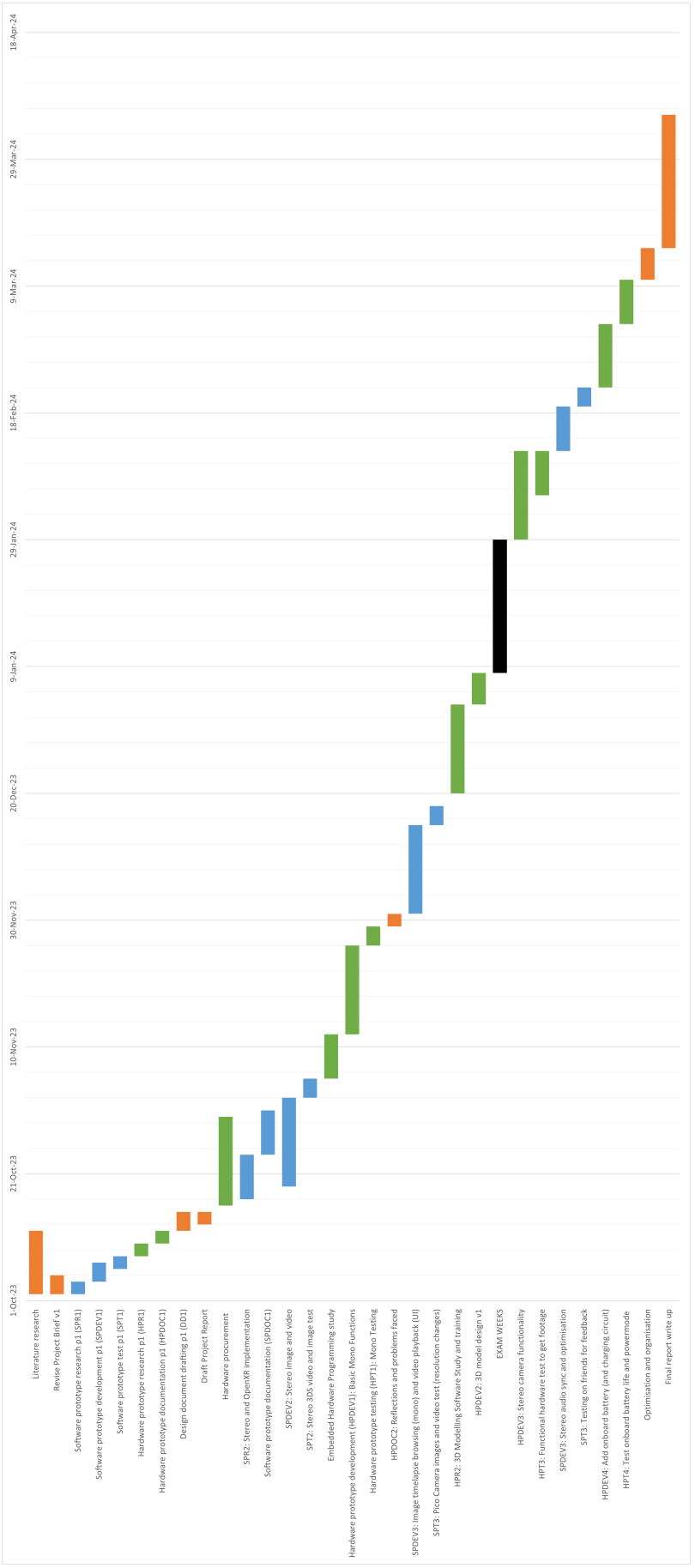

FIGURE 6.1: Planned Gantt Chart. Legend: Orange - Report or writing related task, Blue - Software related task, Green - Hardware related task.

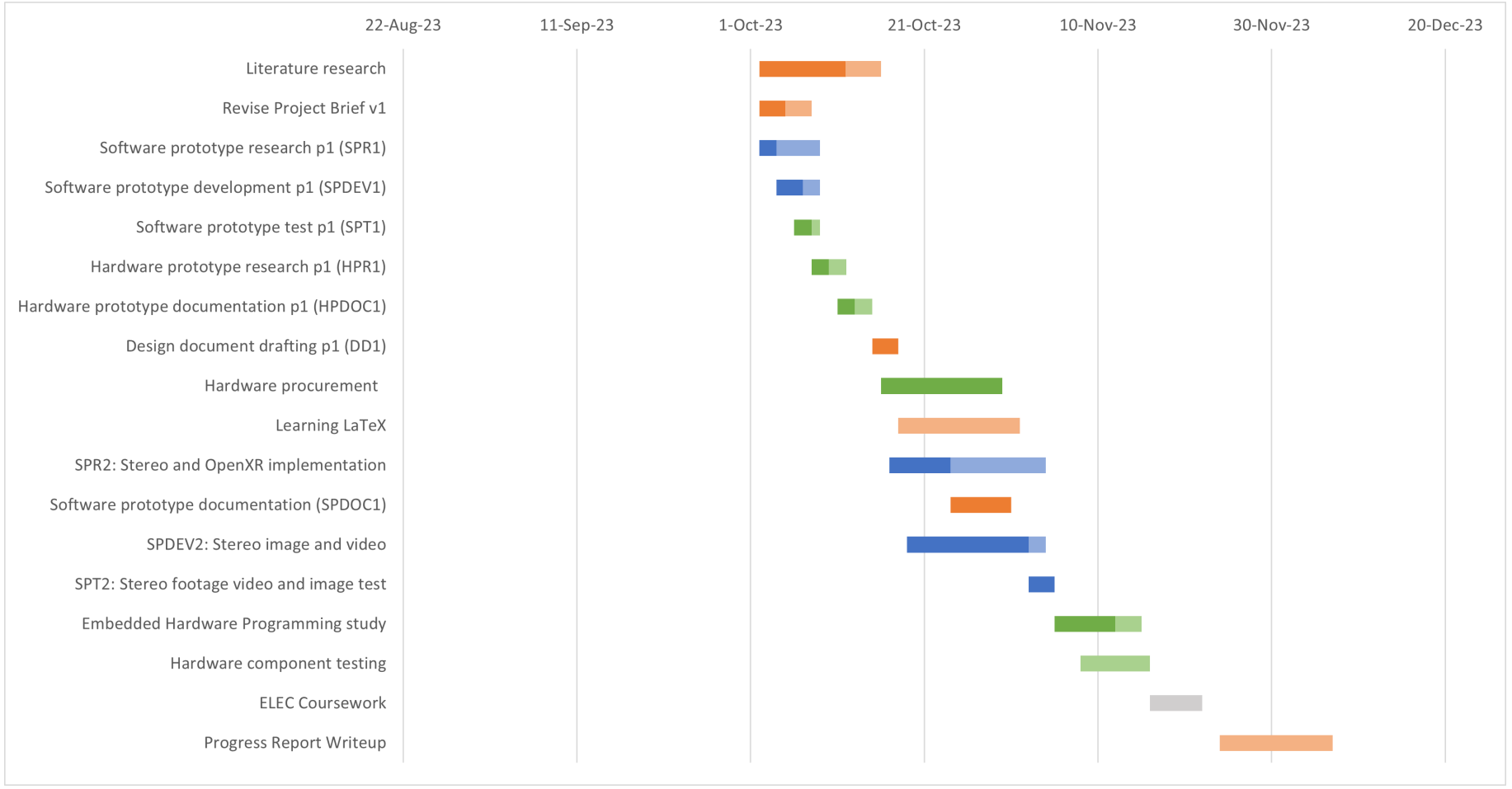

FIGURE 6.2: Combined Gantt Chart before Progress Report submission. Legend: Orange - Report or writing related task, Blue - Software related task, Green - Hardware related task. Softer color represents tasks over expected time or unexpected tasks compared to the Planned Gantt chart.

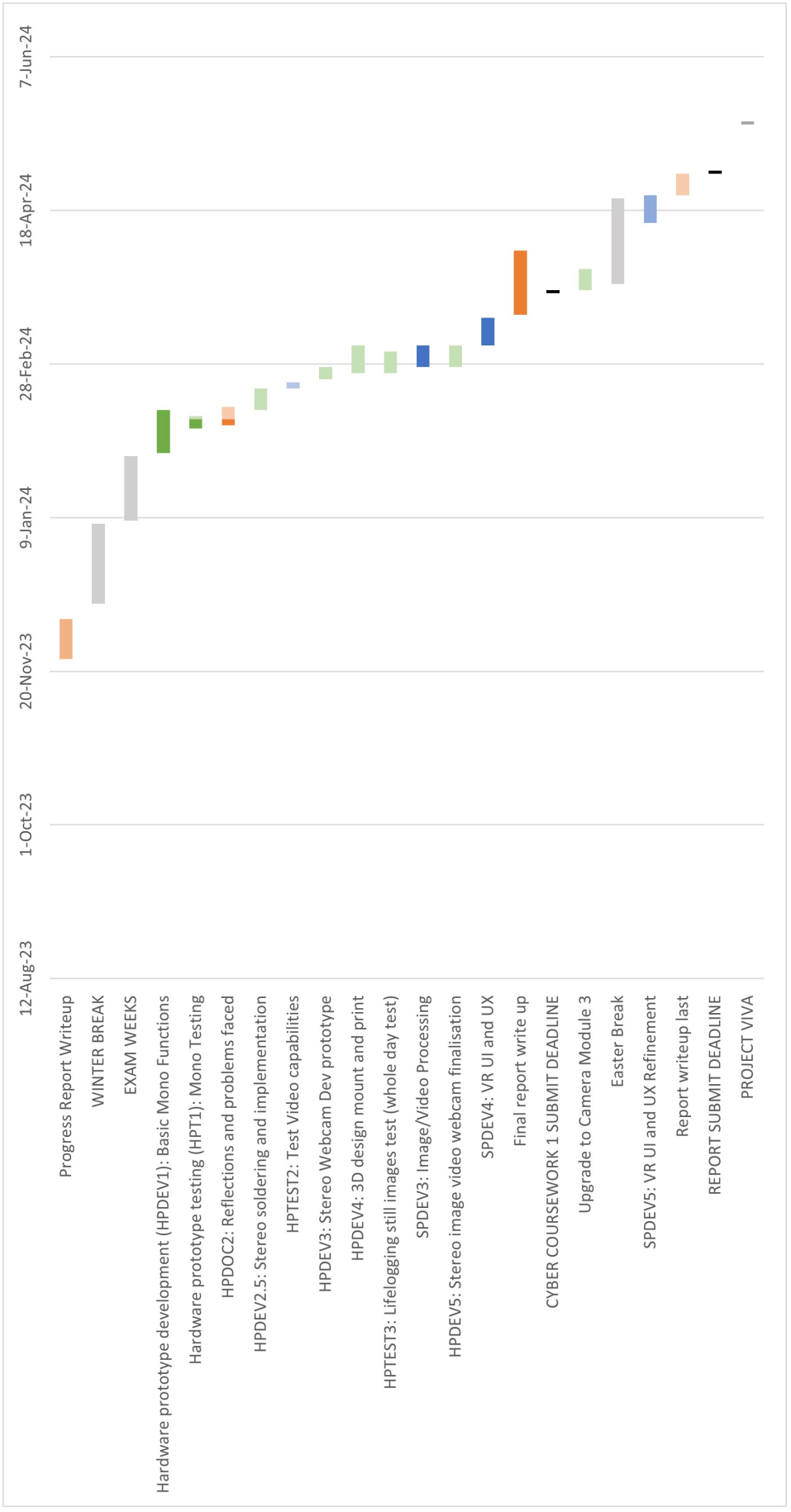

FIGURE 6.3: Combined Gantt Chart after Progress Report submission. Legend: Orange - Report or writing related task, Blue - Software related task, Green - Hardware related task. Softer color represents tasks over expected time or unexpected tasks compared to the Planned Gantt chart.

Positive Aspects:

- Evolving Project Management: Embracing an evolving project management approach proved beneficial, allowing for quicker initiation of work without being overly constrained by extensive planning phases. This flexibility enabled adaptability to changing requirements and prevented overemphasis on initial design specifications.
- Organizational Efficiency: Maintaining all project folders and files within a single, well-structured directory facilitated organization and streamlined access to essential resources. This approach enhanced efficiency during development and minimized time spent searching for relevant materials.
- Backup Systems: Utilizing both OneDrive and GitHub ensured reliable backup systems were in place, safeguarding project data and preventing loss in the event of unforeseen circumstances. This redundancy instilled confidence in the integrity and accessibility of project files.
- Documentation Practices: Maintaining a detailed logbook during work sessions proved invaluable for tracking progress, documenting findings, and identifying errors encountered. This practice facilitated report writing and provided a reference point for future troubleshooting.

Areas for Improvement:

- Gantt Chart Specificity: The level of specificity in the Gantt chart tasks hindered flexibility and daily tracking, making it challenging to adapt to changing priorities or unforeseen circumstances. Future iterations should consider employing a more generalized approach, focusing on weekly rather than daily task tracking.
- Task Delegation Tools: Overreliance on the Gantt chart for task delegation and tracking limited the ability to manage complex tasks effectively. Supplementing this tool with platforms like Trello for specifying and managing intricate tasks could enhance project coordination and collaboration.
- Component Scrutiny: In hindsight, more thorough scrutiny and research on project components before procurement would have been beneficial. Checking GitHub issue pages and community forums could have provided critical

insights, such as the incompatibility of certain components like the Raspberry Pi Pico and SPI camera (ArduCam OV5640), thus avoiding potential setbacks and wasted resources.

Reflecting on these experiences, future projects can benefit from a balanced approach to project management, leveraging flexibility, organizational practices, and thorough research to navigate challenges effectively and maximize success.

Chapter 7

Conclusion

7.1 Summary

Throughout this project, several key findings, achievements, and contributions have emerged, shedding light on the feasibility and potential of developing a low-cost stereo video camera system tailored for lifelogging and content creation.

Key Findings:

- The Raspberry Pi Pico and SPI-based camera configuration proved inadequate for high framerate video capture due to limitations in the serial peripheral interface bandwidth. As a result, transitioning to the more powerful Raspberry Pi 5, with its dual MIPI CSI camera ports, significantly improved performance and resolution capabilities.
- Leveraging versatile tools like FFMPEG for video processing, combined with Python automation, proved highly effective in streamlining the stereo processing pipeline.
- The integration of the MAX-Scene Classifier API facilitated efficient scene detection and organization within the VR software application, enhancing user experience and navigation.
- Godot 4 demonstrated exceptional capabilities for VR software development, particularly with its advanced XR toolkit and GDScript, offering a user-friendly platform for prototyping immersive experiences.

- The importance of 3D personal content in evoking strong emotions and enhancing user engagement was underscored, suggesting a need for greater focus and exploration in this area.

Achievements:

- All project objectives were successfully achieved, with some surpassing initial expectations, such as resolution capabilities exceeding 1080p.
- While hardware development presented challenges, particularly regarding compatibility and performance, the stereo processing pipeline and VR software components operated flawlessly, demonstrating the project's overall success.
- The development of modular scripts for the stereo processing pipeline, active involvement in the Godot XR community, and integration of API filtering within the VR application contributed significantly to project innovation and effectiveness.

Contributions:

- The project contributes to the validation of low-cost, off-the-shelf components for stereo video camera development, showcasing the potential for customization to suit various needs, such as hands-free lifelogging.
- Modular scripts for automation, collaboration within the Godot XR community, and integration of API functionality within the VR application serve as valuable resources for future projects and developers.
- By highlighting the importance of personalized stereo content and accessibility in VR experiences, the project fosters awareness and exploration in this emerging field.

7.2 Future Work

Looking ahead, several opportunities for future work and enhancements present themselves:

- Integration of a stereo microphone for improved spatial audio quality.
- Further design and mounting possibilities to accommodate various use cases, including a handheld version.
- Development of plug-and-play UI elements for the stereo processing pipeline to enhance accessibility for non-programmers.
- Exploration of containerization options, such as Docker images, to streamline deployment and usage.
- Implementation of eye tracking and hand tracking for enhanced user interaction and immersion.
- Integration of additional 3D control elements, such as interactive memory bookshelves, to enrich the user experience.

7.3 Finishing thoughts

In conclusion, this project serves as a catalyst for advancing the accessibility and innovation of personalized stereo content in virtual reality. By showcasing the capabilities of low-cost hardware and versatile software tools, it opens doors for further exploration and development in this exciting field. May this project inspire others to embark on their own journeys of discovery and creativity, leading to new heights of immersive storytelling and user engagement in the realm of VR.

Bibliography

[1] *Meta Quest 3: New Mixed Reality VR Headset*. [Online]. Available: https://www.meta.com/gb/quest/quest-3/.

[2] *Introducing Apple Vision Pro: Apple's first spatial computer*, Jun. 2023. [Online]. Available: https://www.apple.com/newsroom/2023/06/introducing-apple-vision-pro/.

[3] *Spectacles by Snap Inc. Spectacles 3*, 2023. [Online]. Available: https://www.spectacles.com/uk/shop/spectacles-3.

[4] *Kandao QooCam Ego 3D Camera*. [Online]. Available: https://www.kandaovr.com/qoocam-ego/.

[5] *Apple unveils iPhone 15 Pro and iPhone 15 Pro Max*, Sep. 2023. [Online]. Available: https://www.apple.com/newsroom/2023/09/apple-unveils-iphone-15-pro-and-iphone-15-pro-max/.

[6] M. Park, *3 keys to improving user retention in virtual reality*, Oct. 2017. [Online]. Available: https://venturebeat.com/games/3-keys-to-improving-user-retention-in-virtual-reality/.

[7] V. Weis, *3DOF, 6DOF, RoomScale VR, 360 Video and Everything In Between — Blog — Packet39*, 2018. [Online]. Available: https://packet39.com/blog/3dof-6dof-roomscale-vr-360-video-and-everything-in-between/.

[8] E. Callenbach, ": The Five C's of Cinematography: Motion Picture Filming Techniques Simplified . Joseph V. Mascelli.," *Film Quarterly*, vol. 19, no. 4, 1966, ISSN: 0015-1386. DOI: 10.1525/fq.1966.19.4.04a00320.

[9] N. Dodgson, "Autostereoscopic 3D displays," *Computer*, vol. 38, no. 8, pp. 31–36, Aug. 2005, ISSN: 0018-9162. DOI: 10.1109/MC.2005.252. [Online]. Available: http://ieeexplore.ieee.org/document/1492263/.

[10] C. Gurrin, A. F. Smeaton, and A. R. Doherty, "LifeLogging: Personal Big Data," *Foundations and Trends® in Information Retrieval*, vol. 8, no. 1, pp. 1–125, 2014, ISSN: 1554-0669. DOI: 10.1561/1500000033. [Online]. Available: http://www.nowpublishers.com/articles/foundations-and-trends-in-information-retrieval/INR-033.

[11] M. Dimiccoli and P. Radeva, "Visual Lifelogging in the Era of Outstanding Digitization," *Digital Presentation and Preservation of Cultural and Scientific Heritage*, vol. 5, pp. 59–64, Sep. 2015, ISSN: 2535-0366. DOI: 10.55630/dipp.2015.5.4. [Online]. Available: https://dipp.math.bas.bg/dipp/article/view/dipp.2015.5.4.

[12] E. Waisberg, J. Ong, M. Masalkhi, *et al.*, "Meta smart glasses—large language models and the future for assistive glasses for individuals with vision impairments," *Eye*, 2023, ISSN: 1476-5454. DOI: 10.1038/s41433-023-02842-z. [Online]. Available: https://doi.org/10.1038/s41433-023-02842-z.

[13] A. Orchard, M. O'Gorman, C. La Vecchia, and J. Lajoie, "Augmented Reality Smart Glasses in Focus: A User Group Report," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, New York, NY, USA: ACM, Apr. 2022, pp. 1–7, ISBN: 9781450391566. DOI: 10.1145/3491101.3503565. [Online]. Available: https://dl.acm.org/doi/10.1145/3491101.3503565.

[14] D. Marr and T. Poggio, "A computational theory of human stereo vision," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 204, no. 1156, pp. 301–328, May 1979, ISSN: 0080-4649. DOI: 10.1098/rspb.1979.0029. [Online]. Available: https://royalsocietypublishing.org/doi/10.1098/rspb.1979.0029.

[15] M. S. Banks, J. C. A. Read, R. S. Allison, and S. J. Watt, "Stereoscopy and the Human Visual System," *SMPTE Motion Imaging Journal*, vol. 121, no. 4, pp. 24–43, May 2012, ISSN: 1545-0279. DOI: 10.5594/j18173. [Online]. Available: https://ieeexplore.ieee.org/document/7308394/.

[16] K. Matzen, M. F. Cohen, B. Evans, J. Kopf, and R. Szeliski, "Low-cost 360 stereo photography and video capture," *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–12, Aug. 2017, ISSN: 0730-0301. DOI: 10.1145/3072959.3073645. [Online]. Available: https://dl.acm.org/doi/10.1145/3072959.3073645.

[17] R. Aggarwal, A. Vohra, and A. M. Namboodiri, "Panoramic Stereo Videos with a Single Camera," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2016-December, IEEE, Jun. 2016, pp. 3755–3763, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.408. [Online]. Available: http://ieeexplore.ieee.org/document/7780777/.

[18] Umar Shakir, *The iPhone 15 Pro is getting spatial video capture in iOS 17.2*, Nov. 2023. [Online]. Available: https://www.theverge.com/2023/11/9/23954310/apple-iphone-15-ios-17-beta-spatial-video-vision-pro.

[19] M. Mühlhausen, M. Kappel, M. Kassubeck, *et al.*, "Immersive Free-Viewpoint Panorama Rendering from Omnidirectional Stereo Video," *Computer Graphics Forum*, vol. 42, no. 6, 2023, ISSN: 14678659. DOI: 10.1111/cgf.14796.

[20] R. Pagés, K. Amplianitis, D. Monaghan, J. Ondřej, and A. Smolić, "Affordable content creation for free-viewpoint video and VR/AR applications," *Journal of Visual Communication and Image Representation*, vol. 53, 2018, ISSN: 10959076. DOI: 10.1016/j.jvcir.2018.03.012.

[21] A. Serrano, I. Kim, Z. Chen, *et al.*, "Motion parallax for 360° RGBD video," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 5, 2019, ISSN: 19410506. DOI: 10.1109/TVCG.2019.2898757.

[22] J. Armstrong, *"The Entire History of You"*, 2011.

[23] H. Strasburger, *Seven Myths on Crowding and Peripheral Vision*, 2020. DOI: 10.1177/2041669520913052.

[24] A. Duane, B. P. Jónsson, and C. Gurrin, "VRLE: Lifelog Interaction Prototype in Virtual Reality: Lifelog Search Challenge at ACM ICMR 2020," in *LSC 2020 - Proceedings of the 3rd Annual Workshop on the Lifelog Search Challenge*, 2020. DOI: 10.1145/3379172.3391716.

[25] H. Chen, R. Shi, D. Monteiro, N. Baghaei, and H. N. Liang, "VRCockpit: Mitigating Simulator Sickness in VR Games Using Multiple Egocentric 2D View Frames," in *IEEE Conference on Computatonal Intelligence and Games, CIG*, vol. 2022-August, 2022. DOI: 10.1109/CoG51982.2022.9893678.

[26] S. H. Park and G. C. Lee, "Full-immersion virtual reality: Adverse effects related to static balance," *Neuroscience Letters*, vol. 733, 2020, ISSN: 18727972. DOI: 10.1016/j.neulet.2020.134974.

[27] K. Lim, J. Lee, K. Won, N. Kala, and T. Lee, "A novel method for VR sickness reduction based on dynamic field of view processing," *Virtual Reality*, vol. 25, no. 2, 2021, ISSN: 14349957. DOI: 10.1007/s10055-020-00457-3.

[28] A. S. Fernandes and S. K. Feiner, "Combating VR sickness through subtle dynamic field-of-view modification," in *2016 IEEE Symposium on 3D User Interfaces, 3DUI 2016 - Proceedings*, 2016. DOI: 10.1109/3DUI.2016.7460053.

[29] O. M. Zarka and Z. J. Shah, "Virtual Reality Cinema: A Study," *IJRAR-International Journal of Research and Analytical Reviews*, vol. 3, no. 2, 2016.

[30] *FFMPEG A complete, cross-platform solution to record, convert and stream audio and video.* [Online]. Available: https://ffmpeg.org/.

[31] IBM, *IBM Code Model Asset Exchange: Scene Classifier*, 2018. [Online]. Available: https://github.com/IBM/MAX-Scene-Classifier.

[32] *Godot 4.2 Game Engine Docs.* [Online]. Available: https://docs.godotengine.org/en/4.2/.

[33] Malcolmnixon, *Malcolmnixon / godot-stereo-video Github Repository*, Nov. 2023. [Online]. Available: https://github.com/Malcolmnixon/godot-stereo-video.

Risk Assessment

Risk event	Likelihood (1-5)	Impact (1-5)	Risk Exposure (1-25)	Mitigative Action	Alternative Action	Actions taken
Fitting electronics into a spectacle frame clip-on enclosure is too technically complex and difficult.	4	4	16	Conduct thorough prototyping and testing of the case design early in the project to identify and address integration issues.	Explore alternative enclosure designs or materials that are more manageable in terms of complexity.	Instead of a glass frame design, a head-mounted design is chosen
Processing power and onboard memory of Raspberry Pi Pico might not be enough for the complexity of this project.	4	5	20	Optimize code and data storage to work efficiently within the constraints of the Raspberry Pi Pico boards.	Consider upgrading to more powerful microcontrollers if necessary.	
Powering three Raspberry Pi Pico devices with their electronics from a single supply may lead to suboptimal or non-functional performance.	5	3	15	Calculate power consumption and explore energy-efficient solutions. Implement features such as power management and hot-swappable batteries to address power challenges.	Use separate power supplies for each Raspberry Pi Pico to ensure stable and independent power sources.	

Table 1: Risk Management Table (Part 1)

Risk event	Likelihood (1-5)	Impact (1-5)	Risk Exposure (1-25)	Mitigative Action	Alternative Action	Actions Taken
Developing VR-specific lifelogging software with features like immersive user interface elements, scene/object detection, and metadata auto-tagging may introduce technical complexities, potentially causing project delays.	5	2	10	Simplifying the software scope, such as omitting scene detection and reducing metadata features, can help mitigate technical complexities and minimize project delays.	Utilize pre-existing VR software or libraries to simplify the software development process while still achieving project goals.	
Implementing stereo content within a VR game engine may prove technically challenging and time-consuming.	5	5	25	Plan for potential difficulties in stereo implementation and be prepared to use existing applications for viewing SBS content as an alternative.	Explore VR game engine plugins or assets specifically designed for SBS video playback, simplifying the integration and reducing technical complexities.	Consulting with GodotXRTools developer results in a working stereo video player prototype.
NEW: Components breakdown or not working perfectly due to technical error or manufacturing defects	5	5	25	Be careful when testing components using a multimeter, especially when turned on.	Have enough budget left to buy spare components, or use other available components from the university.	

Table 2: Risk Management Table (Part 2)

Onboard Algorithm Code

run_me.py:

```
import subprocess
import time
from datetime import datetime
import os

use_webcam = False  # Set this to True if you want to use a USB webcam

# Create directories if they don't exist
home_dir = "/home/chronohax/"
video_dir = home_dir + "videos"
image_dir = home_dir + "images"
audio_dir = home_dir + "audio"
vid0_dir = os.path.join(video_dir, "vid0")
vid2_dir = os.path.join(video_dir, "vid2")
img0_dir = os.path.join(image_dir, "img0")
img2_dir = os.path.join(image_dir, "img2")
for dir_path in [video_dir, image_dir, audio_dir, vid0_dir, vid2_dir,
                 img0_dir, img2_dir]:
    os.makedirs(dir_path, exist_ok=True)

while True:
    # Video recording every 5 minutes
    print("Starting automated recording for both cameras...")
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

    # Define filenames for each camera
    filename_vid0 = os.path.join(vid0_dir, f"video_vid0_{timestamp}")
    filename_vid2 = os.path.join(vid2_dir, f"video_vid2_{timestamp}")
    filename_audio = os.path.join(audio_dir, f"audio_{timestamp}.wav")

    if use_webcam:
        # Define the FFmpeg commands for each USB webcam with optimized settings
        command_vid0 = f"ffmpeg -f v4l2 -framerate 24 -ss 1 -video_size 1280x720
        -input_format mjpeg -i /dev/video0 -preset ultrafast -pix_fmt yuv420p
        -t 30 {filename_vid0}.mkv"
        command_vid2 = f"ffmpeg -f v4l2 -framerate 24 -ss 1 -video_size 1280x720
        -input_format mjpeg -i /dev/video2 -preset ultrafast -pix_fmt yuv420p
        -t 30 {filename_vid2}.mkv"
    else:
        # Define the libcamera/rpicam commands for the Raspberry Pi camera
        command_vid0 = f"rpicam-vid --camera 0 --width 1920 --height 1080
        --framerate 30 -t 30000 --codec mjpeg -o  {filename_vid0}.mjpeg"
```

```
        command_vid2 = f"rpicam-vid --camera 1 --width 1920 --height 1080
        --framerate 30 -t 30000 --codec mjpeg -o  {filename_vid2}.mjpeg"

    # Define the arecord command to record audio from the USB microphone
    command_audio = f"arecord -d 30 -f dat {filename_audio}"

    # Run the commands using subprocess.Popen() to start each process
    process_cam0 = subprocess.Popen(command_vid0, shell=True)
    process_cam2 = subprocess.Popen(command_vid2, shell=True)
    process_audio = subprocess.Popen(command_audio, shell=True)
    print("Automated recording started for both cameras...")

    # Wait for both processes to finish
    process_cam0.wait()
    process_cam2.wait()
    print("Automated video recording completed for both cameras...")

    # Image capture every 10 seconds
    for j in range(2):  # 30 iterations = 5 minutes of images every 10 seconds
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

        # Define filenames for each camera
        filename_img0 = os.path.join(img0_dir, f"image_img0_{timestamp}.jpg")
        filename_img2 = os.path.join(img2_dir, f"image_img2_{timestamp}.jpg")

        if use_webcam:
            # Define the FFmpeg commands for each USB webcam
            command_img0 = f"ffmpeg -f v4l2 -framerate 30 -ss 1 -video_size 1280x720
            -input_format mjpeg -i /dev/video0 -vframes 1 {filename_img0}"
            command_img2 = f"ffmpeg -f v4l2 -framerate 30 -ss 1 -video_size 1280x720
            -input_format mjpeg -i /dev/video2 -vframes 1 {filename_img2}"
        else:
            # Define the libcamera commands for the Raspberry Pi camera
            command_img0 = f"rpicam-jpeg --camera 0 --width 4608 --height 2592
            --quality 75 -o {filename_img0}"
            command_img2 = f"rpicam-jpeg --camera 1 --width 4608 --height 2592
            --quality 75 -o {filename_img2}"

        # Run the commands using subprocess.Popen() to capture images
        process_img0 = subprocess.Popen(command_img0, shell=True)
        process_img2 = subprocess.Popen(command_img2, shell=True)

        # Wait for both image capture processes to finish
        process_img0.wait()
        process_img2.wait()
        print("Automated image capture completed for both cameras...")

        time.sleep(10)  # Wait for 10 seconds before capturing the next set of images

    print("5 minutes completed. Restarting the automated recording process...")
```

Stereo Processing Pipeline Codes

Full stitching code, **2run_stitch.py**:

```
import os
import subprocess
from datetime import datetime
import piexif
from PIL import Image

samples_path = "input" # for rasp pi camera, change the left/right

def stitch_videos(output_dir):
    # Create the output directory if it doesn't exist
    output_video_dir = os.path.join(output_dir, "videos")
    if not os.path.exists(output_video_dir):
        os.makedirs(output_video_dir)

    # Iterate over the video files in the vid0 and vid2 subdirectories
    vid0_dir = os.path.join(samples_path, "videos", "vid0")
    vid2_dir = os.path.join(samples_path, "videos", "vid2")
    for filename in os.listdir(vid0_dir):
        if filename.endswith(".mkv"):
            # Get the input video paths
            vid0_path = os.path.join(vid0_dir, filename)
            vid2_path = os.path.join(vid2_dir, filename.replace('_vid0', '_vid2'))

            # Check if the input videos exist
            if not os.path.exists(vid0_path) or not os.path.exists(vid2_path):
                print(f"Input video files not found for {filename}")
                continue

            # Define the output video path
            output_path = os.path.join(output_video_dir, //
            f"stitched_{filename.replace('_vid0', '').replace('_vid2', '')}")

            # Define the ffmpeg command to stitch the videos side by side
            ffmpeg_command = [
                "ffmpeg",
                "-i", vid2_path, #raspi cam use 2,0 instead of 0,2
                "-i", vid0_path,
                "-filter_complex", "[0:v][1:v]hstack=inputs=2",
                "-c:v", "libtheora",  # Use Theora codec for .ogv
                "-qscale:v", "7",  # Set quality scale for Theora codec
                "-an",  # Disable audio
                # Copy metadata from the first input file
```

```
                "-map_metadata", "0:g:0",
                "-y",  # Overwrite output file without asking
                # Change output file extension to .ogv
                output_path.replace(".mkv", ".ogv")
            ]

            # Run ffmpeg command
            try:
                subprocess.run(ffmpeg_command, check=True)
                print(f"Videos stitched successfully: {output_path}")
            except subprocess.CalledProcessError as e:
                print(f"Error stitching videos: {e}")

def stitch_images(output_dir):
    # Create the output directory if it doesn't exist
    output_image_dir = os.path.join(output_dir, "images")
    if not os.path.exists(output_image_dir):
        os.makedirs(output_image_dir)

    # Iterate over the image files in the img0 and img2 subdirectories
    img0_dir = os.path.join(samples_path, "images", "img0")
    img2_dir = os.path.join(samples_path, "images", "img2")
    for filename in os.listdir(img0_dir):
        if filename.endswith(".jpg"):
            # Get the input image paths
            img0_path = os.path.join(img0_dir, filename)
            img2_path = os.path.join(img2_dir, filename.replace('_img0', '_img2'))

            # Check if the input images exist
            if not os.path.exists(img0_path) or not os.path.exists(img2_path):
                print(f"Input image files not found for {filename}")
                continue

            # Define the output image path
            output_path = os.path.join(output_image_dir,
            f"stitched_{filename.replace('_img0', '').replace('_img2', '')}")

            # Load input images
            img0 = Image.open(img0_path)
            img2 = Image.open(img2_path)

            # Get the width and height of the input images
            width1, height1 = img0.size
            width2, height2 = img2.size

            # Create a new blank image with the combined width
            result_width = width1 + width2
            result_height = max(height1, height2)
            result = Image.new('RGB', (result_width, result_height))

            # Paste the input images side by side
            result.paste(img2, (0, 0)) #raspi cam use 2,0 instead of 0,2
            result.paste(img0, (width1, 0))

            # Load metadata from the first input image
            try:
                exif_data = piexif.load(img0_path)
            except FileNotFoundError:
```

```
                print(f"Image file not found: {img0_path}")
                continue
            except piexif.InvalidImageDataError:
                print(f"Invalid image data: {img0_path}")
                continue

            # Save the stitched image with the copied metadata
            try:
                exif_bytes = piexif.dump(exif_data)
                result.save(output_path, exif=exif_bytes)
                print(f"Images stitched successfully: {output_path}")
            except Exception as e:
                print(f"Error stitching images: {e}")

if __name__ == "__main__":
    # Path to output directory
    output_dir = samples_path + "/output"

    # Stitch videos
    stitch_videos(output_dir)

    # Stitch images
    stitch_images(output_dir)
```

Full mpeg to mkv conversion code, **0.5run-convert-mjpeg-to-mkv.py**:

```
import os
import subprocess

samples_path = "input/"

def convert_mjpeg_to_mkv(input_dir):
    # Iterate over the video files in the input directory
    for filename in os.listdir(input_dir):
        if filename.endswith(".mjpeg"):
            # Get the input video path
            input_path = os.path.join(input_dir, filename)

            # Check if the input video exists
            if not os.path.exists(input_path):
                print(f"Input video file not found: {input_path}")
                continue

            # Define the output video path
            output_path = os.path.join(input_dir, //
            f"{os.path.splitext(filename)[0]}.mkv")

            # Define the ffmpeg command to convert the video
            ffmpeg_command = [
                "ffmpeg",
                "-i", input_path,  # Input video file
                "-c:v", "copy",  # Copy video stream without re-encoding
                "-r", "30",  # Set the output framerate to 30 fps
                "-f", "matroska",  # Output format: Matroska (MKV)
                "-y",  # Overwrite output file without asking
                output_path
            ]
```

```
            # Run ffmpeg command
            try:
                subprocess.run(ffmpeg_command, check=True)
                print(f"Video converted successfully: {output_path}")
                # Remove the original .mjpeg file
                os.remove(input_path)
                print(f"Removed original file: {input_path}")
            except subprocess.CalledProcessError as e:
                print(f"Error converting video: {e}")

# Specify the paths to the vid0 and vid2 directories
vid0_dir = samples_path + "videos/vid0"
vid2_dir = samples_path + "videos/vid2"

# Call the conversion function for both directories
convert_mjpeg_to_mkv(vid0_dir)
convert_mjpeg_to_mkv(vid2_dir)
```

Rotate 180, **0.75run-flip-vertical-all.py**:

```
import os
import subprocess

samples_path = "input/"

def rotate_files(input_dir):
    # Iterate over the files in the input directory
    for filename in os.listdir(input_dir):
        input_path = os.path.join(input_dir, filename)
        if filename.endswith(".mkv"):  # Check if the file is a video
            # Define the output video path
            output_path = os.path.join(input_dir, //
            f"{os.path.splitext(filename)[0]}_rotated.mkv")

            # Define the ffmpeg command to rotate the video 180 degrees clockwise
            ffmpeg_command = [
                "ffmpeg",
                "-i", input_path,  # Input video file
                "-vf", "transpose=2,transpose=2",  # Rotate 180 degrees clockwise
                "-c:a", "copy",  # Copy audio stream without re-encoding
                "-y",  # Overwrite output file without asking
                output_path
            ]

            # Run ffmpeg command
            try:
                subprocess.run(ffmpeg_command, check=True)
                print(f"Video rotated successfully: {output_path}")
                # Remove the original video file
                os.remove(input_path)
                print(f"Removed original file: {input_path}")
                # Rename the rotated video to replace the original
                os.rename(output_path, os.path.join(input_dir, //
                f"{os.path.splitext(filename)[0]}.mkv"))
                print(f"Renamed rotated file: {output_path} to {input_path}")
            except subprocess.CalledProcessError as e:
```

```
                print(f"Error rotating video: {e}")

        elif filename.endswith(".jpg"):  # Check if the file is an image
            # Define the output image path
            output_path = os.path.join(input_dir, //
            f"{os.path.splitext(filename)[0]}.jpg")

            # Define the ffmpeg command to rotate the image 180 degrees clockwise
            ffmpeg_command = [
                "ffmpeg",
                "-i", input_path,  # Input image file
                "-vf", "transpose=2,transpose=2",  # Rotate 180 degrees clockwise
                "-y",  # Overwrite output file without asking
                output_path
            ]

            # Run ffmpeg command
            try:
                subprocess.run(ffmpeg_command, check=True)
                print(f"Image rotated successfully: {output_path}")
            except subprocess.CalledProcessError as e:
                print(f"Error rotating image: {e}")

# Specify the paths to the vid0 and vid2 directories
vid0_dir = samples_path + "videos/vid0"
vid2_dir = samples_path + "videos/vid2"
img0_dir = samples_path + "images/img0"
img2_dir = samples_path + "images/img2"

# Call the rotation function for all directories
rotate_files(vid0_dir)
rotate_files(vid2_dir)
rotate_files(img0_dir)
rotate_files(img2_dir)
```

.bat script to automatically run all required scripts sequentially, **run-all.bat**:

```
@echo off

echo Running convert mjpeg to mkv script...
python 0.5run-convert-mjpeg-to-mkv.py

:wait_convert_mjpeg_to_mkv
tasklist /fi "imagename eq python.exe" /fo csv 2>NUL | //
find /i "0.5run-convert-mjpeg-to-mkv.py" >NUL
if %ERRORLEVEL% equ 0 goto wait_convert_mjpeg_to_mkv

echo Running rotate all 180 script...
python 0.75run-flip-vertical-all.py

:wait_rotation_all_180
tasklist /fi "imagename eq python.exe" /fo csv 2>NUL | //
find /i "0.75run-flip-vertical-all.py" >NUL
if %ERRORLEVEL% equ 0 goto wait_rotation_all_180

echo Running metadata tagging script...
python 1run-metadata-tagging.py
```

```
:wait_metadata_tagging
tasklist /fi "imagename eq python.exe" /fo csv 2>NUL | //
find /i "1run-metadata-tagging.py" >NUL
if %ERRORLEVEL% equ 0 goto wait_metadata_tagging

echo Running stitching script...
python 2run-stitch.py

:wait_stitch
tasklist /fi "imagename eq python.exe" /fo csv 2>NUL | //
find /i "2run-stitch.py" >NUL
if %ERRORLEVEL% equ 0 goto wait_stitch

echo Running json copy script...
python 3run-copy-json.py

:wait_copy_json
tasklist /fi "imagename eq python.exe" /fo csv 2>NUL | //
find /i "3run-copy-json.py" >NUL
if %ERRORLEVEL% equ 0 goto wait_copy_json

echo Running analyse jsons script...
python 4run-analyse-jsons.py

:wait_analyse_jsons
tasklist /fi "imagename eq python.exe" /fo csv 2>NUL | //
find /i "4run-analyse-jsons.py" >NUL
if %ERRORLEVEL% equ 0 goto wait_analyse_jsons

echo Done!
pause
```

Shaders Code

The following is the .gdshaders code used for the SBS stereo effect:

```
shader_type spatial;
render_mode unshaded;

uniform sampler2D movie : source_color;

void vertex() {
  UV = vec2(UV.x * 0.5, UV.y);
  if (VIEW_INDEX == VIEW_RIGHT) {
    UV.x += 0.5;
  }
}

void fragment() {
  ALBEDO = texture(movie, UV).rgb;
}
```

More shaders options can be found in Godot Shaders documentation.[1]

[1] Link: https://docs.godotengine.org/en/stable/tutorials/shaders/shader_reference/spatial_shader.html

Design Archive Files

As previously mentioned in Chapter 6, the folders are named according to its designated roles:

Sorted by Name.

Files:

- Risk Assessment v2.pdf.
- Purchasing Card Request Form MASTER COPY.doc for hardware purchase.
- Project Brief - mhby1g21.pdf
- Gantt Chart.xlsx containing all the related Gantt charts and task tracking,
- Design Document.docx contains initial design feasibility study.

Folders:

- **Software_Processing**
  - run-all.bat is the script that will runs all relevant scripts sequentially
  - Logbook-GANYU.docx is the logbook for stereo processing pipeline work.
  - 4run-analyse-jsons.py is used to generate json prediction analysis.
  - 3run-copy-json.py is used to copy predicted response from img0 folder to output/images folder and the same for videos.
  - 2run-stitch.py is used to run the stitching process
  - 1run-metadata-tagging.py is used to run the prediction api response.
  - 0.75run-flip-vertical-all.py is used to rotate everything 180 when using final design.

  - 0.5run-convert-mjpeg-to-mkv.py is used to convert captured mjpeg video to mkv for processing.
  - Year2024 is the designated folder directory to work with VR software UI.
  - input folder is where the images, videos and audios folder and files should be put for processing.

- **Software_Game**
  - Logbook-Ganyu.docx is the logbook for software/game development work.ss
  - 2D file browser and viewer fodler is as the name suggest, the 2D UI app version.
  - VR Lifelog Data Viewer folder is the final VR software revision.
    The files in each folder is self-explanatory, but the most notable is the .exe and .apk inside the builds folder, which is the exported app ready to run with any OpenXR runtime for VR app.
- **Software_Algorithm** which contains only the logbook and the run_me.py script which is what running on the hardware on final version (Raspberry Pi 5)
- Hardware_mount contains .stl files for 3d printing case and enclosure for mounting and stereo alignment.
  Onshape public link sharing for USB Webcam design and Raspberry Pi Camera Module 3
- Hardware_deprecated contains the abandoned and failed initial design using Raspberry Pi Pico progress and work.
- FileZilla-3.66.5 contains FileZilla executable used to transfer files from and to Raspberry Pi 5.

Due to size and storage restrictions, most images and videos captured is only available on OneDrive.

Both OneDrive and Github link is public as well to see more footage and works not included in design archive to limit size.

Project Brief

Student Name: Muhammad Hazimi Bin Yusri (mhby1g21@soton.ac.uk)
Supervisor Name: Dr. Tom Blount
Title: Open-Source Stereo Video Camera System and Software Implementation for Virtual Reality (VR) Lifelogging and Content Creation

Problem Statement

The current landscape of VR camera systems, offered by major companies, presents significant challenges for accessibility due to their proprietary nature and inflated cost. This exclusivity hinders innovation in both hardware and software within the VR industry, limiting its growth potential.

Project Goals

- Develop an Open-Source, Low-Cost, and Modular Hardware System: The primary objective is to design and build an open-source, affordable, and modular hardware system tailored for VR content creation. This system will focus on lifelogging by incorporating a snap/clip-on design for spectacle frames. This innovative approach aims to lead the way in expanding the scope and methods of VR lifelogging technology. This system would also ideally be robust enough to withstand normal use handling and minimal water resistance.
- Create a Prototype Lifelogging VR Software: In conjunction with the hardware, develop prototype VR software for viewing the recorded content on a

desktop PCVR platform. The software will ideally include metadata auto-tagging capabilities using scene/object detection, enhancing content indexing and search efficiency. This also takes advantage of VR freedom of movement so virtual space can be used to show longer timeline which is important for lifelogging with vast amount of footage. VR environment can also be influenced and changed depending on content viewed.

Scope

Hardware: The project encompasses the creation of a hardware system consisting of two small, lightweight cameras and a microphone to be attached to the sides of spectacles. This system will include robust wiring to facilitate connections to necessary components, such as power management/battery solutions and the microcontroller unit (MCU), central processing unit (CPU), or motherboard.
Software: The software component of this project will involve the development of a side-by-side (SBS) video player and search software, leveraging sophisticated features built using a game engine. This allows for more customization and freedom such as adding interactivity and weather simulation based on metadata information.

Features Priority

1. Develop an Open-Source, Low-Cost, and Modular Hardware System: This is the top priority, focusing on creating a hardware system that is open-source, affordable, and modular, enabling accessibility and innovation.

2. Create a Prototype Lifelogging VR Software: The development of prototype VR software is essential to complement the hardware and provide users with a means to view and interact with the recorded content efficiently.

3. High-Quality SBS Video Capture: Ensuring high-quality video capture including stereo audio is crucial for an immersive and non-nauseating VR lifelogging experience.

4. Convenient battery system with charging support.

5. Additional sensors that can influence VR environment such as heart monitor.

This project aims to democratize VR content creation by providing an open-source, affordable, and versatile solution accessible to a wide range of users within the VR community. It strives to contribute to the advancement of VR content creation technology while fostering innovation and inclusivity in the field.