

Master Thesis

Prototype Development of a Handheld Speed Camera

for the attainment of the academic degree

Master of Mechanical Engineering

submitted by

Muhammad Haziq Bin Mohd Sabtu



Technische Hochschule Brandenburg

Department of Technology

Studiengang Maschinenbau

Reviewer 1: Prof. Dr.-Ing. Christian Oertel

Reviewer 2: Prof. Dr.-Ing. Peter M. Flassig

Brandenburg, October 26, 2023

I, MUHAMMAD HAZIQ BIN MOHD SABTU, student in the Mechanical Engineering program of the Brandenburg University of Applied Sciences, declare in oath that this thesis has been written by myself and has not been written with other than the other than the indicated aids. The research presented in this thesis, including all references and sources, is entirely my own work, unless otherwise stated.

I further declare that the work presented in this thesis has not been previously included in any other work, academic or otherwise, for which I have received a degree or diploma. Any assistance received during the course of this research has been acknowledged, and all contributions by individuals or sources have been appropriately cited.

MUHAMMAD HAZIQ BIN MOHD SABTU

Abstract

This research presents the development of a handheld speed measurement device offering a cost-effective alternative to conventional speed pistols. A prototype has been engineered by integrating affordable computational components and high-quality cameras and employing 3D printing technology. The design process was guided by VDI Guideline 2221. Multiple variants were generated and evaluated against their technical and economic rating to find the optimal design. The selected variant was then produced by additive manufacturing and tested for functionality. During the software development, the focus was on the implementation of a user-friendly interface incorporating the MVC architectural pattern. The Waterfall Model was used as the software development process model.

Keywords: Speed Mesurement, 3D Printing, VDI Guideline 2221, MVC, Waterfall Model

Kurzfassung

Die vorliegende Arbeit befasst sich mit der Entwicklung eines mobilen Geschwindigkeitsmessgeräts, das eine kostengünstige Alternative zu herkömmlichen Geschwindigkeitspistolen darstellt. Ein Prototyp wurde durch die Integration von kostengünstigen Rechenkomponenten und hochwertigen Kameras sowie durch den Einsatz von 3D-Drucktechnologie entwickelt. Der Entwurfsprozess orientierte sich an der VDI-Richtlinie 2221. Es wurden mehrere Varianten erstellt und anhand ihrer technischen und wirtschaftlichen Bewertung bewertet, um das optimale Design zu finden. Die ausgewählte Variante wurde dann mittels additiver Fertigung hergestellt und auf ihre Funktionalität getestet. Bei der Softwareentwicklung lag der Schwerpunkt auf der Implementierung einer benutzerfreundlichen Oberfläche, die das MVC-Architekturmuster verwendet. Das Wasserfallmodell wurde als Softwareentwicklungsprozessmodell verwendet.

Schlüsselwörter: Geschwindigkeitsmessung, 3D-Druck, VDI-Richtlinie 2221, MVC, Wasserfallmodell

Contents

List of Abbreviations	1
1 Introduction	2
1.1 Motivation	2
1.2 Problem Statement	2
1.3 Previous Work	3
1.4 Research Objectives	3
2 State of the Art - Speed Pistol	5
I Prototype Development	7
3 Literature Review	8
3.1 Methodic Product Development	8
3.2 Fused Deposition Modeling	9
4 Planning and Task Clarification	14
4.1 Establishing the Prototype's Requirements	15
4.2 Identifying the Prototype's Requirements	16
4.3 Requirement List	22
5 Conceptual Design	24
5.1 Abstraction	24
5.2 Function Structures	26
5.3 Developing Working Principles	29
5.4 Combination of Working Principles	31
5.5 Firming Up into Principle Solution Variants	33
5.6 Filtering of Solution Variants	41

CONTENTS

6 Embodiment Design	44
6.1 Basic Rules of Embodiment Design	44
6.2 Guideline of Embodiment Design	47
6.3 Preliminary Design	49
6.4 Evaluation with VDI 2225	61
7 Detail Design	69
8 Printing and Assembly	76
8.1 Printing	76
8.2 Assembly	77
8.3 Final Product	83
9 Conclusion	86
 II GUI Development	 88
10 Literature Review	89
10.1 Waterfall Model	89
10.2 Model-View-Controller	90
11 Requirement Analysis	92
11.1 Project Requirements	94
12 Design	96
12.1 Software Architecture	96
12.2 User Interface Design	100
13 Coding	108
13.1 View Implementation	108
13.2 Model Implementation	118
13.3 Controller Implementation	119
13.4 Algorithm Improvement	123
14 Testing and Maintenance	129
14.1 Unit Testing	129
14.2 Maintenance	131

CONTENTS

15 Conclusions and Future Work	133
15.1 Conclusions	133
15.2 Future Work	134
III Indexes and Appendix	138
List of Figures	139
List of Tables	142
Bibliography	144
A Appendix	154
A.1 Sketches of Working Principles	154
A.2 CAD Drawings	160
A.3 Technical Specifications	167
A.4 Cost Calculation	178
A.5 Evaluation Data	197
A.6 Documentation	205
A.7 C++ Unit Tests for Bank Account Management	205
A.8 User Manual	208
A.9 Developer Manual	243

List of Abbreviations

3D	Three-Dimensional
ABS	Acrylonitrile Butadiene Styrene
CPU	Central Processing Unit
FDM	Fused Deposition Modeling
G-Code	Geometric Code
GPIO	General Purpose Input Output
GUI	Graphical User Interface
HDMI	High Definition Multimedia Interface
ID	Identification
LAN	Local Area Network
LCD	Liquid Crystal Display
LIDAR	Light Detection and Ranging
MVC	Model View Controller
PC	Personal Computer
PET	Polyethylenterephthalat
PETG	Polyethylenterephthalat Glycol
PLA	Polylactic Acid
POS	Point of Service
RADAR	Radio Detection and Ranging
ROI	Region of Interest
UI	User Interface
USB	Universal Serial Bus
VDI	Verein Deutscher Ingenieure

1 Introduction

1.1 Motivation

Excessive speeding poses a significant threat to the safety of people on our roads. It impacts those directly involved in accidents, their families, and their communities. Over four years, from 2019 to 2022, Germany experienced an average of nearly 38,000 accidents annually due to speeding [Sta23]. These accidents can have long-lasting consequences, inflicting physical and emotional harm. Countless individuals are injured or tragically losing their lives, profoundly impacting their loved ones. It is crucial to address this issue, and the goal of this thesis is to develop an innovative speed gun system to aid in the reduction of excessive speeding.

1.2 Problem Statement

Law enforcement agencies and road safety organizations often find the current speed gun technologies available in the market too expensive for widespread adoption. These technologies can cost around 2000 €, which creates a financial barrier preventing effective speed monitoring in many regions. This thesis aims to explore alternative speed gun systems that use computer vision to address this challenge. By using this technology, we aim to create a more affordable and robust solution for speed monitoring, which will address the financial constraints of traditional speed guns and open up new possibilities for innovation and customization in speed monitoring technology. Ultimately, this will enhance road safety and reduce accidents caused by excessive speed.

1.3 Previous Work

Previous research [BMS23] has delved into various computer vision methods to tackle the complex challenge of speed monitoring. One remarkable innovation is the image alignment algorithm, which employs feature detection techniques to fix unintended movement during video recording. This algorithm enhances the stability of recorded footage and ensures accurate speed assessment.

Another technique explored is optical flow analysis, which is a method that tracks objects in a video sequence by calculating the velocity of points within the images and estimating where those points could be in the following sequence. This technique is computationally efficient, making it a practical and cost-effective solution for real-time speed monitoring.

Lastly, a method to measure object distance using the pinhole camera model was also explored. This model leverages the pinhole camera's geometry to calculate the distance of objects from the camera's vantage point. Integrating this model into speed monitoring provides a streamlined and economically viable method for assessing speeds with high precision.

After conducting thorough testing, it has been established that these techniques exhibit a high level of effectiveness in monitoring speed, achieving an impressive accuracy rate of 94 %. This outcome signifies their viability for advancing to the next stage of development and potential incorporation into a prototype speed gun system.

1.4 Research Objectives

Our research aims to develop an alternative to current implementation of speed gun by utilizing computer vision technology. The work will consist of two parts. In the first part, we will design the physical structure of the prototype using the VDI 2221 methodology. The second part will focus on software development, creating an easy-to-use interface that leverages computer vision for real-time

Introduction

speed assessment and data analysis. The goal is to create a state-of-the-art speed gun that addresses speeding concerns and sets a new road safety technology standard.

2 State of the Art - Speed Pistol

Law enforcement agencies worldwide utilize speed guns, also known as radar guns or speed pistols, as crucial tools to combat speeding. These devices play a pivotal role in maintaining road safety by accurately measuring the velocity of vehicles on the road [Hul20].

The current implementation of speed pistol usually utilize either the LIDAR (Light Detection and Ranging) or RADAR (Radio Detection and Ranging) technology [Rad23] [Fly23] [Sig23] [Tec23]. These technologies serve as the cornerstone of modern speed measurement devices, allowing law enforcement officers to gauge the speed of vehicles in real-time accurately.

LIDAR technology operates on emitting laser pulses towards a target vehicle and measuring the time it takes for the light to bounce back [Fly23]. The LIDAR device can precisely calculate the vehicle's speed by analyzing the returned signals.

On the other hand, RADAR technology has been a reliable tool in speed enforcement for decades. Radar guns emit radio waves as a focused beam, which bounce off the target vehicle and return to the device. By analyzing the frequency shift of the returned signal, the device can determine the vehicle's speed [rad].

However, both of these technologies have their limitations. For instance, LIDAR can be affected by adverse weather conditions like rain or fog, potentially reducing its effectiveness [DSPB23]. Conversely, RADAR signals may be susceptible to interference from nearby objects, which can complicate speed measurements in densely populated areas [Hos].

Both LIDAR and RADAR technologies, while highly effective in speed measurement, come with a notable price tag. A LIDAR device can range from several thousand to tens of thousands of euros [Pro]. In comparison, RADAR guns, though generally less expensive than LIDAR, can still cost around 2000 € for a high-quality unit [Sup].

These costs present a significant consideration for law enforcement agencies, especially those operating with limited budgets or in smaller jurisdictions. This limitation highlights the need for exploring more cost-effective alternatives without compromising accuracy and efficiency in speed enforcement.

Part I

Prototype Development

3 Literature Review

3.1 Methodic Product Development

Methodic Product Development, by Pahl and Beitz, stress the importance of systematic design in product development [Pah07, 9]. They differentiate between design science and methodology, with the latter being a practical approach based on scientific analysis and practical experience.

Pahl and Beitz describe the product development process as a series of stages, each with defined objectives and activities [Pah07, 128-133]. The four main stages are:

Planning and Task Clarification: The process starts with planning and defining tasks, often in collaboration with the marketing or dedicated planning team. It is essential to thoroughly understand the task, whether from a product proposal or a customer request. This step involves gaining detailed insights into requirements, constraints, and their importance, forming a solid foundation for the next steps.

Conceptual Design: This phase involves defining the necessary functions, establishing working principles, and integrating them into a working structure. Ultimately, this leads to creating a fundamental solution that embodies the core of the design vision.

Embodiment Design: Guided by technical and economic considerations, the physical structure is defined. Various initial layouts are assessed to identify design strengths and weaknesses, ultimately leading to selecting the most promising version.

Detail Design: In this phase, precise arrangements, dimensions, materials, and other aspects are defined. Product documentation is created, including drawings, parts lists, and assembly instructions.

Figure 3.1 shows the stages involved in Pahl and Beitz's design process.

3.2 Fused Deposition Modeling

Fused deposition modeling (FDM) is a widely used technique in additive manufacturing, particularly in 3D printing. It offers several advantages that contribute to its popularity in various industries. One of the main advantages of FDM is its ability to reproduce complex geometries with high precision and accuracy [GGA18].

This method makes it suitable for creating intricate and customized designs that may not be achievable with traditional manufacturing methods. Additionally, FDM is a cost-effective process as it reduces material waste by only depositing the necessary amount of material layer by layer [GGA18], which not only saves costs but also promotes sustainability in manufacturing.

Common plastics used in FDM include Acrylonitrile Butadiene Styrene (ABS), Polylactic Acid (PLA), and Polyethyleneterephthalate (PET) [Bat]. These materials have different mechanical properties, advantages, and disadvantages, making them suitable for different applications.

3.2.1 Original Prusa i3 MK3S+

The Original Prusa i3 MK3S+ is an FDM 3D printer designed for desktop use, ideal for tasks such as rapid prototyping and small-scale production. With a build volume of 250 mm x 210 mm x 200 mm, it can achieve layer heights ranging from 0.05 mm to 0.35 mm [Prua]. This printer has a heated bed and is compatible with various materials such as PLA, ABS, PETG, and nylon [Prua]. The default nozzle size is 0.4 mm, although alternate sizes can be utilized based



Figure 3.1: Pahl and Beitz's Design Process [Pah07, 130]

on specific printing needs.

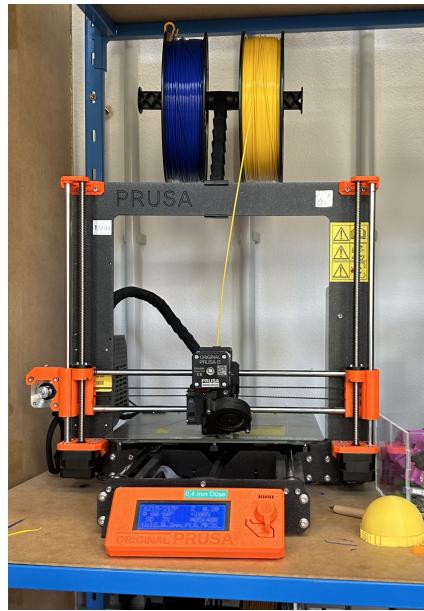


Figure 3.2: Original Prusa i3 MK3S+

This 3D printer is accessible to students and faculty at the University of Applied Sciences Brandenburg and will play a pivotal role in producing the prototype for this work. Figure 3.2 visually represents the Original Prusa i3 MK3S+, located within the University of Applied Sciences Brandenburg Workshop.

3.2.2 PrusaSlicer

PrusaSlicer [Pruc] is a free and open-source slicing software that converts 3D models into G-code, a language instructing the 3D printer on printing the object. It is compatible with a wide range of 3D printers and supports a variety of filament materials. PrusaSlicer offers many features that allow users to customize the printing process to suit their needs.

PrusaSlicer's ability to adjust printing parameters is crucial. These parameters include layer height, infill density, and print speed. Layer height determines the thickness of each layer of the printed object. Infill density refers to the material used to fill the object's inside. Print speed is the rate at which the printer moves while printing the object. Adjusting these parameters can help achieve

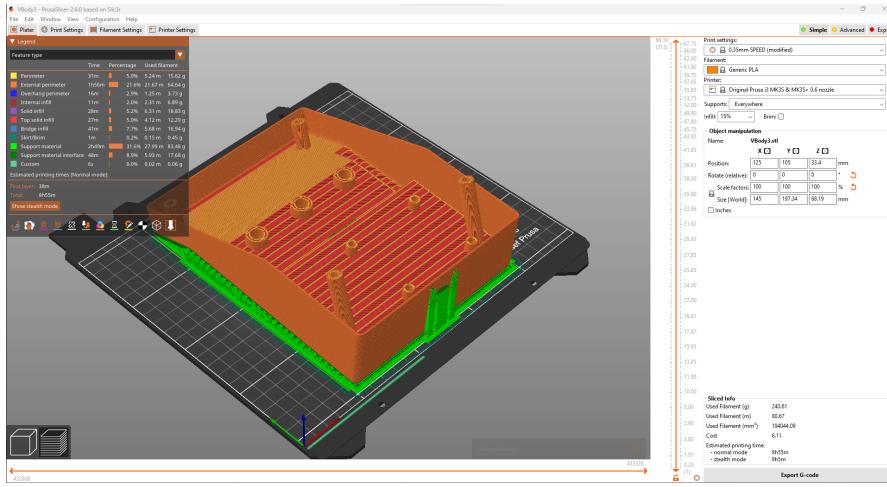


Figure 3.3: Example View of PrusaSlicer

the desired quality of the final product.

Adding support to 3D prints is an essential function offered by PrusaSlicer. Supports are structures that print alongside the object to provide extra stability during printing and prevent any potential collapse or deformation of the object. Depending on the complexity of the printed object, supports can be added manually or automatically. They can be made of the same material as the object or a different one. Polyvinyl Alcohol (PVA) is an excellent material for support as it is water-soluble, making it easy to remove after printing.

This software can also generate a preview of the printed object, which lets users visualize the result. Additionally, PrusaSlicer provides an estimate of the amount of filament required for the printing process and the duration of the printing process. Figure 3.3 shows a screenshot of PrusaSlicer. The left side of the window shows the 3D model of the object, while the right side shows the various parameters that can be adjusted.

3.2.3 Printing Cost

To perform a cost analysis of the 3D printing process, we will consider the following parameters:

- Material Cost (C_m)
- Energy Cost (C_e)

Equation 3.1 shows the formula used to calculate the material cost. This formula involves multiplying the weight of filament used (m_{fil}) by the cost of filament per kilogram(C_{fil}). The cost of the filament is dependent on the type of material used. We will use PLA for this project, which costs 29.99 €/kg [Prub].

Energy cost refers to the electricity cost of the printing process and is calculated using Equation 3.2. The printing duration (t_p) is estimated directly from the PrusaSlicer software, while the power consumption (P) of the printer is estimated to be about 0.08 kW [Prua]. By observing the average price of electricity in Germany for the year 2022 [Poo], the electricity price (C_{el}) is estimated at 0.235 €/kWh.

Equation 3.3 shows the formula for calculating the cost of 3D printing.

$$C_m = m_{fil} \cdot C_{fil} \quad (3.1)$$

$$C_e = t_p \cdot C_{el} \cdot P \quad (3.2)$$

$$C_{print} = C_m + C_e \quad (3.3)$$

This cost analysis focuses only on the expenses incurred during the 3D printing process and does not take into account the cost of acquiring the printer itself. By excluding the initial investment required to purchase a 3D printer, this analysis simplifies cost calculation and focuses solely on expenses related to printing. This method provides an accurate assessment of operational costs, enabling a meticulous understanding of the expenditure involved in producing each printed object.

4 Planning and Task Clarification

This chapter delves into planning and clarifying tasks for the prototype, as can be seen in Figure 4.1. As mentioned previously in Chapter 3.1, this step plays a critical role in the product development process. They involve precisely defining and understanding the requirements and expectations related to a specific task or project. The goal is to eliminate confusion and ensure comprehension among all involved parties.

During this step, the task's specific goals, limitations, and requirements are identified [Pah07, 145]. Additionally, Pahl and Beitz formulated a series of questions that must be answered to help better understand the project requirements [Pah07, 145]. These questions are:

- What is the objective of the solution?
- What characteristics should the solution have?
- What characteristics should the solution avoid?

The requirements for the solution can be identified and spelled out by answering these questions. These requirements will serve as the basis for the subsequent phases of the product development process. The outcome of this step is a list of requirements that outline the needs, expectations, and restrictions tied to the task [Pah07, 145].



Figure 4.1: Planning and Task Clarification [Pah07, 146]

4.1 Establishing the Prototype's Requirements

To properly establish the requirements for the prototype, it is suggested to properly define the objectives of the prototype and classify them into demands and wishes [Pah07, 146].

Demands, as described by Pahl and Beitz [Pah07, 147], are the essential and non-negotiable requirements that must be fulfilled for the product to be considered successful. They represent the core functionality and characteristics that the product must possess to meet its intended purpose and provide value to the users. Demands are typically based on objective criteria and are crucial for

ensuring the product's functionality and compliance.

On the other hand, wishes, as defined by the authors [Pah07, 147], are desirable but non-essential requirements or features that clients or stakeholders would like to see in the product. Wishes often involve additional functionalities, aesthetics, or user experience enhancements that would provide added value or differentiate the product in the market. While wishes may not be mandatory, they can contribute to customer satisfaction, competitive advantage, and overall product excellence.

In addition, if possible, all of the requirements defined must be quantifiable, which means that the requirements must be measurable and testable [Pah07, 147]. This specification is essential for ensuring that the requirements are met and that the product can fulfill its intended purpose.

4.2 Identifying the Prototype's Requirements

In this section, the requirements of the prototype will be established. The checklist (see Figure 4.2) will be used as a guideline to ensure that all the requirements are correctly identified and defined.

Geometry

When creating a prototype, it is crucial to adhere to specific size parameters to ensure its functionality and usability for end-users or customers. The prototype must conform to a general size limit of 300mm x 300mm x 300mm (length x width x height). Any size larger than the specified parameters could lead to difficulties in handling, portability issues, and a bulky user experience. Therefore, it is essential to keep the prototype's size within the limit to ensure its effectiveness.

Additionally, we must consider the limitations of the 3D printer's size capacity. We have opted to employ the printer mentioned previously in Section 3.2.1. This particular printer has a maximum printing area of 210 mm x 210 mm x 250 mm [Prua].

Main headings	Examples
Geometry	Size, height, breadth, length, diameter, space requirement, number, arrangement, connection, extension
Kinematics	Type of motion, direction of motion, velocity, acceleration
Forces	Direction of force, magnitude of force, frequency, weight, load, deformation, stiffness, elasticity, inertia forces, resonance
Energy	Output, efficiency, loss, friction, ventilation, state, pressure, temperature, heating, cooling, supply, storage, capacity, conversion.
Material	Flow and transport of materials. Physical and chemical properties of the initial and final product, auxiliary materials, prescribed materials (food regulations etc)
Signals	Inputs and outputs, form, display, control equipment.
Safety	Direct safety systems, operational and environmental safety.
Ergonomics	Man-machine relationship, type of operation, operating height, clarity of layout, sitting comfort, lighting, shape compatibility.
Production	Factory limitations, maximum possible dimensions, preferred production methods, means of production, achievable quality and tolerances, wastage.
Quality control	Possibilities of testing and measuring, application of special regulations and standards.
Assembly	Special regulations, installation, siting, foundations.
Transport	Limitations due to lifting gear, clearance, means of transport (height and weight), nature and conditions of despatch.
Operation	Quietness, wear, special uses, marketing area, destination (for example, sulphurous atmosphere, tropical conditions).
Maintenance	Servicing intervals (if any), inspection, exchange and repair, painting, cleaning.
Recycling	Reuse, reprocessing, waste disposal, storage
Costs	Maximum permissible manufacturing costs, cost of tooling, investment and depreciation.
Schedules	End date of development, project planning and control, delivery date

Figure 4.2: Checklist for Establishing the Prototype's Requirements [Pah07, 149]

Ideally, each printed component should fall within the specified printing size range. However, should a component exceed these dimensions, it will be necessary to divide it into two or more parts to facilitate printing. This approach guarantees that each part can be accommodated within the printer's size constraints.

Energy

The energy required for the prototype is crucial because it determines its usefulness and convenience. We have set a requirement that the prototype should function independently for at least an hour using the provided power supply. This guideline is in place to ensure that the prototype can function autonomously and provide users with a seamless experience.

By meeting this requirement, the prototype demonstrates that it can operate reasonably without frequent charging or relying on external power sources. This feature enables users to use the device without any concerns for an extended period, giving them more opportunities to explore its functionality and capabilities.

One possible solution could be incorporating a battery compartment into the prototype design. This design would enable users to easily swap out the batteries when they need to be replaced. Such a feature would offer greater flexibility and convenience for users, allowing them to replace depleted batteries quickly and continue using the device without any interruption.

Forces

The force requirement for the prototype has two main aspects: ensuring it can handle the weight of its components while also adhering to a maximum weight limit.

Firstly, it is crucial to verify that the prototype can effectively support the weight of its components without compromising its overall structure or functionality, which ensures the prototype's durability and ability to withstand the forces exerted by its components. Additionally, it guarantees that the prototype can be manipulated and operated without the risk of damage or malfunction.

Furthermore, there is a specific constraint that the total weight of the prototype must not surpass 2 kg. This requirement encompasses the collective weight of all internal components, including predefined components and any additional materials integrated during the design process. Adhering to this weight limit ensures the prototype remains lightweight and manageable while meeting the

intended performance criteria.

Materials

When developing the prototype, it is of utmost importance to thoughtfully consider the specific materials and elements that will be utilized. The client has already preselected specific components for this project, which are mandatory to meet the requirements. These components include the Raspberry Pi 4B, a 7-inch touch screen, and the Raspberry Pi Camera V2.

These elements are fundamental building blocks for the prototype's operation. The Raspberry Pi 4B, functioning as a versatile single-board computer, furnishes computational power and acts as the central control unit for the prototype. The 7-inch touchscreen enhances user interaction by providing a responsive and user-friendly interface for both input and output. The Raspberry Pi Camera V2 facilitates the capturing of images and videos.

Ergonomics

Creating a prototype that satisfies specific ergonomic size, weight, and handling requirements is essential. The key objective is to produce a compact and lightweight prototype, enabling effortless carrying and maneuvering, thereby enhancing user comfort and convenience.

In addition, guaranteeing that users can comfortably hold the prototype is critical to the ergonomic specifications. This specification requires a detailed evaluation of the prototype's shape, grip, and balance to ensure that it is easy to hold and secure. The design should seamlessly fit into the user's hand, providing a stable, ergonomically sound grip.

Production

The client has specified the fabrication of the prototype components to utilize the 3D Printing technology. Furthermore, the design of the prototype must accommodate the use of PLA filament. This material, known for its widespread availability and well-rounded combination of strength and flexibility, aligns with the requirements of the prototype.

Operation

There are two fundamental criteria that the prototype needs to meet: it should be user-friendly for freehand operation without any additional support and adaptable for use with a tripod to guarantee stability.

To meet the first criterion, the prototype's design should enable easy operation without extra assistance. It should have an ergonomic design, featuring a comfortable grip and user-friendly controls, ensuring an intuitive experience for users.

The second criterion requires the prototype to seamlessly integrate with a tripod, providing enhanced stability whenever necessary. This feature lets users securely attach the prototype to a tripod, creating a stationary and steadfast setup. With tripod compatibility, the prototype can cater to situations where a stable and controlled operation or positioning is essential.

Assembly

The assembly requirement for the prototype emphasizes the importance of considering the ease of assembly and disassembly of its components. This design consideration enables users to access the inner components easily, facilitating maintenance and repair tasks.

By designing the prototype with ease of assembly in mind, it becomes simpler for users to put the components together without requiring complex tools or specialized knowledge. This requirement promotes user-friendliness and reduces the time and effort required for initial assembly or subsequent modifications. Similarly, easy disassembly allows users to access the internal components when needed, simplifying troubleshooting, repairs, or component replacements.

The prototype's design should have swappable properties so individual components or modules can be easily removed and replaced without disassembling the entire prototype. Swappable parts enhance modularity, flexibility, and cost-effectiveness, as users can upgrade or replace specific components as needed rather than replace the entire prototype.

Costs

The cost requirement for the prototype focuses on the total cost of production. The manufacturing of the prototype must be within a budget of 300 €, excluding the cost of the predefined components. This budget encompasses the cost of all materials and components used in the production process.

Schedules

The schedule requirement for the prototype focuses on the time required for the design and production phase. The prototype's design must allow for manufacturing within a 2-month window, covering the entire production process, from design to assembly.

Durability

The durability standard for the prototype encompasses considerations for its ability to withstand dust and water, if possible. While achieving complete resistance may only sometimes be attainable, efforts should be directed toward enhancing the prototype's durability in these aspects.

Concerning dust resistance, the prototype's design should minimize the entry of dust particles into its internal components and sensitive areas. This requirement involves using appropriate seals, filters, or protective enclosures to prevent dust from negatively impacting the prototype's performance or functionality.

In terms of water resistance, if relevant to the intended use, the prototype should demonstrate a level of protection against water penetration. This specification may incorporate waterproof or water-resistant materials, seals, or coatings to shield the internal components from moisture.

4.3 Requirement List

Table 4.1 and Table 4.2 shows the list of requirements for the prototype, including the demands and wishes. The demands are marked with a D, while the wishes are marked with a W.

TH Brandenburg		Requirement List Speed Camera	Issued on 1/7/2023
Changes	D/W	Requirements	Page: 1
5/7/2023		Geometry D 1. Length < 300 mm D 2. Width < 300 mm D 3. Height < 300 mm W Parts size: 210 mm x 210 mm x 250 mm or less	
21/8/2023		Energy D Minimal operation time: 1 hour	
		Forces Total prototype weight < 2 kg	
		Materials D Use all predefined components	
		Ergonomics W Lightweight W Comfortable handling W Compact	
		Production D 3D Printed D Use PLA filament	
		Operation D Able to be used in freehand D Able to integrate with tripod stand	
		Assembly W Simple assembly or component used D Swappable Parts	

Table 4.1: Requirement List (1/2)

Planning and Task Clarification

TH Brandenburg		Requirement List Speed Camera	Issued on Page:	1/7/2023 2
Changes	D/W	Requirements		
13/7/2023	D	Costs Manufacturing costs < 300 €		
	D	Schedules Finished by: October 2023		
	W	Durability Resistant against water		
	W	Resistant against dust		

Table 4.2: Requirement List (2/2)

5 Conceptual Design

Following the clarification of the task is the conceptual design, where in this section of the product development process, designers engage in creative exploration and evaluation of various design ideas and concepts.

According to Pahl and Beitz, conceptual design is the stage of the design process where important issues are pinpointed through abstraction [Pah07, 159]. The process involves establishing function structures, searching for suitable working principles, and ultimately combining these elements to create a working structure.

Figure 5.1 shows the steps involved in this phase.

5.1 Abstraction

Due to new technologies, procedures, materials, and scientific discoveries, traditional solution principles or designs may not be able to provide optimal answers, and to overcome fixation on conventional ideas, designers utilize abstraction, focusing on the general and essential aspects rather than particular details [Pah07, 161].

To help in identification of the essential problems, following abstraction techniques are used [Pah07, 165]:

- **Step 1:** Eliminate personal preferences.
- **Step 2:** Omit requirements that have no direct bearing on the function and the essential constraints.

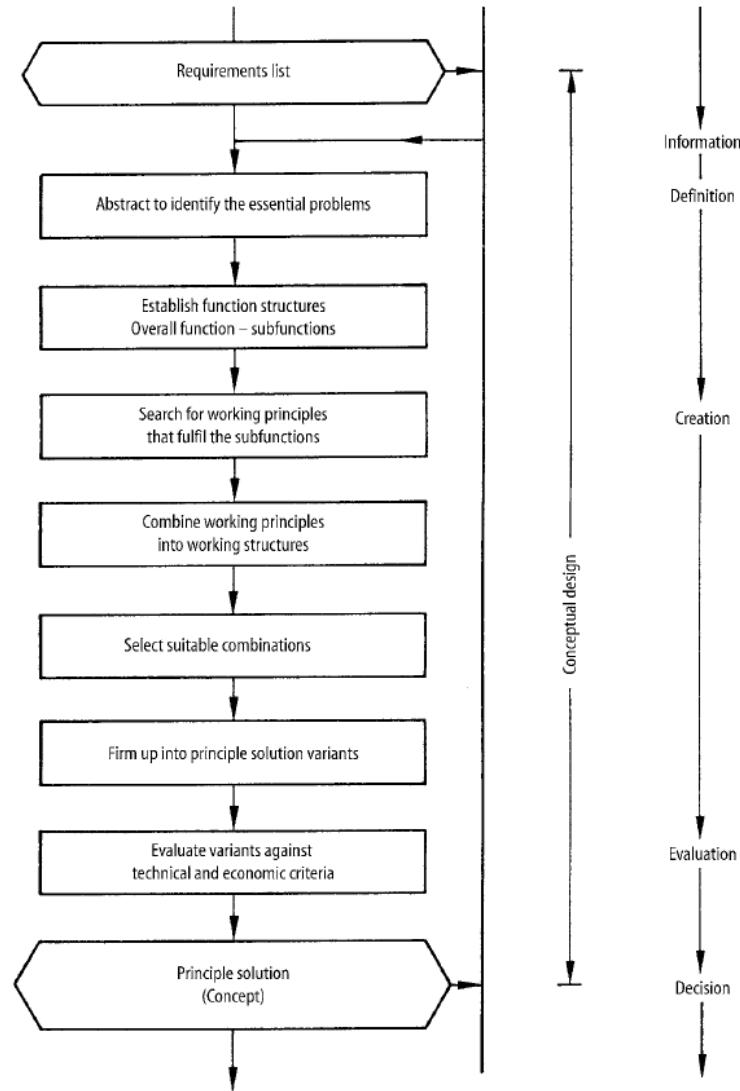


Figure 5.1: Steps in Conceptual Design [Pah07, 160]

- **Step 3:** Transform quantitative into qualitative data and reduce them to essential statements.
- **Step 4:** Generalise the previous step's results.
- **Step 5:** Formulate the problem in solution-neutral terms.

Figure 5.2 shows the result of the abstraction process.

Result of Step 1 and Step 2

- Ergonomic: Comfortable to hold, Easy to use,
Weight distributed evenly
- Portable: Lightweight, Small
- Size (MAX):
 - Length: 300 mm
 - Width: 300 mm
 - Height: 300 mm
- Weight (MAX): 2 kg
- Design: Components are packed in a chassis
- Camera: Camera must be presented in the
prototype
- Power: Battery powered, Rechargeable
battery, Duration min. 1 hour
- Control: Control via touch screen
- Optional Requirements:
- Durability: Water resistance, Dust resistance
- Modular: Easy to assemble and disassemble,
Swappable parts
- Features: Mountable on a tripod
- Production: 3D printed parts

Result of Step 3 and Step 4

- Comfortable to hold, easy to use, and have
evenly distributed weight.
- Lightweight and small.
- Not exceed 300 mm in dimension.
- Weigh less than 2 kg.
- Power that lasts a minimum of 1 hour.
- Produced with 3D Printer.
- Optional Requirements:
- Durable against water and dust.
- Modular

Result of Step 5 (Problem Formulation)

- Design a portable device that prioritizes user
comfort, ease of use, and ergonomic design
while utilizing 3D printing.

Figure 5.2: Result of Abstraction Process

5.2 Function Structures

Pahl and Beitz [Pah07, 31] define function structures as a graphical representation of the functions of a system and their interrelationships. It is a hierarchical representation of the functions of a system, starting with the overall function and breaking it down into sub-functions. The function structure is a helpful tool for identifying a system's essential functions and the relationships between the functions.

Figure 5.3 shows the representation of the function structure and the process of breaking down the overall function into sub-functions.

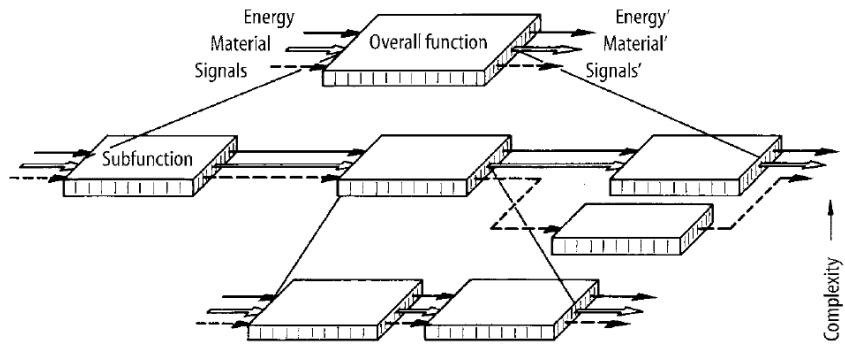


Figure 5.3: Breaking down the overall function into sub-functions [Pah07, 32]

5.2.1 Overall Function

Based on the result of abstraction, the system's overall function can be represented and visualized using a function structure diagram, as shown in Figure 5.4.

In this overall function, the prototype's components are defined as an input, while the prototype itself is defined as the output. The overall function is decomposed into sub-functions in the next section.



Figure 5.4: Overall Function of the System

5.2.2 Sub-Functions

Decomposing the overall function into sub-functions is crucial in the conceptual design process, and as described by Pahl and Beitz [Pah07, 170], the purpose of this decomposition is to reduce the complexity of the overall system and facil-

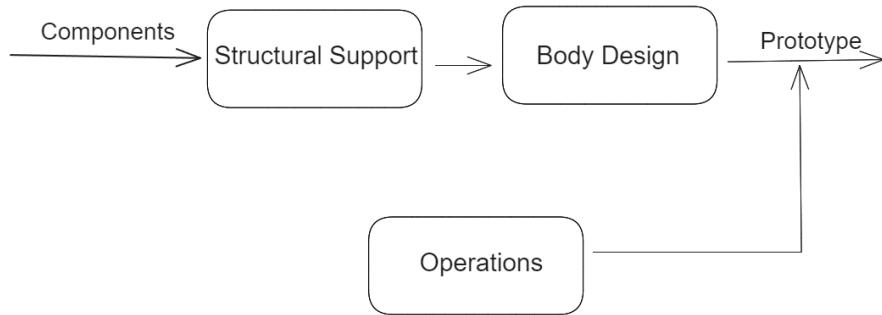


Figure 5.5: Sub-Functions of the System



Figure 5.6: Sub-Functions of the System (Final)

tate the identification of suitable solution principles that can fulfill the required functions.

Figure 5.5 illustrates the sub-functions of the prototype. Deriving from the overall function, labeled as *Prototype Design*, it breaks down into three subfunctions, specifically *Structural Support*, *Body Design*, and *Operation*. The function is then

further decomposed into more detailed sub-functions in Figure 5.6.

The term *Structural Support* refers to the measures taken to ensure the structural integrity of the prototype. This sub-function encompasses activities such as securing and stabilizing the internal components within the prototype. To simplify the function, it decomposes into three sub-functions: *Component Placement*, which specifies the positioning of internal components; *Component Orientation*, which details the alignment of internal components; and *Component Type*, which is the type of component itself.

Body Design describes the sub-functions involving the prototype's physical structure. To further simplify the task, this sub-function is decomposed into *Body Type*, which defines the outline of the structure, and *Handling*, which describes the handling of the prototype.

Operation deals with how the prototype works. It describes the device's usage and the components involved during operation. This function is then divided into *Control*, which describes the component involved in controlling input from user during operation, and *Mounting*, which refers to the integration of the prototype with the tripod stand.

5.3 Developing Working Principles

As defined by Pahl and Beitz, working principles refer to the physical effects and characteristics that fulfill specific functions of the designed structure [Pah07, 181]. These principles are combined to create the working structure, encompassing physical processes and the form design features. Several potential working structures can be generated by varying the physical effects and form design features, known as the solution field.

In developing working principles, there are multiple available methods in idea generation. These methods are categorized into three groups:

- Conventional methods

- Intuitive methods
- Discursive methods

Pahl and Beitz [Pah07, 18-82] describe the *Conventional Methods* as a systematic and data-driven approach. Designers gather information from various sources, such as literature, trade publications, and competitor catalogs, to stay informed about advancements and best practices. They analyze natural systems and existing technical systems to draw inspiration and identify opportunities for improvement.

On the other hand, the *Intuitive Methods*, as described by them [Pah07, 82-89], tap into creativity and associative thinking. *Brainstorming* fosters a collaborative environment where diverse perspectives generate a wide range of ideas without judgment. *Method 635* adds structure to Brainstorming, allowing for systematic idea development within a group. The *Gallery Method* combines individual work with group discussions, using sketches or drawings to explore solution proposals visually.

Additionally, the *Discursive methods* [Pah07, 89-103] combines systematic, step-by-step procedures with elements of intuition and creativity. They involve deliberate analysis of physical processes, leading to multiple solution variants derived from the relationships between variables. This approach fosters a deeper understanding of the problem space.

5.3.1 Searching for Working Principles

Brainstorming and *Analysis of Existing Technical Systems* are utilized in this work to establish working principles. Table 5.1 shows the result of idea generation. For more detailed sketches of the working principles, please refer to Appendix A.1.

		Working Principles			
		1	2	3	4
Function	Components Arrangement	Tablet-like	Point-of-Service-like	Handheld-PC-like	Camcorder-like
	Screen Orientation	Landscape	Portrait		
	Battery Type	Battery Pack	Power Bank	AAA Batteries with Battery Holder	
	Body Type	Bowl	Skeleton	Sandwich	
	Handling	Body Grip	Bump Grip	Pistol Grip	
	External Mounting	Detachable Plate	Built-in Mounting Plate		
Control Mechanism	Button	Touch Screen	Trigger	Touch and Button	

Table 5.1: Classification Scheme for Working Principles

5.4 Combination of Working Principles

In this step, we will combine the working principles assigned to the sub-functions to create potential working structures. To achieve this, the identified working principles must be linked following the functional structure to fulfill the overall function.

The method we will employ for systematic combination is Zwicky's morphological box [Kus22]. In this approach, the potential principles are represented in a table for better clarity and connected to form functional structures using

connecting lines.

Figure 5.7 shows the morphological chart with the generated solution variants. The solution variants are labeled S1 to S8, with each color representing a different solution variant.

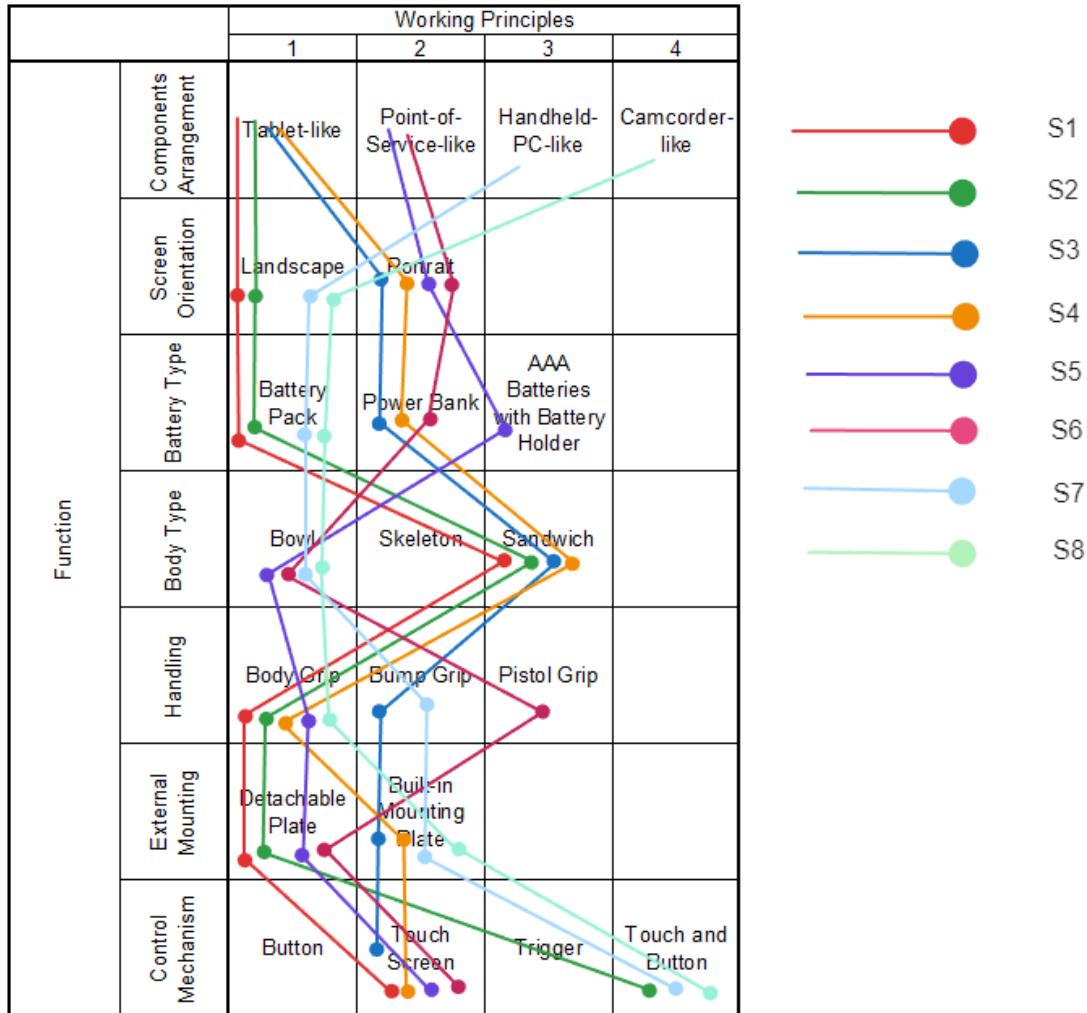


Figure 5.7: Morphological Chart with Solution Variants

5.5 Firming Up into Principle Solution Variants

In this section, we showcase hand-drawn sketches of identified functional structures that have been transformed into practical solution alternatives. Each sketch is accompanied by a brief description of its operations, highlighting its potential strengths and weaknesses. The sketches are based on the result of the morphological chart in Figure 5.7.

5.5.1 Solution Variant 1, S1

In Solution Variant 1, we encounter a tablet-like design that closely resembles a typical tablet device. The key components, including the Raspberry Pi, Battery, Camera, and Screen, are arranged in a manner similar to a tablet. The screen orientation is in landscape mode, offering a broader display view for enhanced visual clarity. This orientation is particularly beneficial when the device is used for tasks that require a wider viewing area.

The design is thoughtfully optimized for handheld use, featuring a body grip that ensures comfortable handling. The internal battery integration contributes to a seamless and integrated appearance. A sandwich-type body provides robust protection for the internal components, comprising a top cover, main body, and bottom cover.

For mounting purposes, Solution Variant 1 utilizes a detachable plate tripod system, offering the convenience of easy attachment and removal from a tripod stand. The primary control mechanism for this variant is a touch screen, allowing for intuitive and user-friendly interactions with the device's functionalities.

Figure 5.8 shows the sketch of Solution Variant 1.

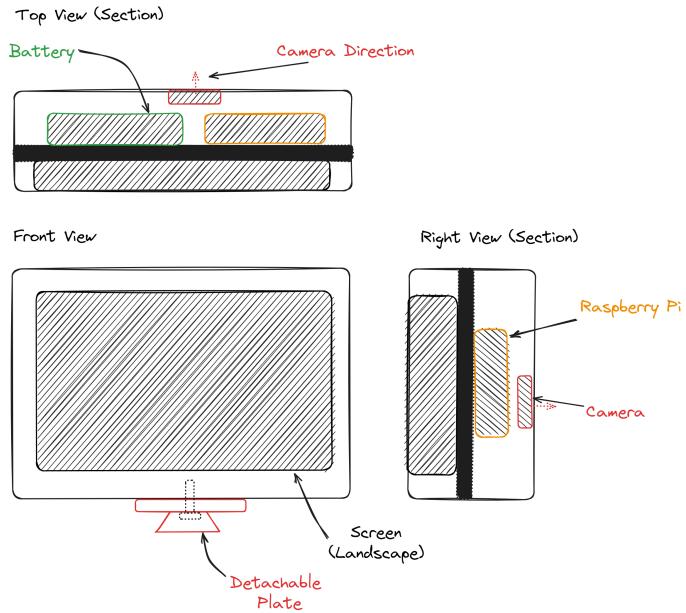


Figure 5.8: Sketch of Solution Variant 1

5.5.2 Solution Variant 2, S2

Similar to previous variant, Solution Variant 2 maintains a tablet-like design, with components arranged like a tablet device (see Figure 5.9). One significant difference lies in the control mechanism. While Solution Variant 1 relies on a touch screen, Solution Variant 2 incorporates physical buttons as the primary control mechanism. This addition enhances versatility and usability in various scenarios, as users can choose between touch-based and tactile input.



Figure 5.9: Sketch of Solution Variant 2

5.5.3 Solution Variant 3, S3

While Solution Variant 3 maintains the tablet-like component placement found in the previous variants, it introduces a significant change by adopting a portrait screen orientation. This shift opens up new possibilities for the device's usage, particularly in scenarios where vertical screen space is more advantageous than horizontal layout.

The design includes a bump grip for secure and comfortable handling in a vertical position. Notably, the battery is positioned externally in this variant, offering the potential advantage of easier access and replacement. The body structure remains a sandwich-type, providing robust protection for the internal components.

One notable advantage of the portrait screen orientation is the improved stability of the device, as the center of gravity is aligned with the device's center. This alignment enhances balance and control when using the device in various orientations, thus enhancing overall usability and versatility.

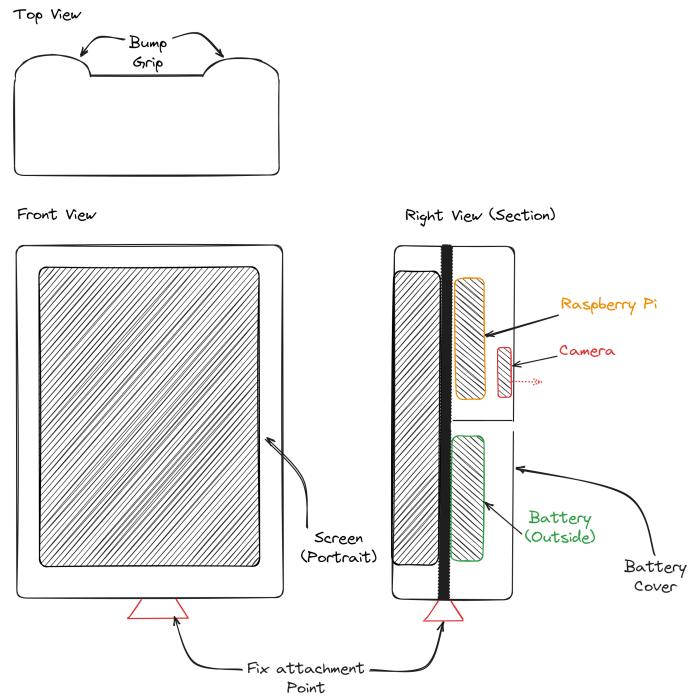


Figure 5.10: Sketch of Solution Variant 3

5.5.4 Solution Variant 4, S4

Solution Variant 4 copies many features from Solution Variant 3, but with one significant change in the body type. Solution Variant 4 opts for a more minimalist skeleton design, which results in a lightweight yet adequately supportive body for the internal components. This design choice is particularly beneficial for users who prefer a lightweight device for extended usage periods. Figure 5.11 shows the sketch of Solution Variant 4.

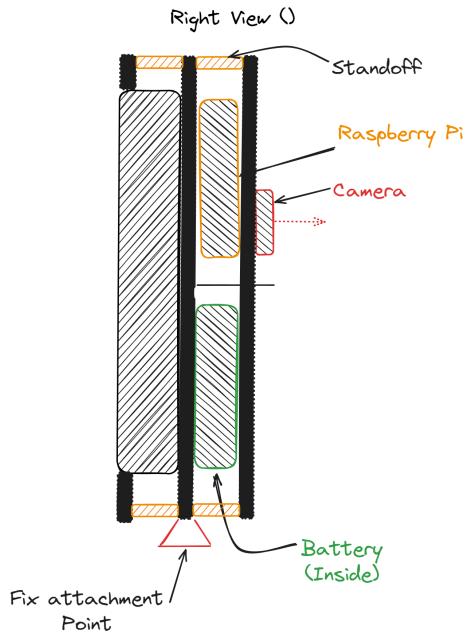


Figure 5.11: Sketch of Solution Variant 4

5.5.5 Solution Variant 5, S5

Solution Variant 5 introduces a unique design approach, deviating from the tablet-like structure seen in previous solutions. Instead, it adopts a Point of Service-like component placement, where the Raspberry Pi, Battery, Camera, and Screen are configured in a distinctive layout. The screen is positioned at an angle, differentiating it from the previous variants (see Figure 5.12).

Regarding screen orientation, Solution Variant 5 retains a portrait mode, which can be advantageous in scenarios requiring vertical displays. The device is designed for body grip handling, offering a secure way to hold and interact with the device.

A notable difference is the external AAA battery setup, which enhances convenience by offering easy battery replacement and compatibility with standard batteries. The body structure follows the familiar sandwich-type design, providing robust protection for the internal components.

For mounting purposes, Solution Variant 5 utilizes the detachable tripod system, enabling seamless attachment and detachment from a tripod stand. Like its predecessors, it relies on a touch screen as the primary control mechanism, ensuring intuitive user interactions.

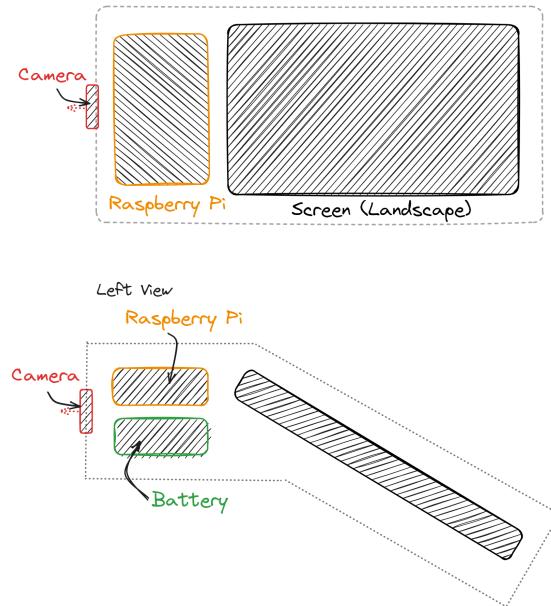


Figure 5.12: Sketch of Solution Variant 5

5.5.6 Solution Variant 6, S6

Solution Variant 6 closely mirrors Solution Variant 5 regarding component placement and screen orientation. This variant, too, adopts the Point of Service-like layout with a portrait screen orientation. However, it introduces a pistol handle for handling, providing a firm and ergonomic grip for users, as can be seen in Figure 5.13.

The battery is positioned externally, offering the same benefits of easy battery replacement and extended usage periods. Regarding body design, Solution Variant 6 employs a bowl-like structure with all components attached to the main body. This design choice provides protection and enclosure while reducing overall weight.

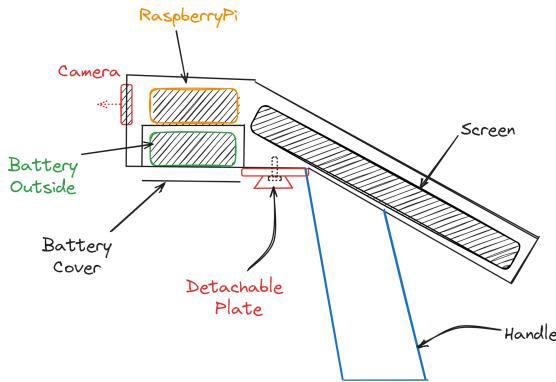


Figure 5.13: Sketch of Solution Variant 6

5.5.7 Solution Variant 7, S7

In Solution Variant 7, a distinct design approach with a Handheld PC-like component placement is produced. This configuration aligns the screen and battery, positioning the Raspberry Pi behind the screen (see Figure 5.14).

The variant is specifically designed with a bump grip, providing a secure and comfortable handling experience. Additionally, a rounded battery can be used within this variant and placed inside the bump grip. This design has the potential to offer a more streamlined and integrated appearance.

The chassis structure adopts a bowl-like design, ensuring secure enclosure and protection for all components. The device incorporates a built-in tripod system for mounting, providing a stable attachment to a tripod stand.

Solution Variant 7 combines a touch screen and physical buttons as the control mechanism. This dual-input approach provides users multiple options for interacting with the device's functionalities, enhancing versatility and usability in various scenarios.

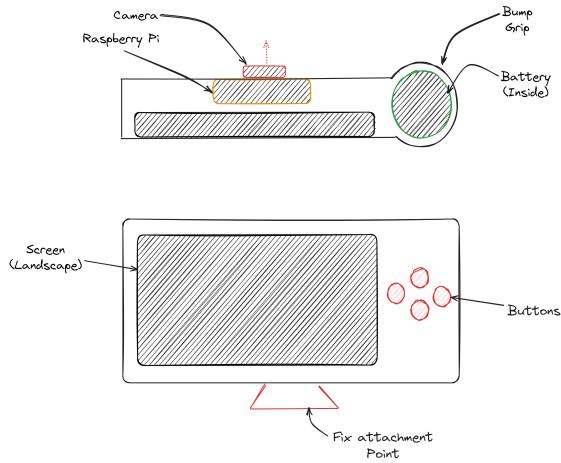


Figure 5.14: Sketch of Solution Variant 7

5.5.8 Solution Variant 8, S8

Solution Variant 8 features a Camcorder-like component placement. The Raspberry Pi, Battery, Camera, and Screen are arranged similarly to a camcorder, with the screen positioned at a hinge, allowing it to change angles for flexible viewing.

The screen orientation remains landscape, providing a broad horizontal display view. The device is designed with a body grip for secure and comfortable handling. The battery is placed internally, and a power bank is used to provide a reliable power source.

The chassis structure follows a bowl-like design, offering protection and sturdiness for the internal components. A fixed-mount tripod system is employed for mounting purposes, providing stability and ease of use when attaching the device to a tripod stand.

As with some of the previous variants, Solution Variant 8 combines both a touch screen and physical buttons as the control mechanism, offering users the flexibility to interact.

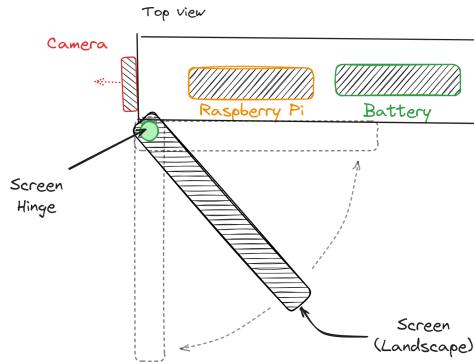


Figure 5.15: Sketch of Solution Variant 8

5.6 Filtering of Solution Variants

As shown in Figure 5.7, multiple solution variants were generated. However, not all of these solutions are feasible and practical. As advised by Pahl and Beitz [Pah07, 106-107], it is necessary to reduce the vast number of theoretically possible but practically unachievable solutions as early as possible. However, caution should be exercised not to discard valuable working principles, as they often play a crucial role in forming a favorable and effective working structure when combined with others.

Additionally, Pahl and Beitz [Pah07, 107] suggest a method that can be used to filter the solution variants. This method is known as the selection chart, which consists of two steps: elimination and preference. Initially, all clearly unsuitable proposals are removed. If a substantial number of solutions remain, preference is given to those who stand out as markedly superior. Only these preferred solutions are evaluated during the final stages of the conceptual design phase.

Pahl and Beitz suggest the following criteria for eliminating unsuitable solutions:

- **Criteria A:** Compatible with the overall task
- **Criteria B:** Fulfill demands of requirement list

- **Criteria C:** Realisable in principle
- **Criteria D:** Within permissible cost

These criteria are applied step by step to examine each solution. If any of the exclusion criteria are not met, the solution is rejected, and further criteria are not assessed. Additionally to the exclusion criteria, the following preference criteria are used to prioritize the remaining solutions:

- **Criteria E:** Incorporates direct safety measures
- **Criteria F:** Preferred by the designer

Criteria E and F are then used to prioritize solutions if there are still too many options after the initial screening. The remarks column provides explanations for excluding or favoring each solution. The final assessment of the functional principles is recorded in the rightmost column of the selection list.

Page 1		Selection Chart									
Solutions Variant	No.	Evaluate solution variants according to selection criteria						Decision			
		Compatible with the overall task		fulfill demands of requirement list		Realisable in principle		Within permissible costs	Incorporates direct safety measures	Preferred by the designer	Remarks:
		A	B	C	D	E	F			(+) Pursue Solution	
										(-) Eliminate Solution	
										(?) More Information Required	
										(!) Check Specification	
S1	1	+	+	+	+	?	+	Might have problem with ergonomic	-		
S2	2	+	+	+	+	?	?			+	
S3	3	+	+	+	+	?	+			+	
S4	4	-	+	+	+	+	+	Have almost no protection of inner components	-		
S5	5	+	+	+	+	?	+	Less ergonomics due to wide body	-		
S6	6	+	+	+	+	?	?			+	
S7	7	+	+	+	+	+	+			+	
S8	8	+	+	-	?	?	-	Too complex	-		

Figure 5.16: Selection Chart for Solution Variants

The result of the selection chart, as shown in Figure 5.16, indicates that solutions S1, S4, S5, and S8 have been eliminated and will not be considered for the next stage of the design process.

6 Embodiment Design

The next phase in the design methodology is embodiment design. This phase, as defined by Pahl and Beitz [Pah07, 227], involves starting with the fundamental solution or concept for a technical product and then advancing the design in alignment with technical and economic criteria, taking into account further information. The ultimate objective is to reach a stage where the subsequent detailed design can smoothly progress into the production phase. Figure 6.1 shows the steps involved in this phase.

6.1 Basic Rules of Embodiment Design

Regarding product design, some basic rules must be followed. As defined by Pahl and Beitz [Pah07, 234-235], they include clarity, simplicity, and safety. Neglecting these rules can potentially result in issues and accidents. Subsequent sections will provide a comprehensive exploration of these guidelines.

6.1.1 Clarity

Clarity, as described by Pahl and Beitz [Pah07, 235], includes establishing clear and unambiguous connections within a design, ensuring straightforward relationships between subfunctions, inputs, and outputs to prevent confusion or misinterpretation.

They also mention that clarity applies to the broader design structure, whether it involves multiple working principles or component combinations. It mandates that the design facilitates the orderly flow of energy, materials, and sig-



Figure 6.1: Steps in Embodiment Design [Pah07, 229]

nals, preventing adverse effects like excessive forces or wear.

6.1.2 Simplicity

As defined by Pahl and Beitz [Pah07, 242], simplicity in design is characterized by an uncomplicated and easily understandable approach, often achievable by using fewer components. Such simplicity can save costs, reduce wear and tear, and minimize maintenance requirements. Nonetheless, striking a balance is crucial, as certain functions inherently demand a minimum number of components.

6.1.3 Safety

According to Pahl and Beitz [Pah07, 247], safety considerations are crucial in ensuring both the adequate performance of technical functions and the protection of people and the environment. Designers rely on a safety methodology outlined in the German industry standard DIN 31 000, encompassing three levels: direct safety, indirect safety, and warnings. Designers should prioritize direct safety measures, seeking solutions that inherently eliminate potential dangers. Only when this is not feasible should they resort to indirect safety measures involving the construction of specialized protective systems.

Warnings highlighting dangers and hazard zones are best utilized in conjunction with direct and indirect safety measures, clarifying specific risks. As designers address technical challenges, they encounter various constraints, not all of which can be simultaneously overcome. However, their objective remains to develop solutions that come as close as possible to meeting all requirements. It is important to note that exceptionally high safety demands can complicate design, potentially diminishing clarity and economic viability and possibly leading to project abandonment.

6.2 Guideline of Embodiment Design

In addition to the basic rules of embodiment design, Pahl and Beitz [Pah07, 308] also stress the importance of following a set of design guidelines to help designers meet specific requirements and constraints. For this project, the *Design for Production* guideline are applied.

6.2.1 Design for Production

The concept of *Design for Production* outlined by Pahl and Beitz [Pah07, 355-356] underscores the significance of factoring in the production process during the design stage. This methodology empowers designers to fine-tune production costs and timelines while maintaining the product's functionality and quality. They highlight that adhering to fundamental principles of clarity and simplicity sets designers on the correct path towards realizing this objective.

Appropriate Overall Layout Design

Pahl and Beitz [Pah07, 355-362] mentioned that the overall layout design, derived from the function structure, influences product division into assemblies and components, including sourcing decisions (in-house, bought-out, standard parts), production procedures, dimensions, batch sizes, joining methods, and quality control. The layout can lead to differential, integral, composite, or building-block construction methods.

Differential Construction involves breaking down components into quickly produced parts, facilitating adaptability, increased component batch sizes, and more accessible quality assurance. However, it demands more outstanding machining and assembly costs and may have functional limitations due to joints.

Integral Construction combines multiple parts into a single component, usually utilized for product optimization. It can greatly reduce cost for assembly and quality control while enhancing functionality and performance. However it usually requires more complex production procedures and may be less adapt-

able.

Composite Construction refers to the integration of various components, each constructed differently, into a single unit that may require additional processing. This includes scenarios like combining cast and forged parts. Moreover, it involves employing multiple methods of joining to bring together different elements simultaneously.

Building Block Construction results from splitting components so that the parts or assemblies can be used in other products or variants, offering flexibility and cost savings. This approach is often used in modular design, where components are designed to be easily interchangeable.

Appropriate Form Design of Components

As mentioned by Pahl and Beitz [Pah07, 362], numerous factors influence the cost, time, and production quality. These include parameters like shapes, dimensions, surface finishes, tolerances, and fits. These choices are essential in determining production procedures, machine types, materials, in-house vs. bought-out components, and quality control measures.

Furthermore, production facilities can impact the design of features, such as dimension limitations that necessitate component division or the procurement of bought-out components. Many guidelines are available for designing appropriate component forms. The design guideline for 3D printed components is shown in Figure 6.2.

Appropriate Use of Standard and Bought-Out Components

The authors advised that [Pah07, 374-375], designers should use readily available standard or bought-out components rather than specially produced ones to ensure favorable supply and storage conditions. Bought-out parts are often more cost-effective than in-house production. The decision between in-house or bought-out components depends on factors like production volume, market demand, costs, and available facilities. These factors may change over time, requiring periodic re-evaluation, especially for unique or batch products in heavy engineering.

Complete design guide for 3D printing:



Common file errors:		Design tips:	Ways to save:
Holes Any holes in a mesh makes it non-manifold and must be closed.	Wrong normals Normals help the computer understand what is in and out, and what the volume of the model is. All normals must be outward facing.	Escape holes For any cavities there must be sufficient escape holes for support material to escape.	Hollowing The most efficient way to save material and money is, if possible, to hollow the model out.
		Clearance To avoid parts fusing when printing, the clearance must be above the minimum clearance*.	Intelligent fill A wire mesh is more than strong enough to do the job of solid fillings with a fraction of the material use.
Non-matching edges With an unequal number of vertices on two connecting edges, it can be interpreted as a hole in the mesh.	Double corners The volume of a mesh must be clearly defined, so a vertex edge or face can only be a part of one shell.	Shrinkage For precision printing it should be taken into account that most materials shrink after printing.	Size Scaling down a model can give surprisingly large material savings. A 20 % smaller cube uses only half as much material.
		Strength To avoid breaking, minimum wall and shell thickness should be obeyed. For parts under more stress extra thickness may be necessary.	Material Materials can be expensive, so if the needs of a project can be fulfilled with a different material that is an easy way to save.
Crossed volumes Volumes cannot intersect, so when two or more volumes cross into each other they must be combined with a boolean operation.	Color prints: For multi-color prints it is important that the 3D model is UV unwrapped correctly over the texture file and that the files are linked correctly.	Details To ensure that details such as engravings or embossments show, minimum detail specifications* should be followed.	3D printing:
		Resolution To avoid visible triangles, the mesh resolution must be high enough according to the print size.	Own 3D printer If you need many 3D prints and want them quickly, it can be a good idea to purchase one.
		*check material specifications at your print service or at the manufacturer of your material.	3D print service To avoid large investments of money and time and to get the best quality, reliability and largest selection of materials, a professional 3D print service is the way to go.

Figure 6.2: Design guidelines for 3D printing [And22]

6.3 Preliminary Design

In this section, we will explore multiple designs for the device. These designs are detailed 3D models of the device that we will use to evaluate their respective designs and assess their feasibility. Each of these preliminary designs will be based on the selected solution from the previous phase. Alongside the models, we will also present the production costs for each of these designs. For a more detailed breakdown of the production costs, please refer to Appendix A.4.

6.3.1 Preliminary Design Variant 2

Figure 6.3 showcases the 3D model of Variant 2, while Figure 6.4 provides various perspectives and body measurements of the device. The key emphasis of this design is its ergonomic shape and user-friendly attributes. With a thickness of 52.2 mm (Figure 6.4b), it successfully balances being slim and accommodat-

ing essential components for optimal performance.

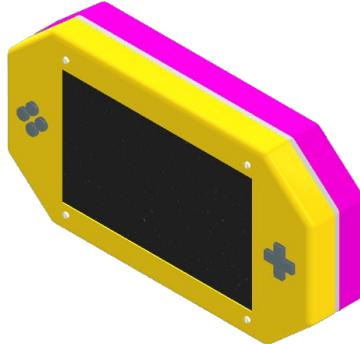


Figure 6.3: Preliminary Design Variant 2

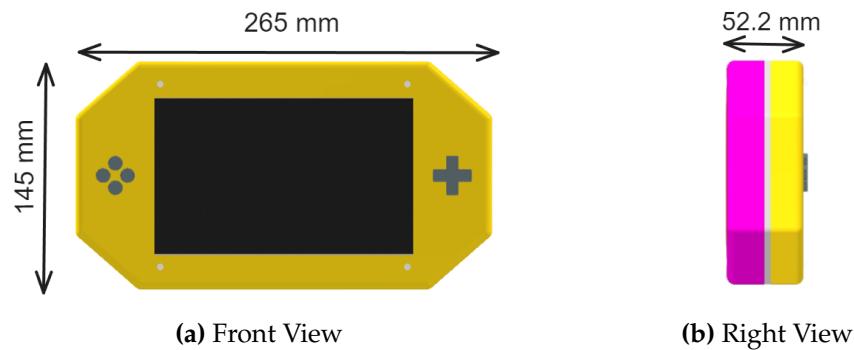


Figure 6.4: Views of Preliminary Design Variant 2

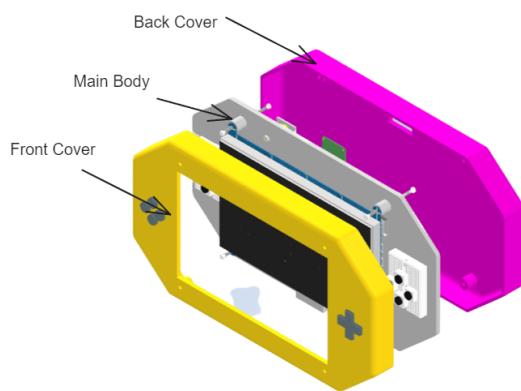


Figure 6.5: Body Components of Preliminary Design Variant 2

The physical design of Solution Variant 2 adheres to a sandwich-like structure comprising a main body, top cover, and back cover (see Figure 6.5). This design

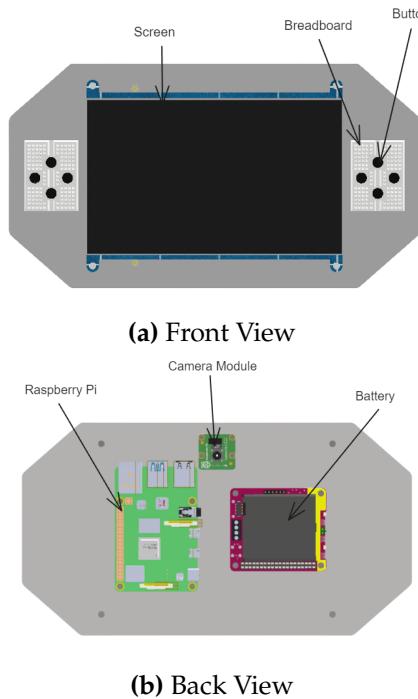


Figure 6.6: Placement of inner components for Variant 2

choice ensures the protection of internal components and simplifies assembly and maintenance. The main body of the device functions as the central hub, accommodating the internal components and features. In contrast, the top and back covers act as protective shields, safeguarding the internal parts from any damage from external factors.

A crucial aspect of the design involves arranging internal components within the device. Following a tablet-like configuration, the main LCD is positioned on the front side of the main body, providing users with a straightforward and interactive interface (see Figure 6.6a). Simultaneously, the camera, Raspberry Pi, and battery were placed on the back side of the body (Figure 6.6b) to optimize the weight distribution and ensure a well-balanced user experience. This arrangement enhances the overall usability and convenience of the device, making it suitable for a wide range of applications.

Ensuring the secure attachment of internal components to the main body is vital in the design process. Various methods of component fastening have been con-



Figure 6.7: Methods to secure 3D-printed components [Her20]

sidered, including direct attachment, threaded inserts, helicoils, side pockets, and bottom pockets, as shown in Figure 6.7.

The most straightforward approach is direct attachment, in which threads are designed into a 3D-printed part to allow components to be screwed in. For more robust connections, threaded inserts can be used by designing holes in the 3D printed part and installing inserts appropriately.

Helicoils offer durable threaded holes by inserting coil-shaped inserts into the holes. Side and bottom pockets create cavities or slots in the 3D-printed part to securely hold components. Each method has its advantages and challenges. After careful evaluation, the variant opts for threaded inserts due to its simplicity and robustness.

Solution Variant 2 employs a hybrid input method combining a touch screen and physical buttons. The touch screen is oriented in landscape mode, while the buttons are positioned on either side of the screen (Figure 6.4a). HDMI and USB connections were established between the touch screen and Raspberry Pi to enable the integration of the touch screen. Additionally, to facilitate the functionality of the physical buttons, they are connected to Raspberry Pi using general-purpose input/output (GPIO) pins.

In Figure 6.8, we observe a quick-release plate designed to be affixed to the tripod stand. To enhance stability during usage, Solution Variant 2 can utilize a quick-release plate, which can be conveniently mounted on a tripod stand.

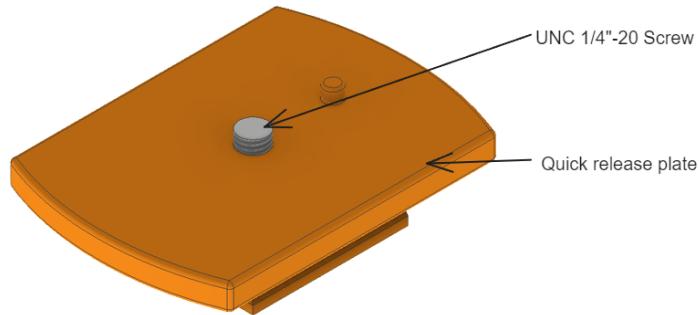


Figure 6.8: Quick release plate

Cost Calculation

Table 6.1 shows the printing cost for each component of this variant, while the total manufacturing cost is shown in Table 6.2. In this calculation, the cost for screen, camera, and Raspberry Pi are not included, as they are used by all variants. The exclusion of these components allows for a more accurate comparison of the manufacturing costs between the variants.

Part Name	Weight of PLA used (g)	Printing Time (h)	Material Cost	Energy Cost	Total Cost
Base	175.07	5.13	5.25 €	0.10 €	5.35 €
Top Cover	105.85	3.32	3.17 €	0.06 €	3.24 €
Back Cover 1	86.73	2.57	2.60 €	0.05 €	2.65 €
Back Cover 2	86.60	2.53	2.60 €	0.05 €	2.64 €
Total	454.25	13.55	13.62 €	0.26 €	13.88 €

Table 6.1: Printing cost for Variant 2

Parts Name	Amount	Price	Remarks	Total Price
Printing Cost	1	13.90 €	Calculated	13.90 €
Screw M2x10mm (DIN 84)	4	0.05 €	Wuerth	0.20 €
Screw M2.5x10mm (DIN 84)	8	0.05 €	Wuerth	0.40 €
Screw M2.5x18mm (DIN 84)	8	0.11 €	Wuerth	0.88 €
Screw M3x10mm (DIN 84)	2	0.05 €	Wuerth	0.10 €
Nut M2 (DIN 934)	4	0.04 €	Wuerth	0.15 €
Standoff M2	4	0.17 €	Amazon	0.67 €
Threaded Inserts M2.5	16	0.50 €	Ruthex	7.99 €
Threaded Inserts 1/4"	1	0.00 €	Ruthex	0.00 €
Breadboard	2	0.04 €	Amazon	0.08 €
Tactile Button	8	0.00 €	Amazon	0.00 €
Female MicroUsb	1	4.95 €	Adafruit	4.95 €
PiJuice	1	45.49 €	Rasppishop	45.49 €
Total				7479 €

Table 6.2: Manufacturing cost for Variant 2

6.3.2 Preliminary Design Variant 3

Variant 3 maintains a similar component arrangement to variant 2, with the screen at the front and the camera, Raspberry Pi, and battery at the rear, as shown in Figure 6.10. However, Variant 3 introduced significant changes with different screen orientations and battery types.

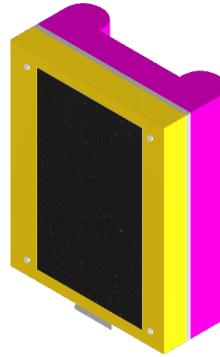


Figure 6.9: Preliminary Design Variant 3

A noteworthy alteration is the inclusion of bumps on the back cover to enhance the ergonomics (Figure 6.11). This adjustment aims to provide a more comfortable grip, improve user engagement, and extend usability. In addition, the tactile bump is added to increase handling stability of the device, ensuring that

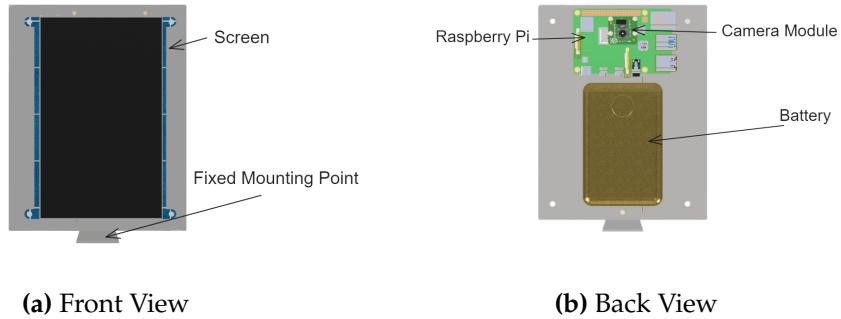


Figure 6.10: Placement of inner components for Variant 3

the device fits better in the user's hand, further enhancing the overall user experience.

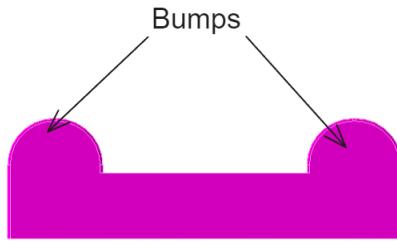


Figure 6.11: Bumps on the back cover

Variant 3 shifted from the standard battery position in variant 2, with a more noticeable difference in battery placement. Figure 6.12 illustrates a designated slot within the back cover, strategically designed to house a power bank outside the body. This configuration enhances the operational stability and simplifies the battery replacement process.

The input methodology was streamlined using a touch screen as the sole interface. This approach simplifies the user experience by eliminating the need for physical buttons and seamlessly integrating screen interactions. Additional information regarding integrating the touchscreen with the Raspberry Pi is provided in Section 6.3.1.

Figure 6.10a shows the position of the mounting point of on the main body in Variant 3. This strategic design allows the mounting point to serve as a sturdy

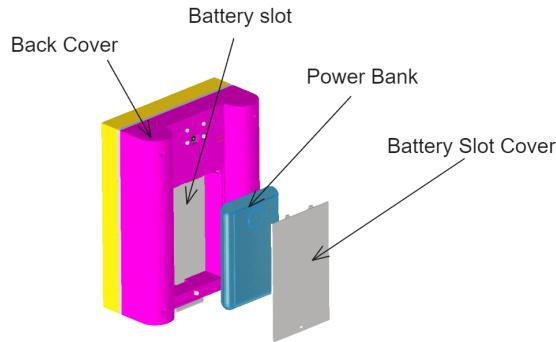


Figure 6.12: Battery Placement

anchor for the device when used in a tripod stand.

Cost Calculation

Table 6.3 shows the printing cost for each component of this variant, while the total manufacturing cost is shown in Table 6.4.

Part Name	Weight of PLA used (g)	Printing Time (h)	Material Cost	Energy Cost	Total Cost
Base	222.96	6.67	6.69 €	0.13 €	6.81 €
Top Cover	71.02	2.08	2.13 €	0.04 €	2.17 €
Back Cover	303.02	9.37	9.09 €	0.18 €	9.26 €
Battery Cover	17.43	0.53	0.52 €	0.01 €	0.53 €
Total	614.43	18.65	18.43 €	0.35 €	18.78 €

Table 6.3: Printing cost for Variant 3

Parts Name	Amount	Price	Remarks	Total Price
Printing Cost	1	18.78 €	Calculated	18.78 €
Screw M2x10mm (DIN 84)	4	0.05 €	Wuerth	0.20 €
Screw M2.5x10mm (DIN 84)	4	0.05 €	Wuerth	0.20 €
Screw M2.5x18mm (DIN 84)	8	0.11 €	Wuerth	0.88 €
Nut M2 (DIN 934)	4	0.04 €	Wuerth	0.15 €
Threaded Inserts M2.5	12	0.13 €	Ruthex	1.54 €
Veektomx	1	25.99 €	Amazon	25.99 €
Total				48.08 €

Table 6.4: Manufacturing cost for Variant 3

6.3.3 Preliminary Design Variant 6

Figure 6.13 provides a detailed and comprehensive overview of the initial design concept of Variant 6. This version stands out by organizing its internal components into a configuration resembling a point-of-service (POS) system. This change from previous iterations purposefully positions the screen at an inclined angle, enhancing user interaction and optimizing the screen visibility (Figure 6.14).

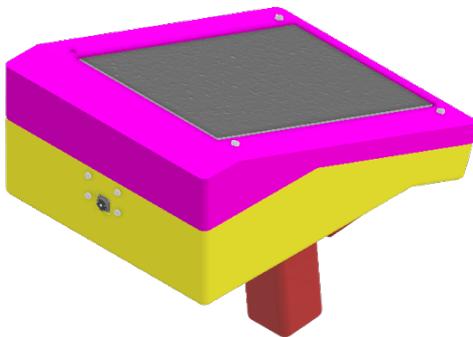


Figure 6.13: Preliminary Design Variant 6

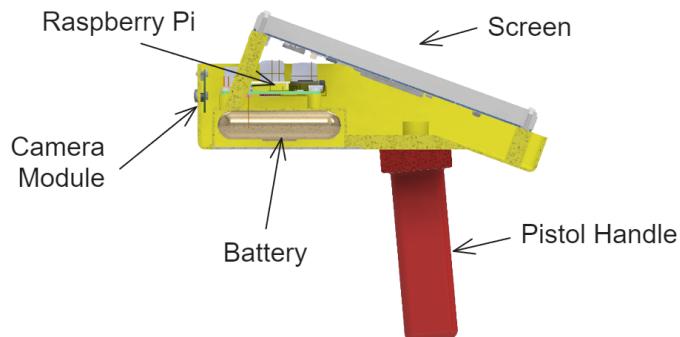


Figure 6.14: Placement of inner components for Variant 3

Figure 6.15 demonstrates the handle grip design, while Figure 6.16a illustrates its placement on the main body. This ergonomic addition ensures a secure and comfortable hold during operation. Additionally, the handling of the device can be easily switched between the quick-change plate and the handle grip, providing users with flexible options for different scenarios (see Figure 6.16b).



Figure 6.15: Handle Grip



(a) Handle Grip with Main Body **(b)** Quick Release Plate with Main Body

Figure 6.16: Placement of handle grip and quick release plate

This variant boasts the same input method and battery placement as Variant 3. Please refer to Section 6.3.2 for a comprehensive explanation.

Cost Calculation

Table 6.5 shows the printing cost for each component of this variant, while the total manufacturing cost is shown in Table 6.6.

Part Name	Weight of PLA used (g)	Printing Time (h)	Material Cost	Energy Cost	Total Cost
Top Cover	55.35	1.87	1.66 €	0.04 €	1.70 €
Main Body	240.25	8.88	7.21 €	0.17 €	7.37 €
Battery Cover	22.21	0.67	0.67 €	0.01 €	0.68 €
Handle Pistol	57.30	1.73	1.72 €	0.03 €	1.75 €
Total	375.11	13.15	11.25 €	0.25 €	11.50 €

Table 6.5: Printing cost for Variant 6

Parts Name	Amount	Price	Remarks	Total Price
Printing Cost	1	11.50 €	Calculated	11.50 €
Screw M2x10mm (DIN 84)	4	0.05 €	Wuerth	0.20 €
Screw M2.5x10mm (DIN 84)	5	0.05 €	Wuerth	0.25 €
Screw M2.5x18mm (DIN 84)	4	0.11 €	Wuerth	0.44 €
Nut M2 (DIN 934)	4	0.04 €	Wuerth	0.15 €
Threaded Inserts M2.5	8	0.13 €	Ruthex	1.03 €
Threaded Inserts 1/4"	3	0.50 €	Ruthex	1.50 €
1/4" Schraube (Camera)	2	1.40 €	Amazon	2.80 €
Vekktomx	1	25.99 €	Amazon	25.99 €
Total				43.84 €

Table 6.6: Manufacturing cost for Variant 6

6.3.4 Preliminary Design Variant 7

In Figure 6.17, we can observe a 3D model for variant 7, which draws inspiration from handheld PCs. The design showcases the integration of Raspberry Pi at the back of the screen, creating a compact and unified structure, while the battery is positioned next to the screen.



Figure 6.17: Preliminary Design Variant 7

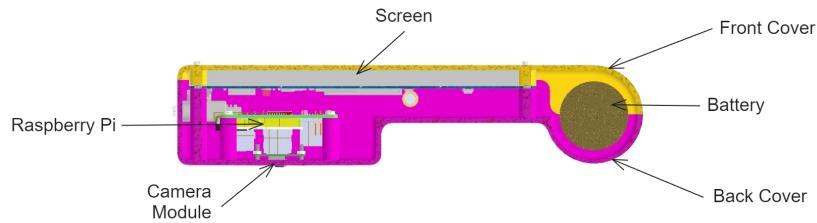


Figure 6.18: Placement of inner components for Variant 7

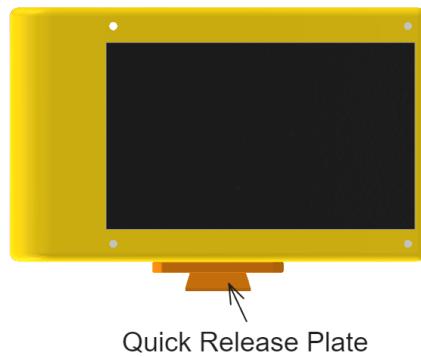


Figure 6.19: Placement of quick release plate

This design includes a bump on the side, enhancing ergonomics as shown in Figure 6.18. As with variants 2 and 6, this variant employs a quick-release plate, facilitating integration with a tripod stand. The placement of the plate can be observed in Figure 6.19.

Cost Calculation

Table 6.7 shows the printing cost for each component of this variant, while the total manufacturing cost is shown in Table 6.8.

Part Name	Weight of PLA used (g)	Printing Time (h)	Material Cost	Energy Cost	Total Cost
Top Cover	61.50	2.08	1.84 €	0.04 €	1.88 €
Back Cover	380.70	11.82	11.42 €	0.22 €	11.64 €
Total	442.20	13.90	13.26 €	0.26 €	13.52 €

Table 6.7: Printing cost for Variant 7

Parts Name	Amount	Price	Remarks	Total Price
Printing Cost	1	13.52 €	Calculated	13.52 €
Screw M2x10mm (DIN 84)	4	0.05 €	Wuerth	0.20 €
Screw M2.5x10mm (DIN 84)	4	0.05 €	Wuerth	0.20 €
Screw M2.5x18mm (DIN 84)	4	0.11 €	Wuerth	0.44 €
Screw M3x10mm (DIN 84)	2	0.05 €	Wuerth	0.10 €
Nut M2 (DIN 934)	4	0.04 €	Wuerth	0.15 €
Threaded Inserts M2.5	8	0.13 €	Ruthex	1.03 €
Threaded Inserts 1/4"	1	0.50 €	Ruthex	0.50 €
Cylinder Power Bank	1	19.99 €	Amazon	19.99 €
Female MicroUsb	1	4.95 €	Adafruit	4.95 €
Total				41.07 €

Table 6.8: Manufacturing cost for Variant 7

6.4 Evaluation with VDI 2225

This section will evaluate the preliminary design variants using the guideline VDI 2225 [Pah07, 109-110]. This guideline is a comprehensive framework for evaluating technical solutions based on a balanced consideration of various aspects.

The guideline advocates for comprehensive evaluation methods covering task-specific requirements and general constraints. These methods aim to quantify and qualitatively assess the properties of different variants, even in the early conceptual phase where information is limited.

The evaluation process, as discussed by Pahl and Beitz [Pah07, 110-124], involves several key steps:

Identifying Evaluation Criteria

This initial step involves defining a set of objectives from which specific evaluation criteria can be derived. These objectives should comprehensively cover decision-relevant requirements and general constraints, ensuring no crucial criteria are overlooked. The objectives should also be as independent as possible and expressed in quantitative or qualitative terms.

The following are the evaluation criteria for the preliminary design variants:

Weight Distribution: The weight distribution is evaluated by measuring the distance between the center of gravity and the point of handling. The point of handling is defined as the point where the user holds the device. A lower distance signifies a more balanced weight distribution, enhancing the user experience. The position of the center of gravity is retrieved from Computer-Aided Design (CAD) models through detailed analysis of the device's structural layout and component placement.

Device Weight: Device weight evaluates the overall heaviness of the equipment. A lighter device is generally easier to handle and transport, reducing user fatigue and enabling greater mobility while maintaining performance and durability. The value for device weight is calculated from CAD models by summing the individual weights of all components, materials, and structural elements that constitute the device.

Device Size: The size criterion considers the device's physical dimensions, assessing its compactness and portability. An optimal device size allows for convenient storage, transportation, and operation in various environments without compromising functionality. The evaluation of device size involves measuring key dimensions such as length, width, height, and any protrusions or extensions.

Ease of Assembly: This criterion evaluates the ease of assembling and disassembling the device. Quick and easy assembly and disassembly saves time and increases user convenience, reducing the risk of errors. Evaluation is done by counting the components used in assembly and disassembly. Fewer components often mean a more straightforward and more user-friendly design. The type and number of fasteners, such as screws or connectors, needed for assembly are also considered.

Swappable Parts: Swappable components refer to the ease with which parts can be interchanged or substituted. This design enhances flexibility, maintenance, and adaptability. The presence of swappable parts encourages component modularity, enabling streamlined repairs and upgrades. Swappable parts are assessed based on the quantity of interchangeable components and their compatibility. A higher number of swappable parts signifies a design that sup-

ports versatility and minimizes downtime for maintenance or repairs.

Weighting Evaluation Criteria

After establishing the evaluation criteria, their relative importance is assessed through weighting factors (w). This step is crucial in eliminating less significant criteria before the actual evaluation. Weightings should reflect the relative importance of each evaluation criterion.

Guideline VDI 2225 aims to avoid weightings and instead relies on criteria of roughly equal importance. However, weightings (like 2x or 3x) are used when there are significant differences between criteria. Table 6.9 shows the weighting factors for the evaluation criteria.

Criteria	Weighting Factor, w
Weight Distribution	3x
Device Weight	2x
Device Size	1x
Ease of Assembly	1x
Swappable Parts	2x

Table 6.9: Weighting Factors for Evaluation Criteria

Assessing Values

This step involves assigning values (v_{ij}) to the variants based on the relative scale of the determined parameters. Guideline VDI 2225 suggests using a range from 0 to 4. Table 6.10 shows the scale used to evaluate the preliminary design variants. Tables 6.11 to 6.15 show the value scales for the individual evaluation criteria. Equation 6.1 shows the formula used to calculate the weighted value (wv_{ij}) for each variant.

Points, v_{ij}	Meaning
0	unsatisfactory
1	just tolerable
2	adequate
3	good
4	very good (ideal)

Table 6.10: Value Scale for Evaluation [Pah07, 115]

Weight Distribution	
Range, mm	Point, v_{ij}
0-10	4
10-50	3
50-100	2
100-150	1
≥ 150	0

Table 6.11: Value Scale for Weight Distribution

Device Weight	
Range, g	Point, v_{ij}
0-500	4
500-1000	3
1000-1500	2
1500-2000	1
≥ 2000	0

Table 6.12: Value Scale for Device Weight

Device Size	
Range, mm	Point, v_{ij}
0-100	4
100-200	3
200-300	2
300-400	1
≥ 400	0

Table 6.13: Value Scale for Device Size

Ease of Assembly	
Range	Point, v_{ij}
0-25	4
25-50	3
50-75	2
75-100	1
≥ 100	0

Table 6.14: Value Scale for Ease of Assembly

Swappable Parts	
Range	Point, v_{ij}
≥ 4	4
3	3
2	2
1	1
0	0

Table 6.15: Value Scale for Swappable Parts

$$wv_{ij} = w_i \cdot v_{ij} \quad (6.1)$$

Determining the Overall Value

The overall value of each variant (OWV_j) is calculated by summing the weighted values (wv_{ij}) of all evaluation criteria (see Equation 6.2).

$$OWV_j = \sum_{i=1}^n wv_{ij} \quad (6.2)$$

Comparing Concept Variants

With the overall values (OWV_j) of the concept variants, the variants can be compared and evaluated based on their rating (R) which is calculated using Equation 6.3. The technical rating (R_t) is calculated using Equation 6.4, where v_{max}

is the maximum value of the value scale, and n is the number of evaluation criteria.

The economic rating (R_e) is calculated using Equation 6.5, where C_o is the comparative cost, and $C_{variant}$ is the cost of the variant. For this project, C_o is set to be 60% of the cost of the least expensive variant (see Equation 6.6).

The best variant is determined by comparing each variant's total rating (R). The variant with the highest total rating is considered the best variant.

$$R = \frac{R_t + R_e}{2} \quad (6.3)$$

$$R_t = \frac{OWV_j}{v_{max} \cdot \sum_{i=1}^n w_i} \quad (6.4)$$

$$R_e = \frac{C_o}{C_{variant}} \quad (6.5)$$

$$C_o = 0.6 \cdot C_{minimum} \quad (6.6)$$

6.4.1 Evaluation of Preliminary Design Variant

The result of the evaluation of the preliminary design variants will be presented in this section. Table 6.16 and Table 6.17 shows the technical evaluation of the preliminary design variants, while Table 6.18 shows the economic evaluation of the variants. The total rating of the variants is shown in Table 6.19. Appendix A.5 shows the detailed calculation of the evaluation.

Based on the total rating, Variant 6 is the best variant, followed by Variant 7, Variant 3, and Variant 2.

Evaluation criteria					Variant 2			Variant 3		
No.	Criteria	Weight	Description	Units	Value	Point	Weighted Weight	Value	Point	Weighted Weight
1	Weight Distribution	3	Distance of center of gravity	mm	2.49	4	12	54.34	2	6
2	Device Weight	2	Weight of device	g	1190.60	2	4	1103.30	2	4
3	Device Size	1	Length of device	mm	265.00	2	2	195.00	3	3
		1	Width of device	mm	145.00	3	3	145.00	3	3
		1	Thickness of device	mm	52.20	4	4	69.20	4	4
4	Ease of Assembly	1	Number of parts required to be assemble	-	58	2	2	42	3	3
5	Swappable parts	2	Number of swappable parts available	-	1	1	2	1	1	2
Total		11				29				25
Technical Rating, Rt						0.659				0.568

Table 6.16: Technical Evaluation of Preliminary Design Variants (1/2)

Evaluation criteria					Variant 6			Variant 7		
No.	Criteria	Weight	Description	Units	Value	Point	Weighted Weight	Value	Point	Weighted Weight
1	Weight Distribution	3	Distance of center of gravity	mm	28.09	3	9	92.48	2	6
2	Device Weight	2	Weight of device	g	1112.60	2	4	889.20	3	6
3	Device Size	1	Length of device	mm	198.77	3	3	222.50	2	2
		1	Width of device	mm	145.00	3	3	135.50	3	3
		1	Thickness of device	mm	92.63	4	4	47.70	4	4
4	Ease of Assembly	1	Number of parts required to be assemble	-	49.00	3	3	35	3	3
5	Swappable parts	2	Number of swappable parts available	-	3.00	3	6	1	1	2
Total		11				32				26
Technical Rating, Rt						0.727				0.591

Table 6.17: Technical Evaluation of Preliminary Design Variants (2/2)

Production Cost		
Variant	Cost, $C_{variant}$ (€)	Economic Rating, R_e
Variant 2	74.79	0.33
Variant 3	48.08	0.52
Variant 6	43.84	0.56
Variant 7	41.07	0.60

Table 6.18: Economic Evaluation of Preliminary Design Variants

Variant	Technical Rating, R_t	Economic Rating, R_e	Total Rating, R
Variant 2	0.66	0.33	0.494
Variant 3	0.57	0.51	0.540
Variant 6	0.73	0.64	0.645
Variant 7	0.59	0.60	0.596

Table 6.19: Total Rating of Preliminary Design Variants

7 Detail Design

Detail design involves finalizing the specifications of individual components, including their shapes, dimensions, materials, and surface properties [Pah07, 436]. Figure 7.1 shows the steps involved in the detail design process.

Additionally, it encompasses the creation of production documents such as detailed component drawings and assembly instructions [Pah07, 436], which are usually accomplished using Computer-Aided Design (CAD) software.

The evaluation of the preliminary design variants shows that Variant 6 is the best. Any improvements will be added to the design, while any weaknesses will be addressed. The result of this process is the final design of the device. The detailed drawing of the device is shown in Appendix A.2.

Power Switch

This component is critical in controlling the Raspberry Pi's power supply. It is imperative to have a reliable method for powering up and shutting down the Raspberry Pi to ensure smooth operation and prevent potential data corruption.

One available method utilizes the GPIO pins to initiate a shutdown sequence for the Raspberry Pi [Lab21]. While effective in bringing the device to a hibernation state, it is essential to note that this method does not completely cut off power. As a result, the Raspberry Pi still draws a minimal amount of power even in this low-power state [jdb].

A more straightforward approach is recommended to achieve more efficient power management, which involves the implementation of a simple physical push button as switch (refer to Figure 7.2). This implementation directly connects and disconnects the power supply to the Raspberry Pi. As a result, when

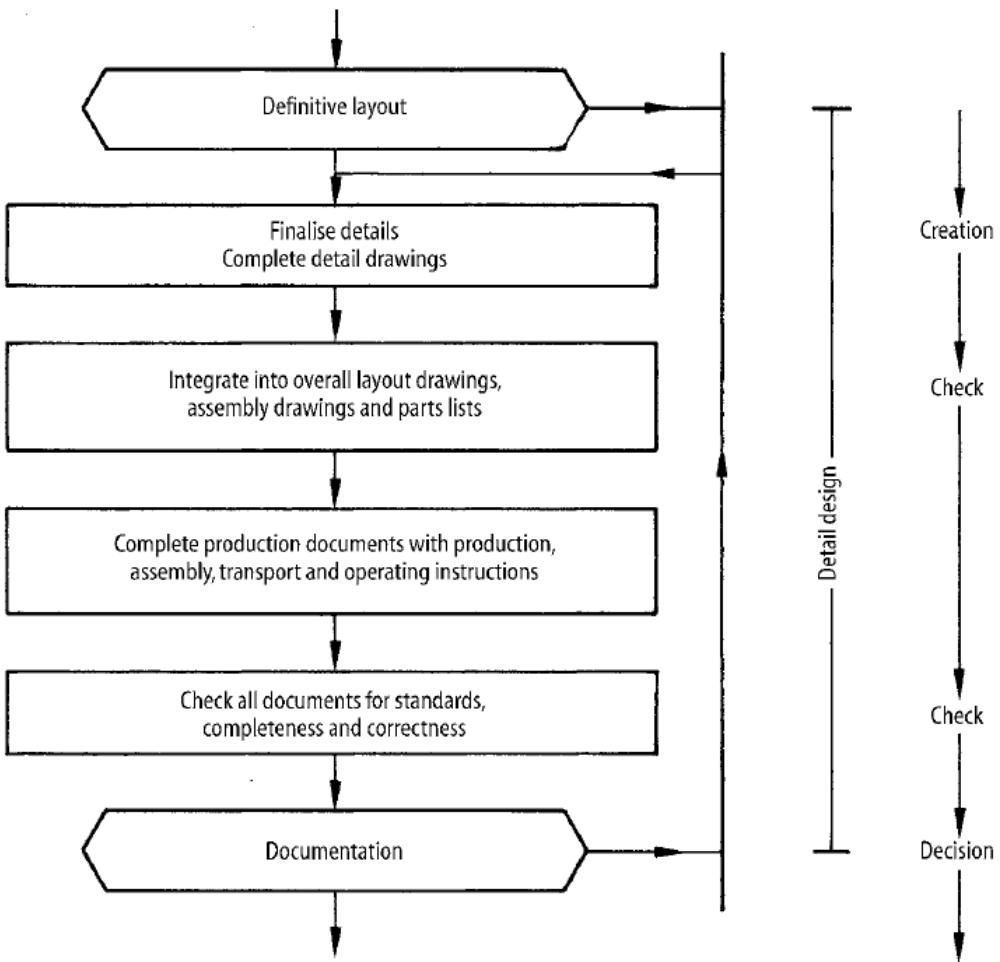


Figure 7.1: Steps in the detail design process

the switch is in the *off* position, it completely severs the power supply, ensuring that the Raspberry Pi consumes no power whatsoever.

Camera Protection

Figure 7.3 shows the original position of the camera component. As seen in the figure, the camera is seamlessly integrated into the device's body, with the lens protruding slightly.

However, this design does pose a potential risk. If the device is inadvertently placed with the lens side facing a surface, it could get damaged, severely affect-

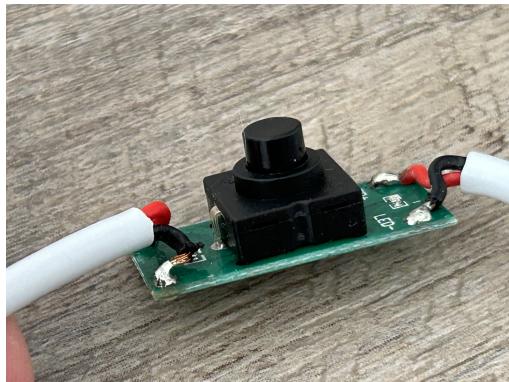


Figure 7.2: Power Switch

ing its performance.

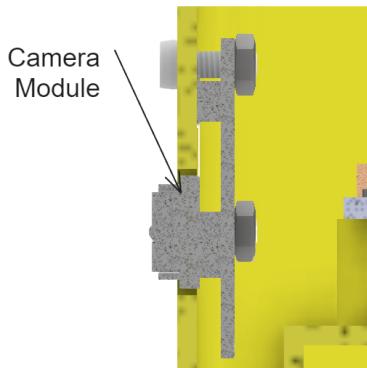


Figure 7.3: Position of the camera component

To address this concern, we have incorporated a 3 mm high bump (as seen in Figure 7.4) as a preventative measure. The bump is strategically positioned to lift the camera lens above surfaces, preventing direct contact.

Screen Protection

Similar to the camera, the screen of the device also requires protection. Similar to the camera protection, a protective bump has been incorporated around the screen perimeter to address this issue. Refer to Figure 7.5 for a visual representation.

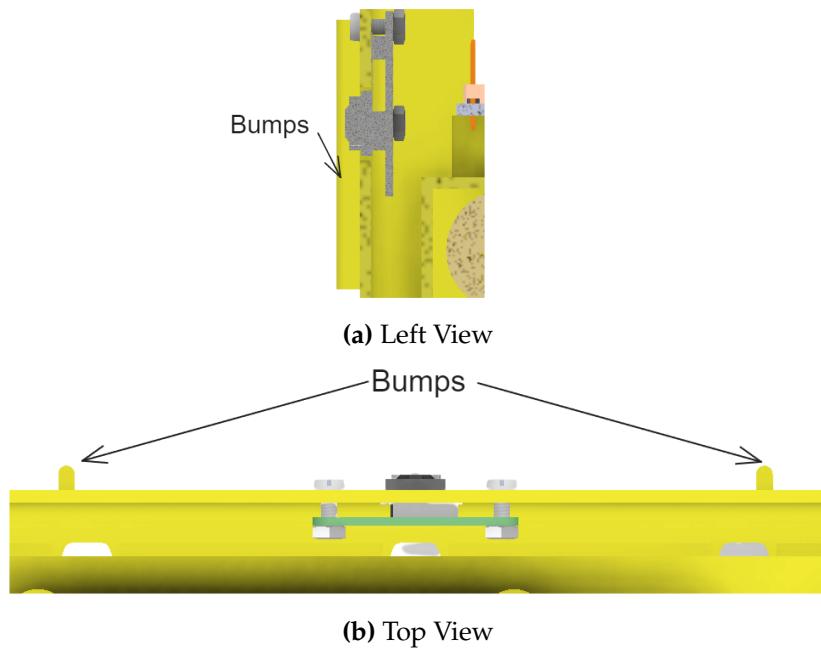


Figure 7.4: Protective bump for camera

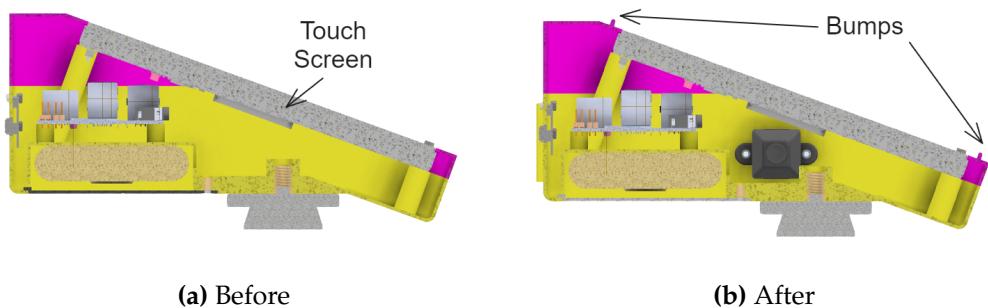


Figure 7.5: Protective bump for screen

Column for Threaded Inserts

Previously, in Section 6.3.1, we delved into utilizing threaded inserts alongside screws to firmly attach components to the body. It is crucial to note that distinct sizes of threaded inserts necessitate particular minimum wall thicknesses for the columns and hole depths to guarantee adequate engagement and steadiness. A sizing guide is provided by Ruthex threaded inserts, as shown in Table 7.1.

Thread Size	Hole size (mm)	Min. thickness (mm)	Min. height (mm)
M2.5	4	1.6	6.7
1/4"	8	3.3	13.7

Table 7.1: Sizing guide for threaded inserts [ruta][rutb]

LAN Port

To enable easier maintenance of the Raspberry Pi, a Local Area Network (LAN) port is added to the device, which enables the Raspberry Pi to be accessed directly without disassembly. This strategic integration allows seamless connectivity, enabling direct access to the Raspberry Pi's functionalities and resources over a local network. Figure 7.6 shows the LAN port.

The LAN port, illustrated in Figure 7.6, is positioned on the side of the device (as shown in Figure 7.7), making it easy for users to perform maintenance tasks.

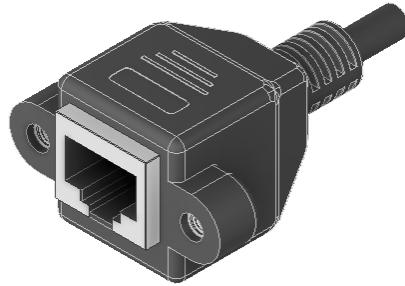


Figure 7.6: The LAN port

Color Scheme

The selection of colors is crucial in making the product visually appealing, especially since the target market is the police force. To achieve this, we utilized the German police logo for inspiration, as shown in Figure 7.8. The predominant color of blue in the logo is used as the primary color for the device. Additionally, we used yellow from the logo as the color for the handle grip and white for the device's top cover. Figure 7.9 shows the device preview with the recolor.



Figure 7.7: Position of the LAN port



Figure 7.8: Germany Police Logo [bun]

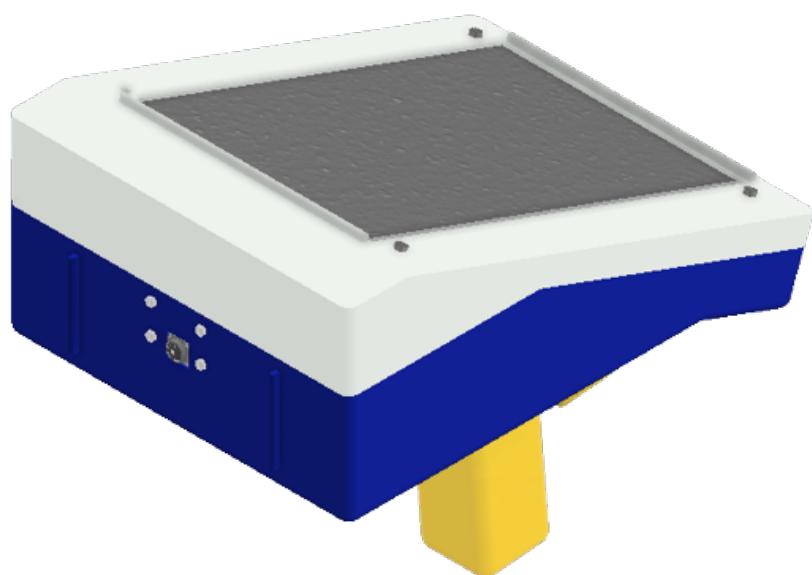


Figure 7.9: Result of recolor

8 Printing and Assembly

8.1 Printing

In this section, we will describe the printing process for the prototype. This process was carried out at the Workshop of the University of Applied Sciences Brandenburg using the printer described in Section 3.2.1. The following parameters are utilized for the process.

- Layer Height: 0.2 mm
- Infill Density: 15 %
- Print Speed: 60 mm/s
- Supports: Everywhere
- Filament: PLA

Table 8.1 provides information on the printing time and the weight of filament used. Additionally, Figure 8.1 showcases the final printed parts after removing the support materials.

Part Name	Weight of PLA used (g)	Printing Time (h)
Top Cover	57.71	2.00
Main Body	245.14	9.25
Battery Cover	22.21	0.67
Switch Cover	1.31	0.05
Handle Pistol	64.63	1.98

Table 8.1: Printing Time and Filament Used

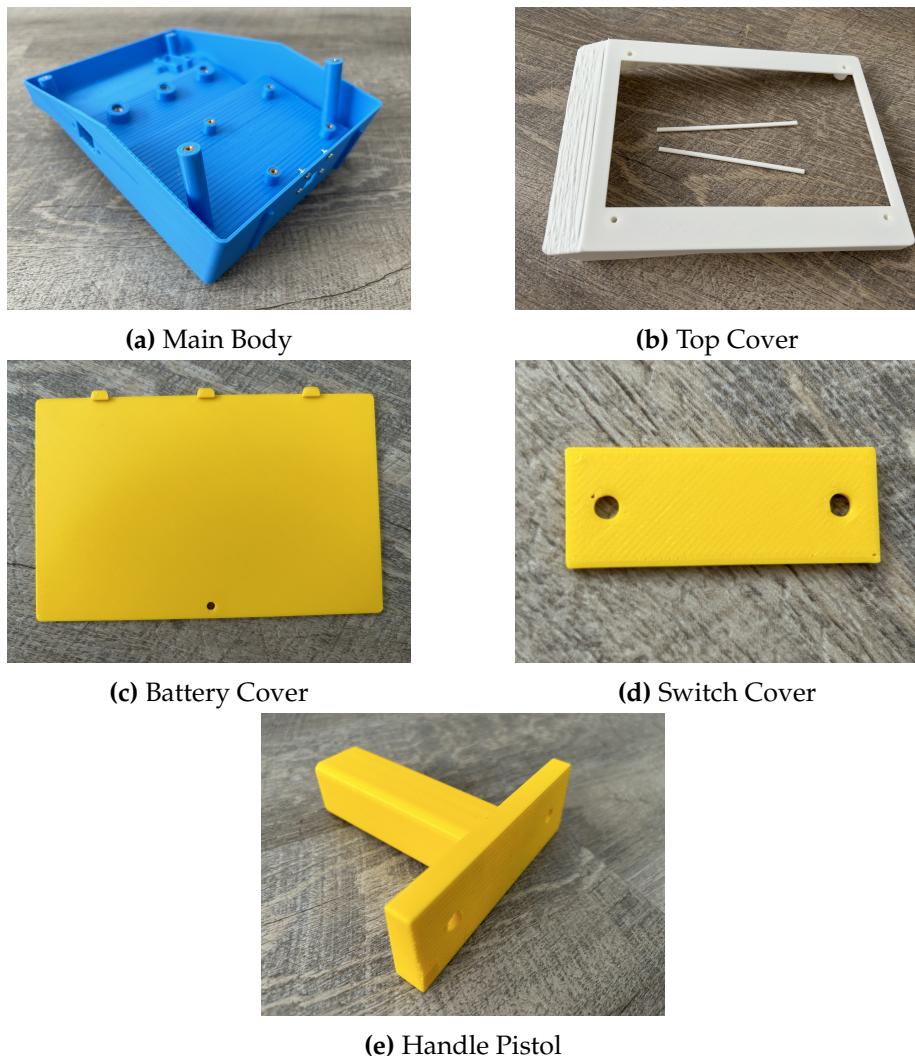


Figure 8.1: Printed Parts

8.2 Assembly

The assembly process is done by following the steps below:

Step 1: Installation of Threaded Inserts

In order to securely install the brass threaded inserts into the main body, it is recommended to use a soldering iron [Her23]. Begin by placing the chosen

threaded insert onto the targeted position, aligning it with the desired hole. Heat the soldering iron to a suitable temperature, ranging from 225 °C to 245 °C for PLA material.

Once the soldering iron has reached the appropriate temperature, apply it gently to the top of the threaded insert. This process will transmit controlled heat into the material, causing the wall to soften and allow the threaded insert to be inserted. For a visual representation of how a threaded insert should look once correctly installed into the main body, please refer to Figure 8.2.



Figure 8.2: The installed threaded insert

Step 2: Installation of Switch

Installing a switch to the main body is a straightforward process that requires a few basic materials: the switch itself, a switch cover, two M2.5 nuts, and M2.5 screws. Position the switch inside the designated switch holder (see Figure 8.3), ensuring that the button faces outward for easy access.

Next, the switch cover is placed on top, aligning it with the switch and the corresponding holes in the main body. Once aligned, the M2.5 screws and nuts secure the switch and the cover to the main body. Figure 8.4 shows the completed installation of the switch.

Step 3: Installation of LAN Port

This step begins by locating the slot of the LAN port on the main body, which is located on the right side of the main body (see Figure 8.5). The LAN port is

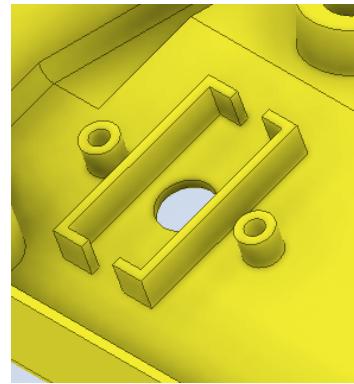


Figure 8.3: The Switch Holder

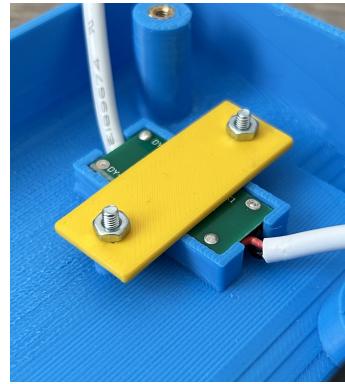


Figure 8.4: The installed switch

inserted into the slot and secured using the M3 screws. Figure 8.6 shows the completed installation of the LAN port.

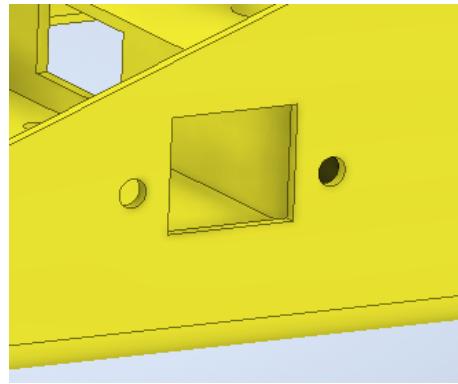


Figure 8.5: The LAN Port Slot

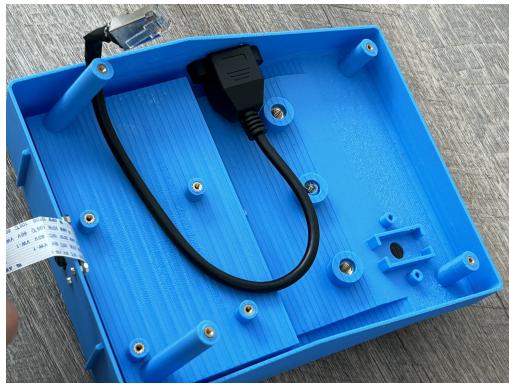


Figure 8.6: The installed LAN port

Step 4: Installation of Camera Module

The camera module is installed to the main body using M2 screws. The camera module is placed in the designated slot on the main body (see Figure 8.7). The M2 screws are then used to secure the camera module to the main body. Figure 8.8 shows the completed installation of the camera module.



Figure 8.7: The Camera Module Slot

Step 5: Installation of Battery

To start the installation process, insert the battery into the holder as shown in Figure 8.9. Then, connect the battery to the switch with a 90-degree USB-A to USB-C connector.

To fasten the battery to the main body, place the battery cover over the battery

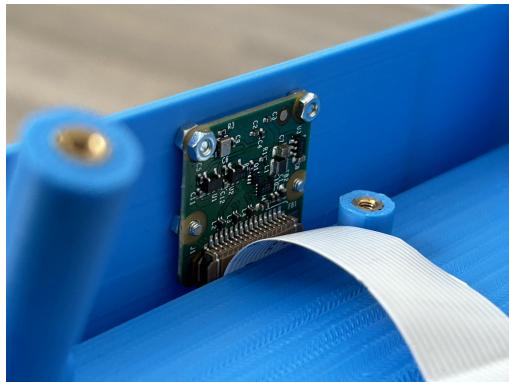


Figure 8.8: The installed camera module

and main body. Use the M2.5 screws to secure the battery cover to the main body. The finished battery installation can be seen in Figure 8.10.

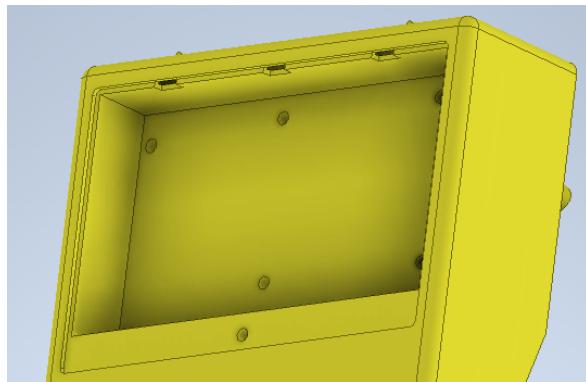


Figure 8.9: The Battery Holder

Step 6: Installation of Raspberry Pi

The Raspberry Pi is installed on the main body by using the M2.5 screws. The Raspberry Pi is placed in the designated slot on the main body (see Figure 8.11). The M2.5 screws are then used to secure the Raspberry Pi to the main body.

Next, the following connections are made to the Raspberry Pi:

- The LAN port is connected to the Raspberry Pi via a LAN cable.
- The camera module is connected to the Raspberry Pi via a ribbon cable.
- The switch is connected to the Raspberry Pi via a USB-C cable.



(a) Without Battery Cover

(b) With Battery Cover

Figure 8.10: The installed battery

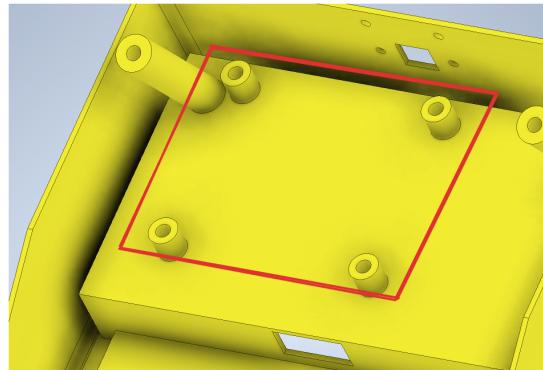


Figure 8.11: The Raspberry Pi Slot

Step 7: Installation of Screen and Top Cover

The final step is to install the screen and the top cover. Begin by placing the screen into the designated slot on the main body (see Figure 8.12) and align the hole on the screen with the hole on the main body. Next, the top cover is placed on top of the main body. The M2.5 screws are then used to secure the top cover to the main body. Figure 8.13 shows the completed installation of the screen and the top cover.

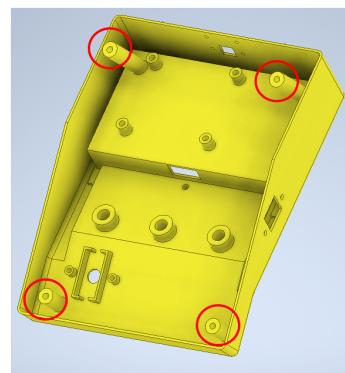


Figure 8.12: The Screen Slot



Figure 8.13: The installed screen and top cover

8.3 Final Product

Figure 8.14 shows the final product. The total cost of building the product including the cost of printing and all of the materials is shown in Table 8.2 and Table 8.3 respectively.



Figure 8.14: The Final Product

Part Name	Material Cost	Energy Cost	Total Cost
Top Cover	1.73 €	0.04 €	1.77 €
Main Body	7.35 €	0.17 €	7.53 €
Battery Cover	0.67 €	0.01 €	0.68 €
Switch Cover	0.04 €	0.00 €	0.04 €
Handle Pistol	1.94 €	0.04 €	1.98 €
Total	11.73 €	0.26 €	11.99 €

Table 8.2: Total Printing Cost

Parts Name	Amount	Price	Remarks	Total Price
Printing Cost	1	11.99 €	Calculated	11.99 €
RaspberryPi 4B 2GB	1	35.00 €	RaspberryPi	35.00 €
Camera Module v2	1	25.00 €	RaspberryPi	25.00 €
Waveshare 7inch screen	1	53.99 €	Waveshare	53.99 €
Veektomx Battery	1	25.99 €	Amazon	25.99 €
Screw M2x10mm (DIN 84)	4	0.05 €	Wuerth	0.20 €
Screw M2.5x10mm (DIN 84)	7	0.05 €	Wuerth	0.35 €
Screw M2.5x18mm (DIN 84)	4	0.11 €	Wuerth	0.44 €
Screw M3x10mm (DIN 84)	2	0.05 €	Wuerth	0.10 €
Nut M2 (DIN 934)	4	0.04 €	Wuerth	0.15 €
Nut M2.5 (DIN 934)	2	0.04 €	Wuerth	0.07 €
Threaded Inserts M2.5	9	0.13 €	Ruthex	1.16 €
Threaded Inserts 1/4"	3	0.50 €	Ruthex	1.50 €
1/4" Camera Screw	2	1.40 €	Amazon	2.80 €
LAN Port	1	5.67 €	Aliexpress	5.67 €
Switch Pi	1	3.90 €	Amazon	3.90 €
90-Degree USB-A to USB-C	1	2.75 €	Amazon	2.75 €
90-Degree USB-A to micro USB	1	5.99 €	Amazon	5.99 €
90-Degree HDMI	1	2.12 €	Aliexpress	2.12 €
Total				168.29 €

Table 8.3: Total Material Cost

9 Conclusion

In summary, this research has developed and demonstrated a highly promising handheld device engineered to measure vehicle speed precisely. This prototype is a cost-effective alternative to the existing speed measurement tools, particularly the conventional speed pistol. By integrating cheap computational components, high-quality cameras, and the power of 3D printing, we propose an affordable, highly accurate, and reliable device.

The design process of the prototype was carried out based on VDI guideline 2221 and divided into four parts: task clarification, conceptual design, embodiment design, and detail design. The task clarification phase was conducted by identifying the problem, defining the requirements, and setting the specifications.

The conceptual design phase was carried out by defining function and function structure. The function structure was then used to generate working principles with the help of brainstorming and analysis of existing technical systems. The result of idea generation is presented inside Zwicky's morphological box. The working principles are then combined to form eight different working structures. After careful evaluation, only four were selected for further development.

In the embodiment design phase, the four working structures were developed further by defining their 3D model with Autodesk Inventor. With the help of the 3D model, the working structures were evaluated based on their physical properties, manufacturability, and ergonomics. The estimated cost of manufacturing each variant is also calculated and compared. The evaluation results show that variant 6 is the most suitable to be chosen as the final design.

The detailed design phase was carried out by developing the final design of

Conclusion

the prototype based on the chosen variant. In this phase, the 3D model of the prototype was developed further by adding more details and components. The 3D model was then used to generate the technical drawing of the prototype.

The final design was then manufactured using 3D printing technology. The prototype was assembled and tested to ensure it worked as intended. The test result shows that the prototype can securely hold the inner components properly and protect them. In terms of ergonomics, the handling of the device seems stable and lightweight.

The prototype of the speed pistol has been developed at a remarkably low cost, totaling only 168.29 €. This cost is significantly lower, approximately 90%, compared to the conventional speed pistol discussed in Chapter 2. However, it is essential to note that this prototype cost does not serve as a direct indicator for setting the product's final selling price. Further development work is required to enhance the prototype's durability and reliability.

In other words, while the current prototype shows promise in terms of cost-efficiency, additional investment and refinement are necessary to ensure that the final product meets the necessary quality and performance standards, which may involve additional testing, material improvements, and refining the production process to create a market-ready product.

A proper Graphical User Interface (GUI) will be developed in the next step to allow the user to interact with the device.

Part II

GUI Development

10 Literature Review

10.1 Waterfall Model

The waterfall model is one of the oldest and most well-known software development methodologies and was initially proposed by Winston W. Royce in 1970 [Roy87]. The model is characterized by its linear and sequential approach to project management [NST23]. Each phase must be completed before moving on to the next, and it is particularly well-suited for projects with well-defined and stable requirements.

The original waterfall model, as outlined by Royce, consisted of seven phases: System Requirements, Software Requirements, Analysis, Program Design, Coding, Testing, and Operations [Roy87]. However, over time, variations of the model have emerged. Hausen [Hau] mentioned that the most commonly recognized version includes five phases: Requirements Analysis, Design, Coding, Testing, and Maintenance.

One advantage of the waterfall model is its simplicity, ease of understanding, and implementation [SYM20]. It provides a clear structure and allows for a step-by-step progression through the development process [SYM20]. Rachma and Muhlas [RM22] argue that the waterfall model is particularly well-suited for projects that involve generic software or systems that provide services to buyers.

However, the waterfall model has also been criticized for its limitations. Zahia et al. [BZJ14] argue that this model is best suited for hardware production and may overlook the unique characteristics of software development. Yahya and Maidin [YM23] stated that the waterfall model does not allow for flexibility or

adaptability during the development process, which means that once a stage is completed, it is difficult to make changes or return to a previous stage. This lack of flexibility can be problematic when uncertainties arise or when there is a need for iterative development.

Figure 10.1 shows the waterfall model that will be utilized for this project.

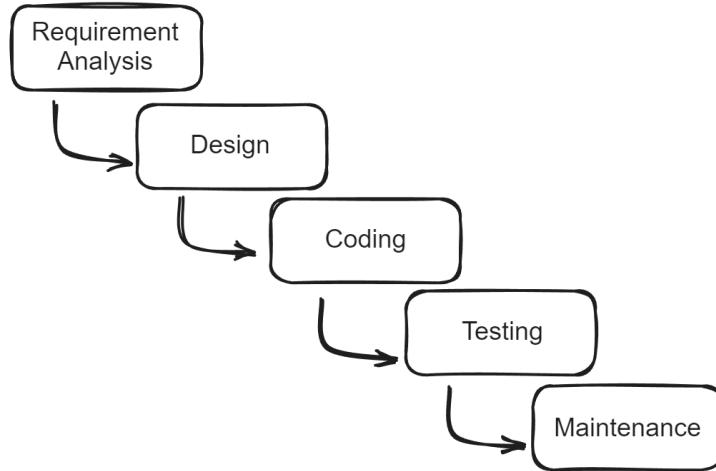


Figure 10.1: Waterfall Model

10.2 Model-View-Controller

The Model-View-Controller (MVC) architecture is a design pattern commonly used in software development to separate the concerns of an application into three distinct components: the model, the view, and the controller [Gar23, 47]. The model represents the data and business logic of the application, the view is responsible for presenting the data to the user, and the controller handles user input and updates the model and view accordingly [SZA14].

The MVC architecture offers several benefits for software development. Firstly, it promotes code reusability by separating the different aspects of the application [SZA14]. Additionally, MVC allows for multiple representations of the same information, creating different views for different user interfaces [SZA14]. This flexibility is advantageous in interactive applications where users may

have different preferences or requirements.

Another significant advantage of MVC is that it allows for the separation of concerns [Ste]. By isolating the data, business logic, and presentation layers, developers can better understand and modify each component without affecting the others. This modularity improves the maintainability and scalability of the application, as changes can be made to one component without impacting the entire system [Koz23].

The application will utilize the MVC pattern as its backbone. Further implementation details will be discussed in the following chapter. Figure 10.2 shows the MVC architecture being used for this project.

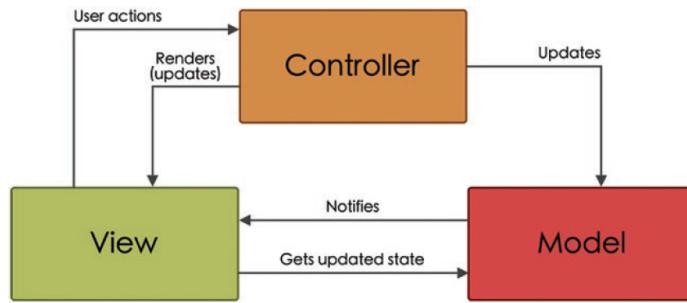


Figure 10.2: Model-View-Controller Architecture [Gar23, 46]

11 Requirement Analysis

In the waterfall model, as proposed by Royce, requirements analysis is typically the first phase of the development process [Roy87]. This phase involves gathering, documenting, and analyzing stakeholders' needs and expectations to define the software project's scope and objectives [Hau]. The objective of requirements analysis is to pinpoint the user needs and translate them into explicit, quantifiable, and attainable requirements that the software development team can employ to design and build the system [Wal23].

As per the findings of Walker [Wal23], diverse teams usually carry out this phase within a large organization. Business Analysts identify and document system needs, while Project Managers oversee the process. Developers use these requirements to design the system, and Testers validate them. Stakeholders provide essential input, and Subject Matter Experts offer specialized knowledge to help ensure the requirements are accurate and complete.

According to Geog and Dharani [GD16], one of the significant challenges in this phase is determining what needs to be constructed, as unlike other phases in development, errors in this phase can have far-reaching consequences if identified later on. They also stated that one of the primary hurdles in the requirements specification process is striking a balance between providing sufficient detail for clear understanding without overly constraining the system.

Hence, to properly define the project requirements, the following steps are taken [Sim23]:

Identify Key Stakeholders and End-Users

During the project's initial phase, it is crucial to identify the key stakeholders with significant influence and play a vital role in determining its scope. Identifying the end-users who will benefit from the product is equally important. In this project, the primary beneficiaries of the product will be the police officers and the administrators of the police department.

Capture Requirements

The subsequent steps involve obtaining input from stakeholders and end-users through various techniques. These methods may include one-on-one interviews, focus groups, use cases, and prototypes to capture their precise requirements. Each technique has its distinct purpose, from gathering detailed individual needs to envisioning the product's overall functionality.

Categories Requirements

Once requirements are gathered, they are categorized into functional and non-functional requirements. This categorization ensures clarity and avoids potential confusion.

Interpret and Record Requirements

Following categorization, a process of interpretation and recording follows. This step involves assessing the feasibility, prioritizing requirements, conducting impact analyses, resolving conflicts, and ensuring precise and detailed descriptions aligned with business objectives.

After a thorough analysis, a comprehensive document detailing the requirements is created and distributed among key stakeholders, end-users, and development teams.

Sign off

A signed agreement from key stakeholders is crucial to maintain the project's scope, prevent uncontrolled expansion, and signify the final decision on re-

quirements.

11.1 Project Requirements

Figure 11.1 and 11.2 show the result of requirement analysis process.

Project Name

Handheld Speed Measurement Device

Objective

Create a portable, lightweight handheld device that accurately measures a moving object or vehicle's speed.

Stakeholders

1. Police officers
2. Police departments
3. Government officials

Figure 11.1: Project Requirements (1/2)

Functional Requirements

1. Capture multiple images or videos.
 - a. Capture the image and the timestamp accurately.
 - b. The image quality is high.
2. Track object
 - a. The algorithm should be able to perform object tracking.
 - b. The method does not consume high computational power.
3. Image processing
 - a. The image should be aligned to remove any movement during capturing.
4. Calibration
 - a. Lane calibration should be able to be performed automatically.
 - b. Solve problems with inconsistency in road width.
 - c. When automatic calibration cannot be performed, a redundant method should come into play (manual calibration or distance measurement)
5. Data Handling
 - a. Users should be able to save and load the captured data whenever possible.

Non-Functional Requirement

1. Performance
 - a. Ensure faster processing by taking leverage of parallel processing with a thread pool.
 - b. Optimize algorithms by analyzing bottlenecks.
2. Usability
 - a. The user interface should be user-friendly and straightforward.
3. Scalability
 - a. The program should be designed with scalability in mind.
 - b. Developers should be able to add features in the future.

Figure 11.2: Project Requirements (2/2)

12 Design

The design phase focuses on developing hardware and software architecture, which addresses performance, security, and data structure [Hau]. Furthermore, alongside technical considerations, attention is given to the user interface, ensuring it is user-friendly and navigable [Hau].

This section offers an overview of the project's software architecture, highlighting the utilization of the MVC model and integrating a thread pool to enhance efficiency. Moreover, it delves into user interface design principles, emphasizing familiarity and simplicity. The wireframe, color scheme, and font selection are also discussed. The concept of user flow is explained through task flows and Wireflow.

12.1 Software Architecture

The design of the MVC architecture for this project is shown in Figure 12.1. Within the figure, the separation between each MVC component is clearly shown. The view and the model components are completely decoupled from each other, and the controller component is responsible for mediating the interaction between the view and the model.

Another feature shown in Figure 12.1 is the implementation of multiple views and controllers. This implementation ensures the system's scalability, a crucial project requirement defined previously in Section 11.1. By doing so, the functionality within the project can easily be expanded in the future.

Moreover, implementing multiple controllers is necessary to prevent the *Promis-*

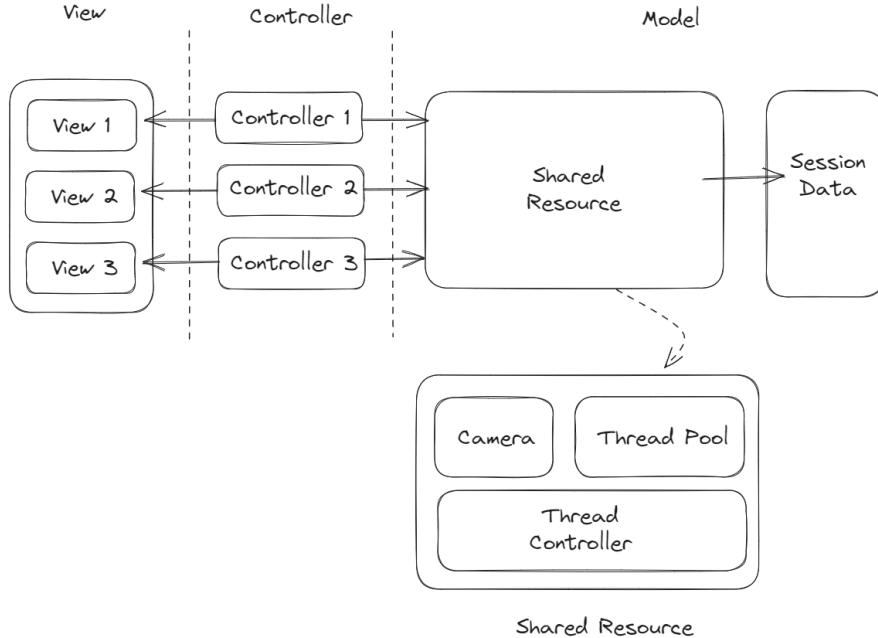


Figure 12.1: Software Architecture

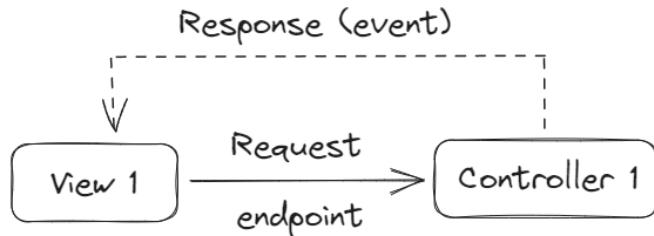


Figure 12.2: Request-Response Cycle

couious Controller problem as mentioned in [ABT⁺17, 2127]. This issue occurs when a single controller handles too many responsibilities, causing it to become bloated and difficult to maintain. By separating the controllers, each controller can efficiently handle specific responsibilities, making the system easier to maintain.

It is worth noting that the view components are carefully separated from each other. This intentional separation prevents any accidental mixing of view elements. For instance, View 1 may require specific UI components, while View 2 may need an entirely different set. This implementation simplifies the imple-

mentation process and makes it easier to understand.

The design of request and response cycles is shown in Figure 12.2. This design is crucial because it enables the View and Controller to communicate with each other. The Controller will handle any request from View via provided endpoints, process it, and then send it back to the View as a response via event.

The model components have two subcomponents: the Shared Resource and Session Data. Both subcomponents are responsible for managing the data within the application. In the later section, we will discuss the design of these subcomponents in detail.

12.1.1 Thread Pool

A thread pool is a collection of worker threads that efficiently execute asynchronous callbacks on behalf of the application [KBM]. It is primarily used to reduce the number of application threads and provide management of the worker threads [KBM].

Utilizing a thread pool involves following some basic principles. Initially, a group of threads is created, called workers or executors. The number of threads in the pool is typically determined by the CPU's number of cores [Gee20]. A more significant number of thread pools, meaning more parallel tasks, can be executed, resulting in a much better efficiency. However, a thread pool that's too large can result in resource inefficiencies, where precious time is expended in context switching between threads [Gee20].

Once the creation of workers is complete, they become available for execution. Tasks can be created and added to a queue. Any available workers will then be assigned to a task from the queue. Once a task is finished, the worker will receive a new task until the queue is empty. If all workers are currently busy, the task will wait in the queue until an available worker can be assigned to it. [Gee20]

Figure 12.3 shows the thread pool architecture overview. The thread pool object will improve the system's efficiency for this project. It is one of the required



Figure 12.3: Thread Pool Architecture [Eug23]

features for this project, as defined in Section 11.1. It solves one of the problems mentioned in [BMS23], where some computer vision algorithms are computationally expensive and can take a long time to execute.

Suppose the tasks are run in a single-threaded environment. In that case, the user has to wait for a long time until the task is completed, which is not ideal for a user interface application because it makes the application unresponsive. On the other hand, if multiple instances of the task are run in parallel without proper monitoring, the system may become overloaded, which can cause it to crash.

The thread pool implementation comes into play to solve this problem. It creates a fixed number of available threads, enabling parallelism while preventing the system from being overloaded. Additionally, the thread pool manages the threads, so the user does not have to worry about creating and managing them. The user only needs to create the task and submit it to the thread pool, which will assign the task to an available thread.

12.1.2 Thread Controller

Including a Thread Controller in the application is crucial because it effectively lightens the workload of the main controller, thus improving its efficiency and performance. By effectively managing complex threading tasks, the Thread

Controller allows the main controller to focus solely on managing communications between the model and view.

Inside the Thread Controller, there are several groups of threads. It is important to note that these threads differ from those defined in a thread pool. In this context, the threads within the Thread Controller are specialized units responsible for executing specific business logic. Each thread operates independently, enabling the system to handle multiple tasks simultaneously.

Furthermore, the Thread Controller is designed with access to the thread pool. This integration empowers the system to carry out tasks that require parallel processing in an organized way. Tasks can be placed in a queue within the Thread Pool, ensuring they are executed efficiently and without resource conflicts.

The design of this system is focused on scalability. Developers can define groups of threads to customize the system to meet specific performance needs. Additionally, the Thread Controller allows developers to control thread execution from the controller closely. This design makes achieving a high level of customization and adaptability to varying workloads possible. This combination of the Thread Controller and thread pool creates a strong foundation for building responsive, efficient, and scalable applications.

12.2 User Interface Design

User Interface (UI) design is a crucial aspect of product development that focuses on a product's look, style, and interactivity [Cou23]. It is the first thing users encounter when they use an application or visit a website [Cou23]. A user-friendly UI design is pivotal in enriching the user experience, boosting user engagement, and ultimately shaping the success of an application [Alg23].

Various design guidelines can be used to create a user-friendly UI design. Paun [Pau20] stated that familiarity with UI design benefits both users and designers. It streamlines workflows, ensures a seamless experience, and efficiently uses

established conventions. Adhering to standards and maintaining consistency is essential for a unified and positive user experience.

Fleck [Fle21] argues that simplicity in design plays a crucial role in UI design. A simple design minimizes cognitive load and allows users to accomplish tasks efficiently. The author also emphasizes that it is essential to remember that simplicity does not mean sacrificing functionality; instead, it means prioritizing essential elements and removing any extraneous details that may cause confusion or overwhelm the user.

In GUI design, feedback is as important as consistency and simplicity. It comes in forms like visual cues and error messages, helping users understand the product's state and how to interact with it. This feedback lessens confusion, builds trust, and aids in learning how to use the product, even when errors occur. [Flo22]

12.2.1 Wireframe

A wireframe is a basic, simplified layout of a digital interface that outlines content placement and page components [Whi23]. It is a quick way to present a concept or idea without extensive time or resources [Whi23].

Gupta [Gup23] stated that a good wireframe excludes elements that could divert attention from decisions about the application's structure. However, it instead prioritizes the functional purpose over visual aesthetics, presenting a straightforward, basic portrayal of the application's features in monochrome. The author also stated that designers should provide annotations to explain and elaborate on specific features, providing clarity and context.

Figure 12.4 shows the rough sketch of the user interface that will be used for this project. The user interface will be divided into four main sections: the title panel, image panel, status panel, and button panel.

The title panel plays a crucial role in the application by presenting the current panel and informing the user regarding the primary function of the panel. Additionally, this component provides various other features, such as closing the

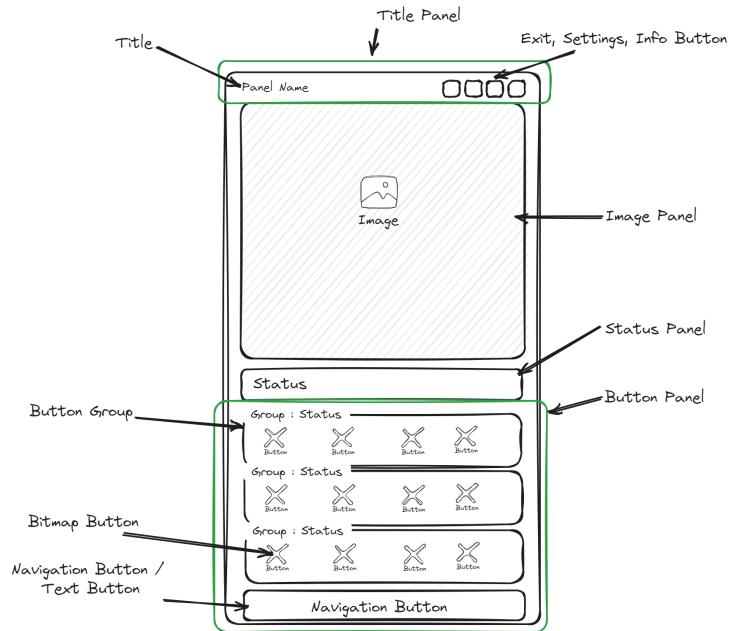


Figure 12.4: Wireframe

application, accessing the settings, and offering basic information about the application.

The image panel has the specific purpose of displaying images to the user. It is intended to display the images captured by the camera and the images being processed. The status panel, on the other hand, is intended to display the current status of the application to the user. Keeping the user informed about the application's status is crucial, allowing them to stay up-to-date with its progress.

The button panel will show the available buttons for the user to interact. The buttons will be categorized based on their functionality. There will be two kinds of buttons: bitmap buttons and text buttons.

12.2.2 Color Scheme

Color is a pivotal element in UI design, playing a vital role in evoking emotions, establishing hierarchy, and distinguishing design components [M.23]. With

correct implementation, color can significantly improve the website or application's usability and overall user experience [M.23].

Ou et al. [OLWW04] emphasized the strong correlation between color and emotion, highlighting how different colors evoke specific emotional responses. Their research underscores the significant impact that color choices can have on users' emotional experience.

Lewandowska and Olejnik-Krugly [LOK21] stated that color is a significant aspect of visual communication. It plays a crucial role in conveying information. It is integral in identifying products, indicating quality, and influencing user interfaces. They also stated that colors can evoke diverse visual sensations, and their impact is often experienced in combination rather than in isolation, underscoring the crucial role of color in human perception and communication.

In their paper, Zhang and Ferris [FZ16] discuss essential factors to consider when choosing colors for design. These include the number of colors used, color harmony, and text overlay. Regarding the number of colors, designers have various options, such as using a single color with different shades or following the 60-30-10 rule for triadic color schemes, emphasizing primary, secondary, and accent colors. Figure 12.5 provides color combinations following the 60-30-10 rule.

In choosing the primary color, it is crucial to consider the brand's message and the emotions it wants to deliver [M.23]. The secondary color should complement the primary color and provide contrast to support the dominant color, while the accent color is used in features like buttons and pop-ups [M.23].

Figure 12.6 shows the color combination used for this project. A simple white and black combination is used for the primary and secondary colors. Since the end-users will be the police officers, the blue shade is the accent color to represent the police force.

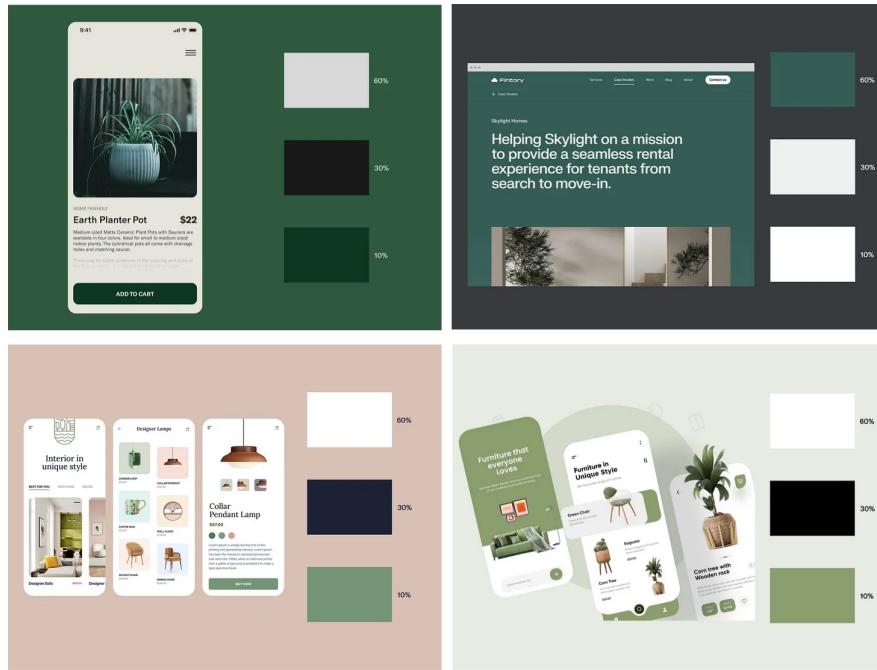


Figure 12.5: Example of Color Combination with 60-30-10 Rule [M.23]

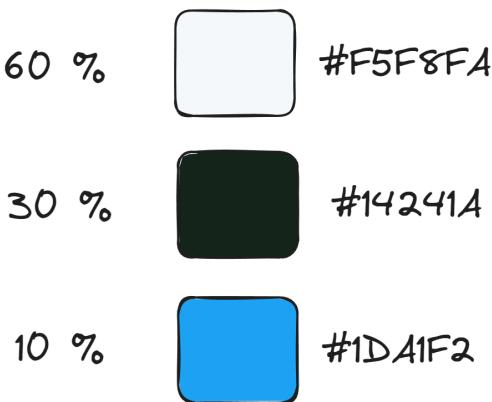


Figure 12.6: Color Combination

12.2.3 Typography

Typography is critical in UI design, as over 90% of online information is presented in text form [FP22]. It involves more than selecting attractive fonts; other factors such as typeface, font, character, baseline, and height are essential [FP22].

Sawyer et al. [SDCR20] stated that picking the right font is essential. It affects how things look and how easy they are to read. This finding is especially true for glances on screens. The authors also stated that recent studies found that different fonts can make a big difference in how easy things are to read in real-life situations.

The font used for this project is Roboto, a sans-serif typeface family created by Google as the system font for its mobile operating system Android [Mot22]. According to Google, this font is designed to provide the best reading experience across different devices [Mot22]. Figure 12.7 shows example text with the Roboto font on multiple styles.

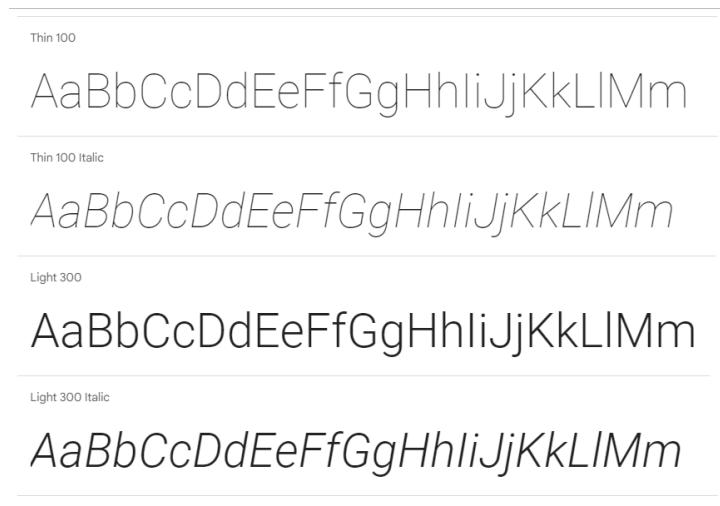


Figure 12.7: Roboto Font

12.2.4 User Flow

User flow in UI design refers to a user's path to complete a specific task within an application or website. It includes each step, from the starting point to the endpoint with diagrams that display a user's complete path when using a product [Bro23]. They help designers understand and anticipate the cognitive patterns of users in order to create products that enable a state of flow [Bro23].

Our project will involve utilizing two user flow types to illustrate user-system

interactions. The first type is the task flow, which outlines a step-by-step process users must follow to accomplish a specific task [Tri23]. Task flows do not incorporate design elements and are typically described in natural language [Tri23]. They provide an overview of the primary actions users need to take to achieve a particular objective or endpoint [Rah22].

The other type is Wireflow, a visual representation of the screens and interactions a user follows to complete specific tasks [Tri23]. It combines aspects of a basic wireframe, task flow, and flowchart to advanced screen flows that depict multiple navigation paths in one diagram [Tri23]. Arrows and annotations between wireframes are added on a single canvas to indicate user paths while using the product [Ang].

Figure 12.8 shows the task flow of the application, showing the task required, from start to finish, to successfully measure the speed of a moving object. The Wireflow of the application is shown in Figure 12.9, showing available panels that the user can navigate.

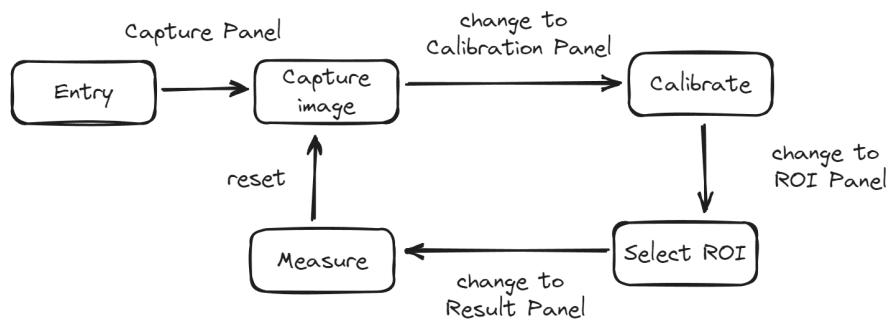


Figure 12.8: Task Flow

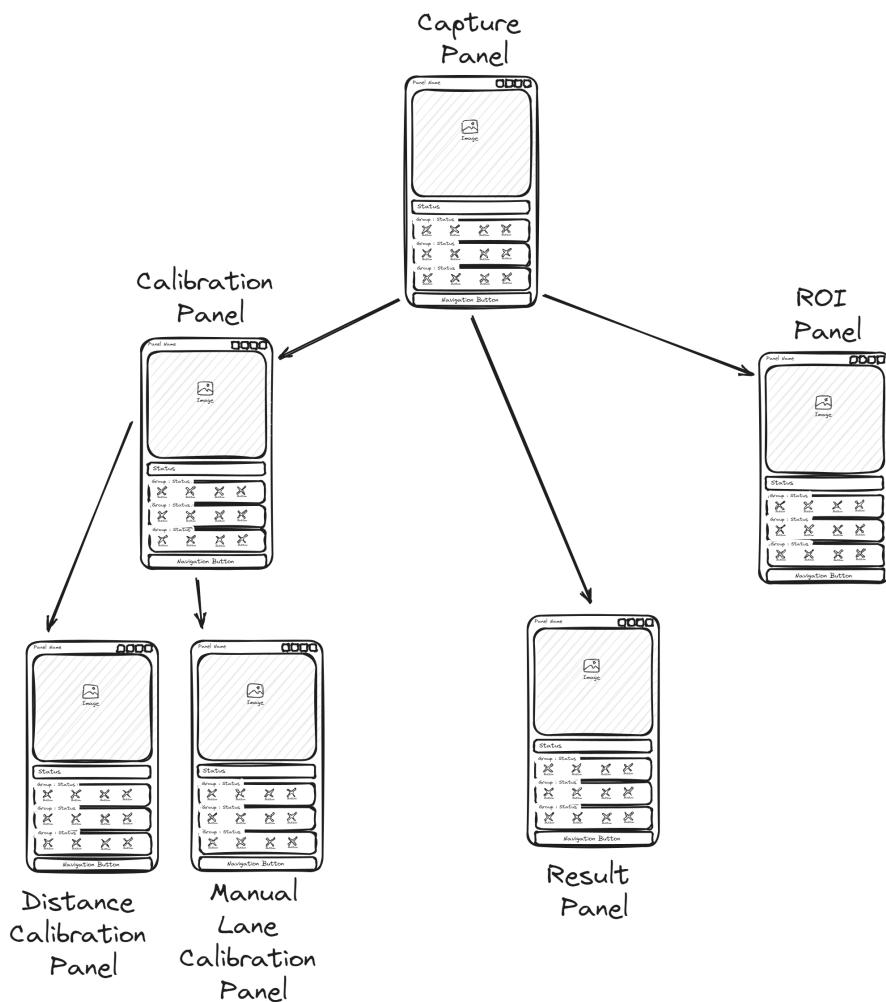


Figure 12.9: Wireflow

13 Coding

In the project's third phase, the application will be implemented as per the established specifications [Hau]. This chapter will provide a comprehensive overview of the implementation process of the application, explaining each component of the MVC architecture and providing relevant examples. Additionally, we will discuss improvements to the algorithm. The application is developed using the C++17 programming language and utilizes the following libraries throughout the development process:

- OpenCV v4.5.5
- wxWidgets v3.2.2.1
- libcamera v0.0.4

Additionally, the application is developed using Visual Studio Code and CMake. A cross-compilation setup speeds up the compilation process, which compiles the code on a separate machine and transfers the compiled code to the target machine. Appendix A.9 provides a detailed explanation of correctly setting up the development environment and the required dependencies.

13.1 View Implementation

As discussed previously in Chapter 10.2, the View is responsible for displaying the data to the user. In other words, it defines the user interface and displays the information from the model [KV20]. In this section, the implementation of the View is discussed in detail.

To further streamline the development process of View, a set of guidelines has been set up to be followed throughout the development process. These guidelines are as follows:

Handling User Input

This guideline emphasizes that the View should be adept at managing and processing user interactions. It is responsible for capturing user inputs, such as mouse clicks, keyboard entries, or touch gestures, and conveying them to the Controller for further processing. This can involve events like form submissions, button clicks, or interactions with interactive elements on the user interface.

Avoid any business logic

This requirement underscores the importance of maintaining a clear separation of concerns within the MVC architecture. The View should refrain from incorporating any form of business logic that involves tasks like data validation, computation, or decision-making processes. Instead, these responsibilities are delegated to the Model and Controller components. By adhering to this guideline, the View remains focused on its core function of presenting data and interacting with the user interface.

Avoiding Direct Model Manipulation

This guideline reinforces the principle of ensuring that the View does not directly modify the state or data of the Model. Instead, any alterations to the underlying data should be orchestrated through the Controller. This guideline establishes a controlled data flow within the application, maintaining data integrity and adherence to the MVC architectural pattern.

Utilizing Templates or Layouts

In the development of the View component, it is recommended to use templating engines or layout systems. These tools enable the creation of modular and reusable components that can be combined dynamically to build the user inter-

face. By utilizing templates or layouts, developers can simplify the design and rendering of the View, leading to a more scalable and maintainable codebase.

Error Handling and Feedback Mechanisms

This guideline emphasizes implementing robust error-handling mechanisms within the View. It is important to provide clear and informative feedback to the user in case of errors or invalid inputs. This guideline guides users through the application's workflow, even when unexpected events occur. Effective error handling contributes to a more user-friendly and reliable user experience.

13.1.1 Main Layout

The implementation of the main layout is the first step in the development of the View. The main layout is the universal layout that is used throughout the application. The implementation is done based on the result of the wireframing process, which is done in Section 12.2.1.

The main layout is shown in Figure 13.1. The main layout consists of the title, image, status, and button panels. All components' functions are explained previously in Section 12.2.1.

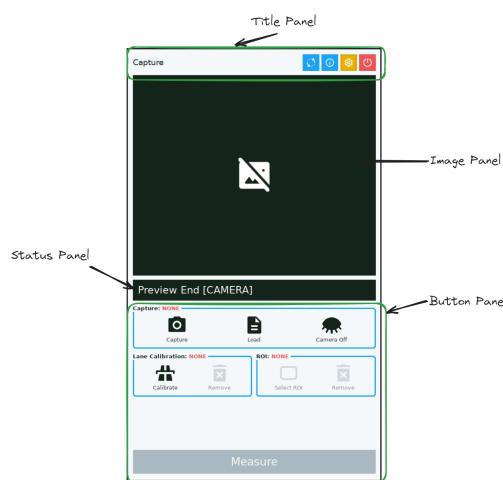


Figure 13.1: Main Layout

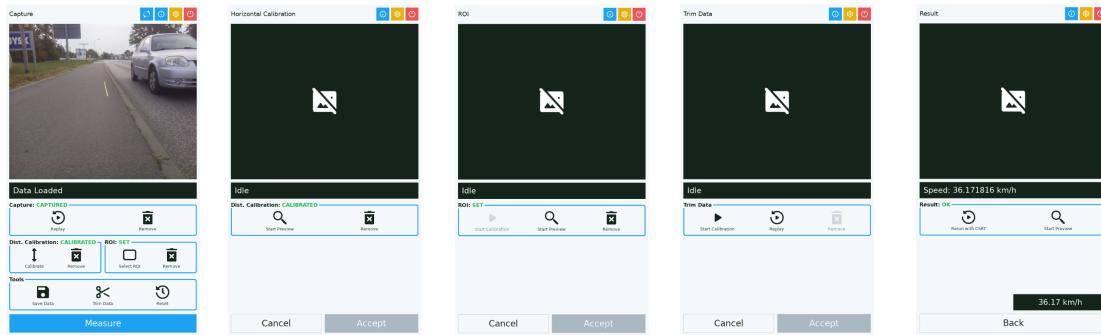


Figure 13.2: Example of other implemented Panels

Figure 13.2 shows multiple other panels used within the application. As observed, these panels all have similar layouts, with the only difference being the components within the button panels. This similarity is intentional, as it helps to maintain consistency within the application. This layout is implemented by the base class `BasePanel` and `BasePanelWithTouch`. For more details, please refer to the project documentation in Appendix A.6.

13.1.2 Button Panel

This panel is responsible for displaying bitmap buttons that users can interact with. Bitmap buttons are buttons that have an image as their icon. They initiate specific processes within the application, such as capturing an image or calibrating the camera. During the implementation phase, multiple types of bitmap buttons are used. Within the panel also contains group boxes, which is used to group buttons with similar functionalities. The implementation of these components is discussed in detail below.

Bitmap Button

Bitmap Button is a type of button that are used widely within the application. It contains a bitmap image and a text label positioned below the image. Figure 13.3 shows an example of the bitmap buttons used within the application.

Throughout the implementation process, three different types of bitmap but-

tons are implemented. They are Type 1, Type 2, and Type 3. Two types, Type 1 and Type 2, share a similar design concept but differ in their state handling.



Figure 13.3: Bitmap Button

Figure 13.4 shows the button appearance in different states for the Type 1 bitmap button. This type of button is designed to be utilized for process that requires a long time to complete. For example, when capturing an image, the button assumes a *NORMAL* state when no image has been captured. When pressed, the process started, and the button transitions to an *ACTIVE* state, signifying that the process is underway. During this active state, the button is also disabled to prevent any potential interruptions or accidental re-triggering of the process. Once the process is completed, the button shifts to a *DISABLE* state, preventing the reinitiation of the process.

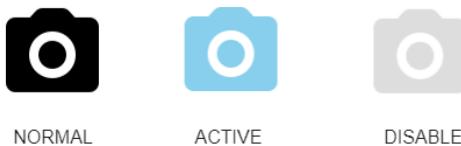


Figure 13.4: Type 1 Button State



Figure 13.5: Type 2 Button State

The states for the Type 2 button are shown in Figure 13.5. This type of button is used when the button is used to trigger a process that requires toggling. For instance, when turning on a camera, the button assumes an *OFF* state when the process has not been initiated and is available for activation. Upon pressing the

button, the process initiates, and the button transitions to an *ON* state, indicating that the process is actively running. If necessary, pressing the button again will revert the state to *OFF*, effectively terminating the ongoing process.



Figure 13.6: Example of Type 3 Button

The last type of bitmap button is Type 3. In terms of appearance, it differs from the previous two types. This button contains only a bitmap image without the text label. It does not have states and is used to handle a straightforward process, such as exiting the application or opening the settings panel. An example of a Type 3 bitmap button can be seen in the title panel (see Figure 13.6).

13.1.3 Group Box

Organizing buttons with similar functions using a Group Box is a beneficial practice that can elevate the user experience. This component can streamline navigation and improve overall functionality by grouping buttons that perform similar tasks. As an illustration, the Capture Panel (as exemplified in Figure 13.7) displays three distinct groups: Capture, Calibration, and ROI. Within the Capture Group, three buttons relate to the capturing process: the Capture Button, which captures images; the Load Button, which loads images; and the Toggle Camera Button, which activates the camera.

Furthermore, the Group Box can also display the state of the application. For example, the Capture Group Box shows the current state of the captured data, whether it is empty or already captured (refer to Figure 13.8). When the data is empty, the Group Box will display *NONE* text in red. On the other hand, if the data is captured, the Group Box will display the text *CAPTURED* in green color.



Figure 13.7: Groups within Capture Panel

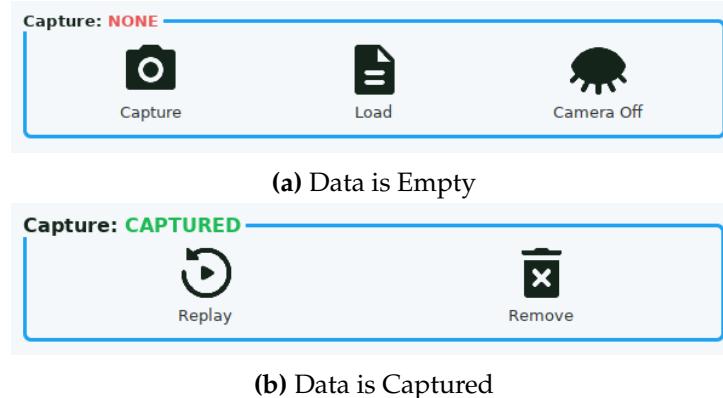


Figure 13.8: Example of different states of Group Box

Handling Button Input

The button input is handled using wxWidgets' provided event mechanism. For more details, refer to wxWidgets Documentation [[wxW](#)].

Handling button input involves binding the button to a specific function. When the button is pressed, an event is captured, and the function is executed. Depending on the implementation, different buttons can trigger different functions. For instance, pressing the button to turn on the camera will trigger a function to activate the camera, while pressing the button to capture an image will trigger a function to take a picture.

To assign a specific function to a button, giving each button a unique ID is necessary. This unique identifier makes identifying and linking the button to a particular function easier. Detailed instructions on binding specific buttons to a function can be found in the developer manual in Appendix [A.9](#). Alternatively, the project documentation in Appendix [A.6](#) also provides a detailed explanation of the process.

Updating Button State

A proper feedback mechanism is essential to ensure that the user is aware of the current state of the application. In the application context, the button state is used to provide feedback to the user on the current state of the application. For example, when the user interacts with a button to turn on the camera, the feedback to the user is that the button state changes from *OFF* to *ON*.

Updating the button state is done using event handling, similar to the process of handling button input. An event is created and captured when an update is required, for example, after a button is pressed or a process is completed, and the button state is updated accordingly. The controller component usually does this process, as it handles the application logic.

However, it is also important to note that while updating the component state is crucial, updating it too frequently can harm the application's performance. Therefore, it is crucial to find a balance between the two.

13.1.4 Image Panel

The Image Panel component is a crucial element of the user interface. It is a dynamic component that provides visual feedback and interactive input to the user. Its value lies in its ability to convey important information in real time to the user, providing updates on the application's state. For example, during the image-capturing process, the Image Panel displays the captured image instantly, giving users immediate visual feedback.

Moreover, the Image Panel also functions as a responsive input mechanism. Through proper event handling, users can interact with the Image Panel using various input methods, such as mouse interactions or touch-based inputs. This section focuses on the processes involved in presenting images to the user and enabling user interactions.

Displaying Image to the User

The `UpdatePreviewEvent` is a specific type of *Data Event* responsible for updating images on the image panel. In general, a *Data Event* is an event that contains data, which is discussed in greater detail in Section 13.3.1. This event is usually created by either the Controller or threads, including the image that needs to be updated.

For instance, in displaying images from the camera within a thread's execution, every thread loop will create the `UpdatePreviewEvent` and the current camera image will also be included. The View will catch the submitted event, unpack the data, and display the information on the image panel.

Handling Touch Input

Input from users can be achieved through the touch input, which shares similarities with the button input. It is possible to attach touch events to a function, which will be executed whenever a touch event occurs.

The `wxMouseEvent` from wxWidgets can be utilized to enable this functionality. Alternatively, the `BasePanelWithTouch` class can also be inherited during View implementation. This class already has implemented touch event handling, including methods to handle these events. For further details, kindly refer to the project documentation in A.6.

13.1.5 Message Dialog

A dialog is a user interface component commonly used in various libraries and frameworks. It is a modal window that appears on top of the application content to provide important information or ask for a decision [Bay22]. When a dialog is open, all app functionality is disabled, and it persists on the screen until the user takes the required action, confirms it, or dismisses it [Bay22].

This UI component is crucial, as it can provide users when immediate attention is necessary [Mat]. A simple use case for Dialog is with error handling. When an error occurs, a Dialog can inform the user about the error and provide options

for the user to take action and resolve the problem. Another use case is when the user needs to make a decision. For example, when the user wants to exit the application, a Dialog can be used to confirm the user's decision.

According to Android Developers [Dev], A dialog box typically includes a title, some text, and buttons for users to take action. The title briefly describes the dialog's purpose, while the text provides more detailed information. Users can take action by clicking on the buttons. Figure 13.9 shows an example of the dialog component.

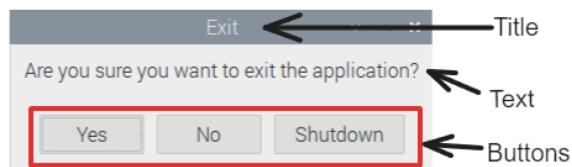


Figure 13.9: Example of Message Dialog

13.1.6 Navigating between different panel

This application has multiple panels or views implemented based on its different functionalities. For instance, the capture panel helps capture an image, while the calibration panel helps perform camera calibration. Since there are multiple views, it is essential to have a proper control method to display them to the user.

To simplify the process, we have implemented a rule that only one View can be displayed within the frame while the others remain hidden. The implementation is done by using the `Show()` and `Hide()` methods provided by `wxWidgets`. The current active View will be hidden with the `Hide()` method during the navigation process, while the new View will be shown.

An example of navigating between different panels by using `Show()` and `Hide()` methods is shown in Figure 13.10.

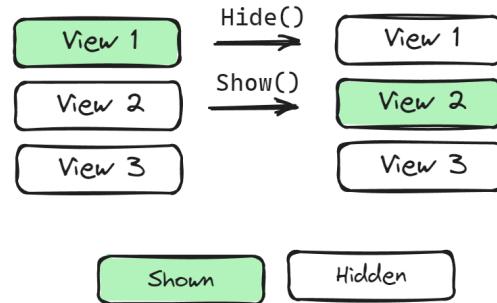


Figure 13.10: Example of navigating between different panel

13.2 Model Implementation

13.2.1 Session Data

In this section, an overview of the Session Data is provided. This component is a part of the Model, which is responsible for storing the data and the application logic. Figure 13.11 shows the overview of the Session Data.

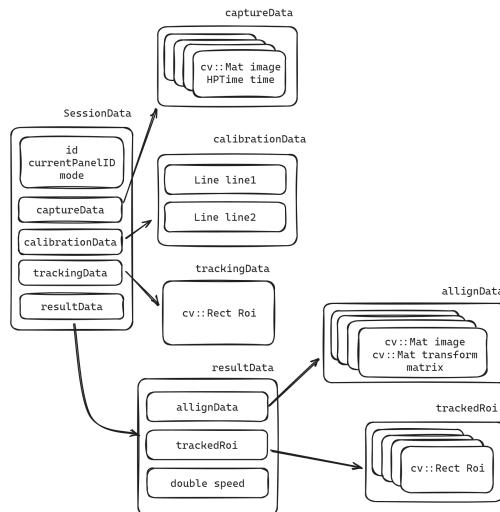


Figure 13.11: Overview of Session Data

The Session Data consists of four subcomponents, which are CaptureData, CalibrationData, TrackingData, and ResultData. Each of these components is explained in detail below.

CaptureData: This data stores a vector of original images captured by the camera. Alongside the image, this data type is also responsible for storing the time the image is captured to be used for measurement purposes.

CalibrationData: Contains the result of the calibration process, which are the variables *line1* and *line2*. Both of these `Line` objects represents the calibrated lines, which are required for the measurement process as mentioned in previous work [BMS23].

TrackingData: This data stores the initial region of interest (ROI) on which the tracked object is located. This implementation helps to filter out unwanted objects during tracking, which will help the optical flow algorithm to focus on the tracked object.

ResultData: The data contains the output of a calculation process, consisting of three variables: `alignData`, `trackedRoi`, and `speed`. The `alignData` variable stores the output of the alignment process, while the `trackedRoi` variable stores the ROIs on which the tracked object is located. The `speed` variable represents the speed of the tracked object, which is calculated using the outputs of both the alignment and tracking processes.

13.3 Controller Implementation

As mentioned previously in Section 12.1, each View component will have its Controller associated with it. By doing so, each Controller will have specific responsibility for handling the application logic for the View. Furthermore, this design also helps in terms of application scaling. Having multiple controllers allows the application logic to be divided into smaller parts, making it easier to maintain and scale.

However, proper controller design guidelines are required to implement these controllers properly. Therefore, we have set up a set of guidelines that we have followed throughout the development process. These guidelines are as follows:

Only handle request and response

This guideline emphasizes that the Controller should only focus on handling requests and responses between View and Model. It should not contain any business logic. Instead, the business logic should be implemented in the threads, which will be discussed in Section [13.3.2](#).

Provide similar endpoints

To ensure proper communication between the Controller and the View, providing similar endpoints for each Controller is important. By doing so, developers can easily identify the communication methods between the View and the Controller. To properly implement this guideline, all methods responsible for communication between the View and the Controller should be named with the prefix `e_`, which stands for an endpoint. For example `e_startCamera()`, `e_captureImage()` and `e_calibrate()`.

Only handle one view

This guideline emphasizes that each Controller should only handle one View. This guideline ensures that the Controller is manageable with a few responsibilities, which enables the Controller to be easily maintained and scaled.

Singular responsibility

Each endpoint should only handle one responsibility. For example, `e_startCamera()` should only handle the process of starting the camera and not other processes, such as capturing an image or calibrating the camera.

Handle error

All endpoints should be wrapped with a try-catch block to handle any error during the process. If an error occurs, an `ErrorEvent` will be created and passed to the view component. The View component will then display the error message to the user using a message dialog, as discussed previously in Section [13.1.5](#).

Prevent handling outside active panel

This guideline emphasizes that the Controller should only handle requests and responses when the View is active. This strict rule is implemented to prevent any unwanted behavior that might occur when the View is not active. The `checkPrecondition()` provides a method to check whether the View is active. If the View is inactive, an error message will be displayed to the user.

13.3.1 Event as a Feedback Mechanism

As mentioned previously in Section 12.1, the Controller is responsible for handling the application logic. This process involves handling requests, which is done via the endpoints, while the responses are handled via the events.

Within this project, events play a huge role in the communication process between the View and the Controller. Events do not simply inform the View regarding the request status; they can also pass data between the View and the Controller. Hence, two types of Events are defined, which are *Empty Event* and *Data Event*.

Empty Event is an event that does not contain any data. It is used to inform the View regarding the status of the request. This type of event is usually utilized to inform the user of the current status of the requested process, such as when the process is started, completed, or failed.

Data Event, on the other hand, acts as a container to pass data between the View and the Controller. This type of event is used when the View requires data from the Model, such as the captured image or the result of the calculation process. This type of event is also used to update the View with the latest data, such as the current camera image or application state.

Creating and handling events is done using wxWidgets' provided event mechanism. For more details, refer to wxWidgets Documentation [wxW]. Alternatively, the Project Documentation in Appendix A.6 also explains how to create custom events.

13.3.2 Threads and Thread Controller

To reduce the Controller's responsibility, handling the business logic is tasked to the *threads*. Separating the execution of business logic on different threads prevents the main thread or the GUI thread from being blocked. This design decision is essential to ensure the application's responsiveness.

When a process needs to be executed, the Controller handles the request and signals the Thread Controller to initiate the process. The process runs independently on different threads, and once the task is finished, the Thread Controller is informed to close the threads. The entire process is handled through the Controller endpoint and Event.

Figure 13.12 provides a sequence diagram illustrating the connections of the MVC components with the Thread Controller.

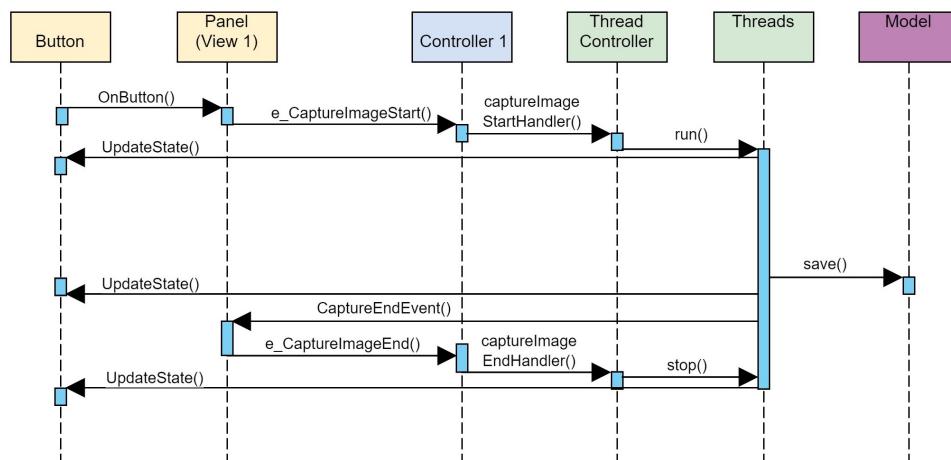


Figure 13.12: Example of Request Handling

With this example, the request starts by pressing a button. This action will trigger an event, which will be handled by the *View1*. The View will then request an initialization of process or thread via the endpoint `e_CaptureImageStart()`. The request will be forwarded to the *ThreadController*, and a thread to capture the image is initiated and run.

The button state is updated to signal the succession of the request. The thread

will run on its own, and when it is finished capturing the image, the data is saved to the Model via the `save()` method. Once again, The state is updated, and an event signaling the completion of the process is sent to the View.

With this event, the View signals the Controller to stop the thread via the endpoint `e_CaptureImageStop()`. The thread is then stopped and joined. The state is updated to signal the completion of the process.

13.4 Algorithm Improvement

In this section, the improvement of the algorithm is discussed. Based on the previous work [BMS23], the algorithm is divided into three main processes: image alignment, object tracking, and distance measurement. Several improvements are made to the algorithm during the implementation process, which are discussed in detail below.

13.4.1 Image Alignment

As per the previous research [BMS23], aligning images is carried out by combining feature detection and feature matching algorithms. The feature detection algorithm identifies key points or features between two images, and the matching results are used to calculate the transformation matrix, which aligns the images. The figure below (Figure 13.13) illustrates the overall image alignment process.

The author has mentioned that the entire process can be pretty time-consuming, especially when dealing with large image sizes. Each step involved in aligning a vector of images of size 1280 x 960 pixels was analyzed to identify the factors contributing to this. Table 13.1 shows the average time taken for each process (t_{proc}) and the percentage of total time taken ($\%_{total}$). The results indicate that the feature detection process on both reference and target image is the most time-consuming, accounting for up to 95% of the overall process.

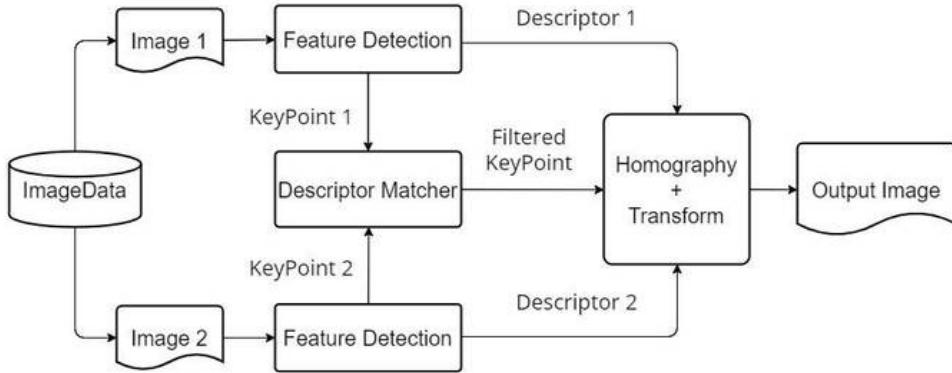


Figure 13.13: Image Alignment Process

Process	t_{proc} [ms]	$\%_{total}$ [%]
Feature detection on reference image	1579.92	47.827
Feature detection on target image	1582.64	47.909
Feature Matching	101.27	3.066
Filter Keypoints	0.30	0.009
Homography	2.27	0.069
Transform	37.01	1.120

Table 13.1: Time taken for each process

After further analysis of the process, it has been determined that performing feature detection on the reference image is unnecessary for every iteration. In this context, the reference image refers to the first image to which we want to align the other images. Therefore, feature detection on the reference image only needs to be done once, and the resulting data can be cached and used for subsequent iterations. Implementing this approach will significantly reduce the time taken for the feature detection process and the overall processing time.

Figure 13.14 illustrates the image alignment process with caching. In this process, the output of feature detection on the reference image is stored in a cache and used in subsequent iterations. Table 13.2 presents the average time taken for aligning the image (\bar{t}_{align}) using different implementations. It also shows the percentage of improvement relative to the original implementation ($\%_{improve}$). As per the results, the improved implementation is approximately 93% faster

than the original implementation.

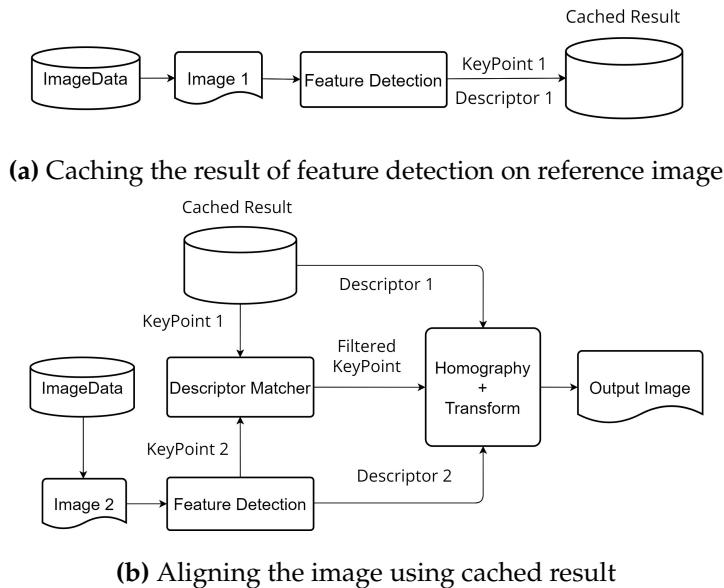


Figure 13.14: Image Alignment Process with Caching

Implementation Type	t_{align} [ms]	$\%_{improve}$ [%]
Original	3160	0
Cached	1631	93.7
Cached + 2 Workers	1085	191.2
Cached + 3 Workers	941	235.8

Table 13.2: Comparison of time taken for image alignment

In addition, by using a thread pool as described in section 12.1.1, the process can be enhanced even more by utilizing parallel execution. Table 13.2 illustrates that with 2 and 3 workers, the process can be improved by 191.2 % and 235.8 %, respectively.

13.4.2 Custom Lane

According to a previous study [BMS23], measuring the distance of an object from the camera requires knowledge of the distance between the two lanes on the road. Typically, in Germany, this distance is around 3.5 meters. However, in

real-world situations, this distance may vary, leading to measurement inaccuracies. The study also highlights the importance of accurately selecting the lane representing the distance between the two lanes. Even a slight error in lane selection can result in a significant measurement error.

A simplified method has been introduced to streamline the selection of the appropriate line. This method introduces a custom object with predetermined dimensions into the camera's field of view, serving as a reference point (as depicted in Figure 13.15). This object includes two lines, one in blue and the other in yellow. Both lines run parallel and are precisely 420 mm apart and will serve as references for the measurement procedure. An illustration of the line selection process can be seen in Figure 13.16.

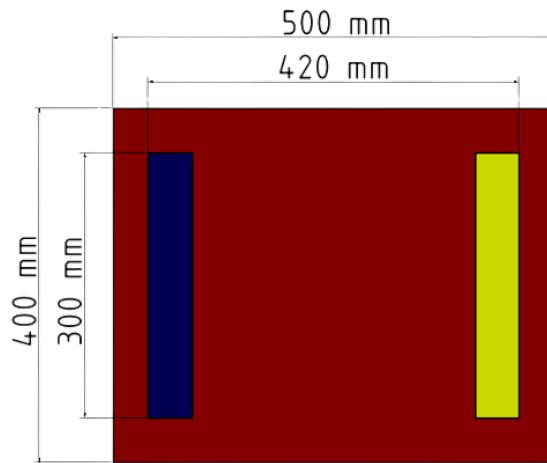


Figure 13.15: Custom Object

13.4.3 Distance Measurement

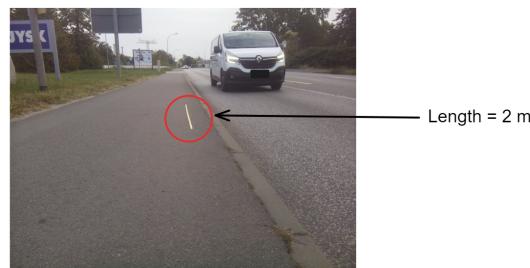
Distance measurement is based on Javadi et al.'s work [JDP19]. This type of speed measurement will serve as an alternative speed calculation method to lane measurement for this research.

To accurately determine the speed of a tracked object, a series of steps must be taken. First, a known object of a specific length should be positioned within

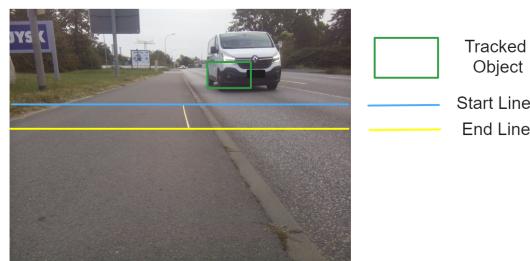


Figure 13.16: Example of selecting lines for measurement

the camera's view to serve as a reference point. This reference object will aid in calculating the speed of the tracked object. Next, two lines should be drawn to mark the starting and ending points of the placed object. An illustration of this process can be observed in Figure 13.17.



(a) Placement of reference object



(b) Calibration of Start and End Line

Figure 13.17: Distance Measurement Process

By tracking an object's movement, we can determine the frame at which it crosses both the start and end lines. This allows us to calculate the time it crossed the start line (t_{start}) and the time it crossed the end line (t_{end}). With

the length of the object (l_{object}) already known, we can estimate its speed using the following formula:

$$Speed = \frac{l_{object}}{t_{end} - t_{start}} \quad (13.1)$$

Compared to the lane measurement method, this method is more straightforward. However, as mentioned by Javadi et al. [JDP19], it may lead to inaccurate calculations as the exact time of the object crossing the start and end lines cannot be precisely determined.

This measurement method is implemented in `DistSpeedMeasurement` class. Please refer to the project documentation on [A.6](#) for more information.

14 Testing and Maintenance

Software testing and maintenance are essential activities in the software development cycle. Software testing is the process of verifying and validating that a software is functioning correctly and meeting its requirements and specifications [ibm] [Ham23].

This phase enables the early detection and resolution of bugs and errors, ensuring a smoother and more efficient delivery of the final product [Ham23]. Additionally, thorough testing guarantees the reliability and security of the software, particularly in applications handling sensitive data or with critical functionalities [Ham23].

In software testing, there are numerous available methods and techniques to ensure the quality of the software. Some of the most common ones are unit testing, integration testing, system testing, and acceptance testing [ibm]. For this project, we will utilize unit testing, which will be discussed in detail in Section 14.1.

14.1 Unit Testing

Unit testing is a type of software testing that focuses on individual units or components of a software system and the purpose of performing this test is to validate that each unit of the software works as intended and meets the requirements [gee23b].

They are designed to validate the smallest possible unit of code, such as a function or a method, and test it in isolation from the rest of the system, which will

allows developers to quickly identify and fix any issues early in the development process, improving the overall quality of the software and reducing the time required for later testing [gee23b].

In writing unit test, it is advised to include multiple number of scenarios to ensure that the code is working as expected. This includes testing the code with valid and invalid inputs, as well as edge cases [Var22]. Additionally, to further simplify the process of testing, it is recommended to use a unit testing framework, which provides a set of tools and utilities to automate the process of writing and executing unit tests [Var22]. For this project, we will be using the GTest framework, which is a C++ unit testing framework developed by Google.

14.1.1 Example of Good Unit Test

An example of a good unit test can be found in the Appendix A.7. This unit test is written in C++ using the GTest framework. The code demonstrates a simple bank account management system, encapsulated within the `BankAccount` class. This class offers functionalities like depositing (`deposit()`), withdrawing (`withdraw()`), and checking the account balance (`getBalance()`).

To ensure robustness, exception handling mechanisms are added to the code. Specifically, when an attempt is made to deposit a negative amount or withdraw more than the available balance, the methods will throw appropriate exceptions, either `std::invalid_argument` or `std::runtime_error`.

The unit tests will thoroughly examine the behavior of these functionalities. The `BankAccountTest.Deposit` case validates that funds are successfully deposited, asserting that the balance matches the expected value. Similarly, `BankAccountTest.Withdraw` verifies correct withdrawal behavior by comparing the resultant balance after a withdrawal operation.

In `BankAccountTest.WithdrawTooMuch`, an attempt to withdraw an excessive amount triggers an exception. The test checks whether this exception is properly thrown, ensuring the balance remains unchanged. Lastly, `BankAccountTest.DepositNegative` evaluates the system's re-

sponse when attempting to deposit a negative amount. This test expects a `std::invalid_argument` exception to be raised. The subsequent assertion checks that the account balance remains unaffected.

Within this test cases, all of the possible scenarios are covered, ensuring that the code is working as expected, which will help to improve the overall quality of the software.

14.2 Maintenance

Software maintenance can be described as the modification of a software product after it has been delivered, which includes correcting faults, enhancing performance, and adapting the product to a modified environment [BSM04]. This phase includes tasks like bug fixing, adding new features, improving performance, and ensuring compatibility with new hardware or software [gee23a].

Software maintenance can be categorized into four types [BSM04][gee23a]:

1. **Corrective Maintenance:** This addresses errors or bugs in the software, ensuring it functions properly. Swift resolution of issues enhances the software's reliability and user satisfaction.
2. **Preventive Maintenance:** This involves making changes, upgrades, or adaptations to prevent potential problems in the future. It helps identify and correct latent errors before they lead to disruptions.
3. **Perfective Maintenance:** After a software is introduced, user needs evolve, prompting the addition of new features or the improvement of existing ones. This ensures the software remains relevant and efficient.
4. **Adaptive Maintenance:** This focuses on adapting the software to changes in technology, policies, and regulations. It includes adjustments for new hardware, operating systems, and compliance requirements.

For our project, following maintenance activities will be performed:

1. **Bug Fixing:** This involves correcting any errors or bugs in the software, ensuring it functions properly. Swift resolution of issues enhances the software's reliability and user satisfaction.
2. **Adding New Features:** This involves adding new features to the software to improve its functionality and user experience. We are interested to explore other algorithms which can improve either the accuracy of the calculation, or the ease of use of the software.
3. **User Interface Improvements:** This involves improving the user interface of the software to improve its usability and user experience. To develop a more user-friendly interface, numerous iterations based on user feedback will be performed.

15 Conclusions and Future Work

15.1 Conclusions

This thesis aimed to design software that utilizes computer vision to measure a moving object's velocity accurately. The development approach adopted for this project was the Waterfall model - a straightforward, sequential method of software development that is particularly suitable for projects with clear and well-defined requirements. The Waterfall model consists of five phases: requirements analysis, design, coding, testing, and maintenance. This model was selected for its simplicity and ease of comprehension.

During the requirements analysis phase, the software requirements are collected and examined. The stakeholders, including the supervisor and the end user, provide these requirements. The collected requirements are then analyzed to ensure they are precise, comprehensive, and coherent. The outcome of this phase is a document containing the complete set of software requirements.

Within the design phase of software development, two crucial components are designed - the architecture and the user interface. To ensure that the software is easy to maintain, the architecture is designed with the Model-View-Controller (MVC) pattern as the backbone of the application. This pattern is selected because it is well-suited for developing user interfaces and separates the business logic from the user interface. A thread pool is also implemented to enhance the software's performance. In this phase, wireframing is performed to design the software's user interface, along with other critical aspects such as color and typography. The user flow is also created to visualize the flow of the software, making it easier to understand and navigate.

During the coding phase, the software is created based on the design from the previous phase. We use the C++ programming language, along with essential libraries like OpenCV, wxWidgets, and libcamera, for developing the software. In this section, we have explained the implementation of the MVC architecture, outlining the role of each component.

Improvements to the algorithms from previous works are also made during the coding phase. We focused on enhancing critical processes such as image alignment and speed measurement. To achieve this, we strategically implemented caching mechanisms and parallel execution through a thread pool. We also integrated a custom reference object to simplify lane selection. Additionally, we implemented an alternative method to measure speed that gives users another option instead of lane measurement.

The software undergoes strict examination throughout the testing phase to guarantee its proper functionality. The primary approach employed for this purpose is unit testing, which concentrates on the individual units or components of the software system. This testing aims to confirm that each unit of the software executes its intended function and complies with the requirements. The unit tests are drafted utilizing the GTest framework, a C++ unit testing framework created by Google.

After delivering the software to the end user, it will be maintained through bug fixing, adding new features, and improving the user interface. The maintenance activities aim to ensure that the software remains efficient and up-to-date.

15.2 Future Work

Due to a lack of time, many adaptations, tests, and experiments have been left for the future. Future work concerns deeper analysis of the algorithms and the implementation of new features and improvements to the user interface.

15.2.1 Software Design

During the design process in Chapter 12, we focused heavily on the software architecture, aiming to realize the MVC concept. While the results are satisfactory, there are still areas for improvement. The following ideas can be explored in the future:

1. **Asynchronous Programming:** Asynchronous programming is a technique that allows a program to start a potentially long-running task without having to wait for it to finish before responding to other events [Moz]. Instead, the program can remain responsive while the task runs in the background. Once the task has been completed, the program is presented with the result. Request and response handling is currently implemented via endpoints and events in the Controller class. While this approach is functional, it can be difficult to understand and appears complicated. Asynchronous programming can simplify the request and response handling process and should be explored as an alternative approach.
2. **Thread pool implementation:** The current thread pool implementation requires a custom task class. Exploring an implementation that accepts functions directly could streamline the process. An example can be seen here [Mtr].

15.2.2 User Interface

The current user interface is simple and user-friendly, yet there is room for enhancement. The following ideas can be explored in the future:

1. **Iterations Based on User Feedback:** Due to time constraints, only a few iterations were made to improve the user interface. Future works could focus on gathering more user information and iterating the UI based on feedback.
2. **Complete Overhaul:** In user interface design, it is important to follow the familiarity concept. This means that the user interface should be designed

to be familiar to the user by taking inspiration from popular applications such as Instagram, Facebook, and Twitter. Following the design concept of these popular mobile applications, the user interface can be redesigned to enhance user experience.

15.2.3 Algorithm Improvement

Some improvements to the algorithms can also be explored to either improve the accuracy of the calculation or the ease of use of the software. The following ideas are proposed for future works:

1. **Process smaller image size:** The current software implementation processes the image in its original size. While this is not an issue for powerful computers, it can cause problems for low-end computers. To address this, the image can be resized to a smaller size before processing. However, this may impact the accuracy of the calculation. In such cases, experiments can be conducted to determine the cost tradeoff between accuracy and processing time.
2. **Increase Capture Rate:** To improve the accuracy of speed calculation, particularly in distance measurement, it is possible to increase the capture rate. However, the ability to do so is restricted by hardware limitations, which can result in the system shutting down due to low voltage. To address this issue, it is necessary to either analyze the power consumption of the hardware and find ways to reduce it or upgrade to more powerful hardware.
3. **Accuracy of Lane Selection:** In Section 13.4.2, it was noted that the accuracy of lane selection is crucial for accurate speed calculation. However, the current lane selection implementation is imperfect, and its accuracy can sometimes be off, even with user input. Two potential solutions are suggested to improve the accuracy. One solution is to implement the ability to zoom in and out of the image within the `ImagePanel` class. This idea would allow users to select the intended lane better and improve accuracy. Another solution is to explore alternative line detection algo-

rithms. The Hough Transform, a popular method for line detection, can be explored, but it relies heavily on parameter tuning, which can be problematic.

4. **Machine Learning Approach:** With the advancement of technology, neural networks can be explored to improve accuracy in computer vision.
5. **Automation of Process:** The ultimate goal of this project is to enable the software to process videos automatically without requiring user input. However, more research and understanding of the algorithm are needed to achieve this goal, particularly in the calibration phase.

15.2.4 Hardware Consideration

The Raspberry Pi Foundation has recently launched the Raspberry Pi 5, an improved iteration of the earlier Raspberry Pi 4 [Pi], notable for its more powerful processor than its predecessor. Additionally, the camera module 3 has been released, an upgrade over the camera module 2 currently in use [Ber23]. These new products can potentially enhance the software's performance and should be considered.

Part III

Indexes and Appendix

List of Figures

3.1	Pahl and Beitz's Design Process [Pah07, 130]	10
3.2	Original Prusa i3 MK3S+	11
3.3	Example View of PrusaSlicer	12
4.1	Planning and Task Clarification [Pah07, 146]	15
4.2	Checklist for Establishing the Prototype's Requirements [Pah07, 149]	17
5.1	Steps in Conceptual Design [Pah07, 160]	25
5.2	Result of Abstraction Process	26
5.3	Breaking down the overall function into sub-functions [Pah07, 32]	27
5.4	Overall Function of the System	27
5.5	Sub-Functions of the System	28
5.6	Sub-Functions of the System (Final)	28
5.7	Morphological Chart with Solution Variants	32
5.8	Sketch of Solution Variant 1	34
5.9	Sketch of Solution Variant 2	35
5.10	Sketch of Solution Variant 3	36
5.11	Sketch of Solution Variant 4	37
5.12	Sketch of Solution Variant 5	38
5.13	Sketch of Solution Variant 6	39
5.14	Sketch of Solution Variant 7	40
5.15	Sketch of Solution Variant 8	41
5.16	Selection Chart for Solution Variants	42
6.1	Steps in Embodiment Design [Pah07, 229]	45
6.2	Design guidelines for 3D printing [And22]	49

LIST OF FIGURES

6.3	Preliminary Design Variant 2	50
6.4	Views of Preliminary Design Variant 2	50
6.5	Body Components of Preliminary Design Variant 2	50
6.6	Placement of inner components for Variant 2	51
6.7	Methods to secure 3D-printed components [Her20]	52
6.8	Quick release plate	53
6.9	Preliminary Design Variant 3	54
6.10	Placement of inner components for Variant 3	55
6.11	Bumps on the back cover	55
6.12	Battery Placement	56
6.13	Preliminary Design Variant 6	57
6.14	Placement of inner components for Variant 3	57
6.15	Handle Grip	58
6.16	Placement of handle grip and quick release plate	58
6.17	Preliminary Design Variant 7	59
6.18	Placement of inner components for Variant 7	60
6.19	Placement of quick release plate	60
7.1	Steps in the detail design process	70
7.2	Power Switch	71
7.3	Position of the camera component	71
7.4	Protective bump for camera	72
7.5	Protective bump for screen	72
7.6	The LAN port	73
7.7	Position of the LAN port	74
7.8	Germany Police Logo [bun]	74
7.9	Result of recolor	75
8.1	Printed Parts	77
8.2	The installed threaded insert	78
8.3	The Switch Holder	79
8.4	The installed switch	79
8.5	The LAN Port Slot	79
8.6	The installed LAN port	80
8.7	The Camera Module Slot	80

LIST OF FIGURES

8.8 The installed camera module	81
8.9 The Battery Holder	81
8.10 The installed battery	82
8.11 The Raspberry Pi Slot	82
8.12 The Screen Slot	83
8.13 The installed screen and top cover	83
8.14 The Final Product	84
10.1 Waterfall Model	90
10.2 Model-View-Controller Architecture [Gar23, 46]	91
11.1 Project Requirements (1/2)	94
11.2 Project Requirements (2/2)	95
12.1 Software Architecture	97
12.2 Request-Response Cycle	97
12.3 Thread Pool Architecture [Eug23]	99
12.4 Wireframe	102
12.5 Example of Color Combination with 60-30-10 Rule [M.23]	104
12.6 Color Combination	104
12.7 Roboto Font	105
12.8 Task Flow	106
12.9 Wireflow	107
13.1 Main Layout	110
13.2 Example of other implemented Panels	111
13.3 Bitmap Button	112
13.4 Type 1 Button State	112
13.5 Type 2 Button State	112
13.6 Example of Type 3 Button	113
13.7 Groups within Capture Panel	114
13.8 Example of different states of Group Box	114
13.9 Example of Message Dialog	117
13.10 Example of navigating between different panel	118
13.11 Overview of Session Data	118
13.12 Example of Request Handling	122

LIST OF FIGURES

13.13Image Alignment Process	124
13.14Image Alignment Process with Caching	125
13.15Custom Object	126
13.16Example of selecting lines for measurement	127
13.17Distance Measurement Process	127

List of Tables

4.1	Requirement List (1/2)	22
4.2	Requirement List (2/2)	23
5.1	Classification Scheme for Working Principles	31
6.1	Printing cost for Variant 2	53
6.2	Manufacturing cost for Variant 2	54
6.3	Printing cost for Variant 3	56
6.4	Manufacturing cost for Variant 3	56
6.5	Printing cost for Variant 6	59
6.6	Manufacturing cost for Variant 6	59
6.7	Printing cost for Variant 7	60
6.8	Manufacturing cost for Variant 7	61
6.9	Weighting Factors for Evaluation Criteria	63
6.10	Value Scale for Evaluation [Pah07, 115]	64
6.11	Value Scale for Weight Distribution	64
6.12	Value Scale for Device Weight	64
6.13	Value Scale for Device Size	64
6.14	Value Scale for Ease of Assembly	65
6.15	Value Scale for Swappable Parts	65
6.16	Technical Evaluation of Preliminary Design Variants (1/2)	67
6.17	Technical Evaluation of Preliminary Design Variants (2/2)	67
6.18	Economic Evaluation of Preliminary Design Variants	67
6.19	Total Rating of Preliminary Design Variants	68
7.1	Sizing guide for threaded inserts [ruta][rutb]	73
8.1	Printing Time and Filament Used	76

LIST OF TABLES

8.2 Total Printing Cost	84
8.3 Total Material Cost	85
13.1 Time taken for each process	124
13.2 Comparison of time taken for image alignment	125

Bibliography

- [ABT⁺17] ANICHE, MAURÍCIO, GABRIELE BAVOTA, CHRISTOPH TREUDE, MARCO AURÉLIO GEROSA and ARIE VAN DEURSEN: *Code smells for Model-View-Controller architectures*. Empirical Software Engineering, 23(4):2121–2157, September 2017.
- [Alg23] ALGORPUBLIC: *The importance of User Interface Design in app development*. <https://www.linkedin.com/pulse/importance-user-interface-design-app-development-algorepublic/>, May 2023.
- [And22] ANDERSTHOESTESEN, DDDIMENSION: *Complete Design Guide for 3D printing*. <https://www.ddd-dimension.com/en/post/complete-design-guide-for-3d-printing>, Mar 2022.
- [Ang] ANGELES, MIKE: *Wireframing user flow with wireflows: Wireframing Academy: Balsamiq*. <https://balsamiq.com/learn/articles/wireflows/>.
- [Bat] BATES, HOWIE: *Materials for 3D printing by fused deposition*. https://www.materialseducation.org/educators/matedu-modules/docs/Materials_in_FDM.pdf.
- [Bay22] BAY, UNCLE BIG: *Create an interactive material UI dialog in react*. <https://www.copycat.dev/blog/material-ui-dialog/>, Dec 2022.
- [Ber23] BERRYBASE: *Raspberry pi camera module 3 wide, 12MP*. <https://www.berrybase.de/raspberry-pi-camera-module-3-wide-12mp>, Jan 2023.

BIBLIOGRAPHY

- [BMS23] BIN MOHD SABTU, MUHAMMAD HAZIQ: *Developing Speed Measurement Algorithm with OpenCV and Raspberry Pi*. 2023.
- [Bro23] BROWNE, CAMREN: *What are user flows in UX design? [full beginner's guide]*. <https://careerfoundry.com/en/blog/ux-design/what-are-user-flows/>, Apr 2023.
- [BSM04] BHATT, PANKAJ, GAUTAM SHROFF and ARUN K. MISRA: *Dynamics of Software Maintenance*. SIGSOFT Softw. Eng. Notes, 29(5):1–5, sep 2004.
- [bun] BUNDESPOLIZEI, DIE: *Die bundespolizei*. https://www.bundespolizei.de/Web/DE/05Die-Bundespolizei/03Organisation/Organisation_node.html.
- [BZJ14] BEN-ZAHIA, MOHAMED A. and IBRAHIM JALUTA: *Criteria for selecting software development models*. In *2014 Global Summit on Computer Information Technology (GSCIT)*, pages 1–6, 2014.
- [Cou23] COURSERA: *What is UI Design? definition, tips, best practices*. <https://www.coursera.org/articles/ui-design>, Jun 2023.
- [Dev] DEVELOPERS, ANDROID: *Dialog - android developers*. <https://developer.android.com/jetpack/compose/components/dialog>.
- [DSPB23] DREISSIG, MARIELLA, DOMINIK SCHEUBLE, FLORIAN PIEWAK and JOSCHKA BOEDECKER: *Survey on LiDAR Perception in Adverse Weather Conditions*, 2023.
- [Eug23] EUGEN, PARASCHIV: *Introduction to thread pools in Java*. <https://www.baeldung.com/thread-pool-java-and-guava>, Aug 2023.
- [Fle21] FLECK, RENEE: *10 fundamental UI design principles you need to know*. <https://dribbble.com/resources/ui-design-principles>, Dec 2021.

BIBLIOGRAPHY

- [Flo22] FLORIDO, DANIEL: *Key characteristics of good UI design – according to 8 experts.* <https://www.uxpin.com/studio/blog/good-ui-design-characteristics/>, Jun 2022.
- [Fly23] FLYGUYS: *Lidar vs radar.* <https://flyguys.com/lidar-vs-radar/>, Jul 2023.
- [FP22] FITZ-PATRICK, MOLLY: *The UX designer's guide to typography.* <https://www.interaction-design.org/literature/article/the-ux-designer-s-guide-to-typography>, Oct 2022.
- [FZ16] FERRIS, KEVIN and SONYA ZHANG: *A Framework for Selecting and Optimizing Color Scheme in Web Design.* In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 532–541, 2016.
- [Gar23] GARCÍA, RAÚL FERRER: *iOS Architecture Patterns.* Apress, 2023.
- [GD16] GEOGY, MANJU and ANDHE DHARANI: *A Scrutiny of the Software Requirement Engineering Process.* Procedia Technology, 25:405–410, 2016.
- [Gee20] GEEKFORGEEK: *Thread pools in Java,* Jul 2020.
- [gee23a] GEEKSFORGEEKS: *Software engineering: Software maintenance.* <https://www.geeksforgeeks.org/software-engineering-software-maintenance/>, Jul 2023.
- [gee23b] GEEKSFORGEEKS: *Unit testing: Software testing.* <https://www.geeksforgeeks.org/unit-testing-software-testing/>, Feb 2023.
- [GGA18] GORDEEV, EVGENIY G., ALEXEY S. GALUSHKO and VALENTINE P. ANANIKOV: *Improvement of quality of 3D printed objects by elimination of microscopic structural defects in fused deposition modeling.* PLOS ONE, 13(6):e0198370, June 2018.
- [Gup23] GUPTA, SAKSHI: *How to create a wireframe: Step-by-step guide,* Jul 2023.

- [Ham23] HAMILTON, THOMAS: *What is software testing? definition.* <https://www.guru99.com/software-testing-introduction-importance.html>, Sep 2023.
- [Hau] HAUSEN, DORIS: *Classic Waterfall Model in Software Engineering.* <https://www.medien.ifi.lmu.de/lehre/ws0607/mmi1/essays/Doris-Hausen.xhtml>.
- [Her20] HERMANN, STEFAN: *Helicoils, threaded insets and embedded nuts in 3D prints - Strength.* <https://www.cnckitchen.com/blog/helicoils-threaded-insets-and-embedded-nuts-in-3d-prints-strength-amp-strength-assessment>, May 2020.
- [Her23] HERMANN, STEFAN: *Tips Tricks for heat-set inserts used in 3D printing.* <https://www.cnckitchen.com/blog/tipps-amp-tricks-fr-gewindestecksel-im-3d-druck-3away>, May 2023.
- [Hos] HOSSAIN, AL-EMRAN: *Advantages and disadvantages of radar systems.* <https://www.linkedin.com/pulse/advantages-disadvantages-radar-systems-al-emran-hossain/>.
- [Hul20] HULL, ROB: *Police forces begin using “Next generation” speed guns on UK roads.* <https://www.thisismoney.co.uk/money/cars/article-9013677/Police-forces-begin-using-generation-speed-guns-UK-roads.html>, Dec 2020.
- [ibm] IBM: *What is software testing and how does it work?* <https://www.ibm.com/topics/software-testing>.
- [jdb] JDB, JDB: *Board index.* <https://forums.raspberrypi.com/viewtopic.php?t=243421>.
- [JDP19] JAVADI, SALEH, MATTIAS DAHL and MATS I. PETTERSSON: *Vehicle speed measurement model for video-based systems.* Computers Electrical Engineering, 76:238–248, June 2019.

BIBLIOGRAPHY

- [KBM] KARL-BRIDGE-MICROSOFT: *Thread pools - win32 apps.* <https://learn.microsoft.com/en-us/windows/win32/procthread/thread-pools>.
- [Koz23] KOZON, TOMASZ: *Understanding the concept of clean architecture.* <https://boringowl.io/en/blog/unveiling-the-core-principles-of-clean-architecture>, Jun 2023.
- [Kus22] KUSHABHRIN, HAIKAL: *The zwicky box: A powerful method for problem solving and creativity.* <https://nesslabs.com/zwicky-box>, Dec 2022.
- [KV20] KRASTEV, GEORGI and VALENTINA VOINOHOVSKA: *Smart Mobile Application for Public Transport Schedules - Logical Model.* TEM Journal, 9:541–545, 05 2020.
- [Lab21] LABIDI, YOUNES: *Raspberry pi shutdown reboot button.* <https://blog.berrybase.de/blog/2021/05/10/raspberry-pi-shutdown-reboot-button/>, Oct 2021.
- [LOK21] LEWANDOWSKA, ANNA and AGNIESZKA OLEJNIK-KRUGLY: *Do Background Colors Have an Impact on Preferences and Catch the Attention of Users?* Applied Sciences, 12(1):225, December 2021.
- [M.23] M., BRYSON: *Principles of color in Ui Design.* <https://uxplanet.org/principles-of-color-in-ui-design-43708d8512d8>, Aug 2023.
- [Mat] MATERIALUI: *React dialog component - material UI.* <https://mui.com/material-ui/react-dialog/>.
- [Mot22] MOTT, NATHANIEL: *Google introduces reading-optimized Roboto Serif typeface.* <https://www.pcmag.com/news/google-introduces-reading-optimized-robotoserif-typeface>, Feb 2022.
- [Moz] MOZDEVNET: *Introducing asynchronous JavaScript - Learn Web Development: MDN.* <https://developer.mozilla.org/en-US/>

- docs/Learn/JavaScript/Asynchronous/Introducing.
- [Mtr] MTREBI: *MTREBI/thread-pool: Thread pool implementation using C++11 Threads.* <https://github.com/mtrebi/thread-pool>.
- [NST23] NOVIANTI, IDHA, JOKO SOEBAGYO and WILDAN TOYIB: *Diagnosis of Maths Teaching Efficacy Beliefs Using Expert System*, 2023.
- [OLWW04] OU, LI-CHEN, MING LUO, ANDREE WOODCOCK and ANGELA WRIGHT: *A study of colour emotion and colour preference. Part I Colour emotions for single colours.* Color Research Application, 29:232 – 240, 06 2004.
- [Pah07] PAHL, GERHARD: *Engineering design*. Springer-Verlag London Limited, 2007.
- [Pau20] PAUN, GORAN: *Designing with efficiency: How familiarity can enhance experiences.* <https://www.forbes.com/sites/forbesagencycouncil/2020/10/02/designing-with-efficiency-how-familiarity-can-enhance-experiences/?sh=355c1d873428>, Oct 2020.
- [Pi] PI, RASPBERRY: *Raspberry pi 5.* <https://www.raspberrypi.com/products/raspberry-pi-5/>.
- [Poo] POOL, NORD: *See yearly day-ahead prices.* <https://www.nordpoolgroup.com/en/Market-data1/Dayahead/Area-Prices/ALL1/Yearly/?view=table>.
- [Pro] PROLASER. <https://www.prolaser4.com/en/12-prolaser-4>.
- [Prua] PRUSA, JOSEF: *Original prusa i3 mk3s+.*
- [Prub] PRUSA, JOSEF: *Prusament Pla pristine white 1kg: Original Prusa 3D-Drucker Direkt von Josef Prusa.* <https://www.prusa3d.com/de/produkt/prusament-pla-pristine-white-1kg/>.

BIBLIOGRAPHY

- [Pruc] PRUSASLICER: *Prusaslicer: Original Prusa 3D printers directly from Josef Prusa.* https://www.prusa3d.com/page/prusaslicer_424/.
- [rad] RADAR, POLICE: *Police RADAR.* <http://hyperphysics.phy-astr.gsu.edu/hbase/Sound/radar.html>.
- [Rad23] RADAR, STALKER: *Stalker II.* <https://www.stalkerradar.com/police-radar/police-radar-gun/>, Jun 2023.
- [Rah22] RAHUL: *User flow and task flow explained.* <https://bootcamp.uxdesign.cc/user-flow-and-task-flow-explained-bf229332a16d>, Nov 2022.
- [RM22] RACHMA, NUR and ITAWIRANDA MUHLAS: *Comparison Of Waterfall And Prototyping Models In Research And Development (RnD) Methods For Android-Based Learning Application Design.* Jurnal Inovatif : Inovasi Teknologi Informasi dan Informatika, 5(1):36, August 2022.
- [Roy87] ROYCE, W. W.: *Managing the Development of Large Software Systems: Concepts and Techniques.* In *Proceedings of the 9th International Conference on Software Engineering, ICSE '87*, page 328–338, Washington, DC, USA, 1987. IEEE Computer Society Press.
- [ruta] RUTHEX, RUTHEX: *Ruthex 1/4" Gewindegewindeinsatz.* <https://www.ruthex.de/collections/gewindegewindeinsatze/products/ruthex-gewindegewindeinsatz-1-4-kameragewinde-20-stuck-rx-1-4-20x12-7-messing-gewindegewindeinsatz-fur-3d-druck-1>.
- [rutb] RUTHEX, RUTHEX: *Ruthex m2,5 Gewindegewindeinsatz.* <https://www.ruthex.de/collections/gewindegewindeinsatze/products/ruthex-gewindegewindeinsatz-m2-5-70-stuck-rx-m2-5x5-7-messing-gewindegewindeinsatz-fur-3d-druck-1>.

BIBLIOGRAPHY

- [SDCR20] SAWYER, BEN D., JONATHAN DOBRES, NADINE CHAHINE and BRYAN REIMER: *The great typography bake-off: comparing legibility at-a-glance*. Ergonomics, 63(4):391–398, March 2020.
- [Sig23] SIGNALS, KUSTOM: *Directional Talon*. <https://kustomsignals.com/handheld-radar/directional-talon>, May 2023.
- [Sim23] SIMPLILEARN: *What is requirement analysis [with requirement analysis example/sample inside]*, Sep 2023.
- [Sta23] STATIS, DE: *Driver-related causes of accidents involving personal injury*. <https://www.destatis.de/EN/Themes/Society-Environment/Traffic-Accidents/Tables/driver-mistakes.html>, Jul 2023.
- [Ste] STEPIEN, MARTY: *How does MVC follow separation of concerns?* <https://goformarty.github.io/MVC-SoC/>.
- [Sup] SUPPLY, DANA SAFETY: *Kustom Signals Talon*. <https://danasafetysupply.com/kustom-signals-talon-ii-law-enforcement-radar-gun-directional-motorcycle-mount-option-hand-held-or-dash-mount-corded/>.
- [SYM20] SUNARDI, SUNARDI, ANTON YUDHANA and GHUFRON ZAIDA MUFLIH: *Sistem Prediksi Curah Hujan Bulanan Menggunakan Jaringan Saraf Tiruan Backpropagation*. JURNAL SISTEM INFORMASI BISNIS, 10(2):155–162, November 2020.
- [SZA14] SARKER, IQBAL and KHALID ZINNAH APU: *MVC Architecture Driven Design and Implementation of Java Framework for Developing Desktop Application*. International Journal of Information Technology, 7:317–322, 09 2014.
- [Tec23] TECH, LASER: *LTI 20/20 TruVISION*. <https://lasertech.com/product/truvision-photo-video-lidar-speed-measurement-device/>, Jul 2023.

BIBLIOGRAPHY

- [Tri23] TRIGO, ALVARO: *What is a Wireflow in UX design.* <https://alvarotrigo.com/blog/wireflows/>, Oct 2023.
- [Var22] VARTANIAN, ERICA: *All about unit testing: 11 best practices and overview.* <https://www.educative.io/blog/unit-testing-best-practices-overview#unit-testing-benefits>, Mar 2022.
- [Wal23] WALKER, RICKIE: *Software requirements analysis.* <https://appmaster.io/blog/software-requirements-analysis>, Jan 2023.
- [Whi23] WHITE, LUCAS: *UI and UX Design: Wireframe.* <https://www.codecademy.com/resources/docs/uiux/wireframe>, Sep 2023.
- [wxW] WXWIDGETS: *Event handling.* https://docs.wxwidgets.org/3.0/overview_events.html.
- [YM23] YAHYA, NORZARIYAH and SITI SARAH MAIDIN: *Hybrid agile development phases: the practice in software projects as performed by software engineering team.* Indonesian Journal of Electrical Engineering and Computer Science, 29(3):1738, March 2023.

A Appendix

A.1 Sketches of Working Principles

A.1.1 Screen Orientation

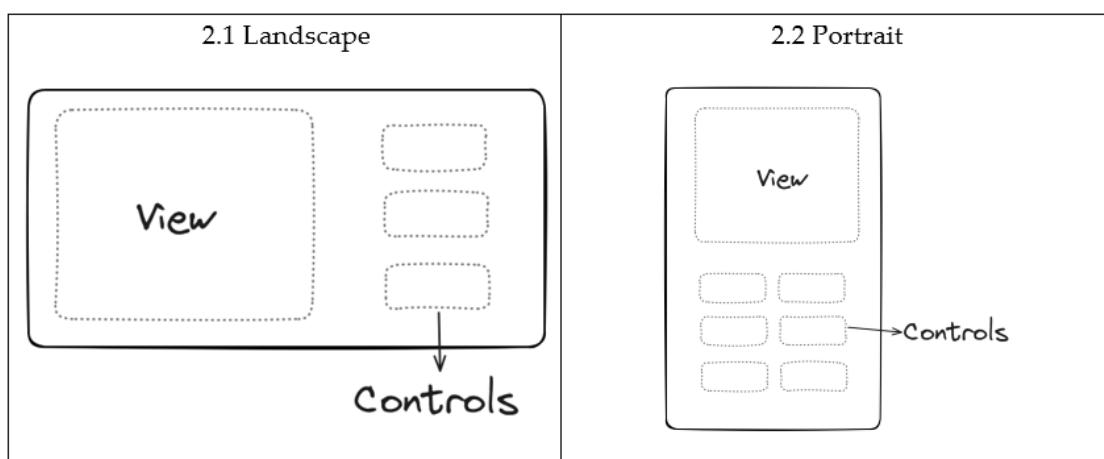


Table A.1: Screen Orientation

A.1.2 Battery Type

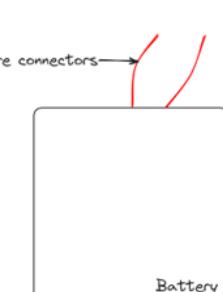
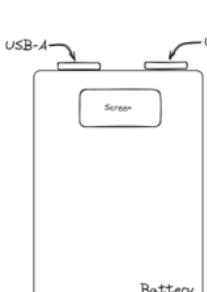
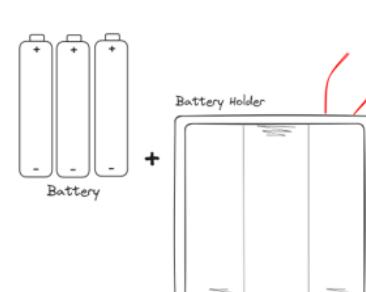
3.1 Battery Pack	3.2 Power Bank	3.3 AAA Batteries with Battery Holder
 <p>Wire connectors</p> <p>Battery</p>	 <p>USB-A</p> <p>Screen</p> <p>USB-C</p> <p>Battery</p>	 <p>Battery</p> <p>+ + + +</p> <p>Battery Holder</p> <p>Wire connectors</p>

Table A.2: Battery Type

A.1.3 Components Placement

<p>1.1 Tablet-like</p>	<p>1.2 Point-of-Service-like</p>
<p>1.3 Handheld-PC-like</p>	<p>1.4 Camcorder-like</p>

Table A.3: Components Placement

A.1.4 Body Type

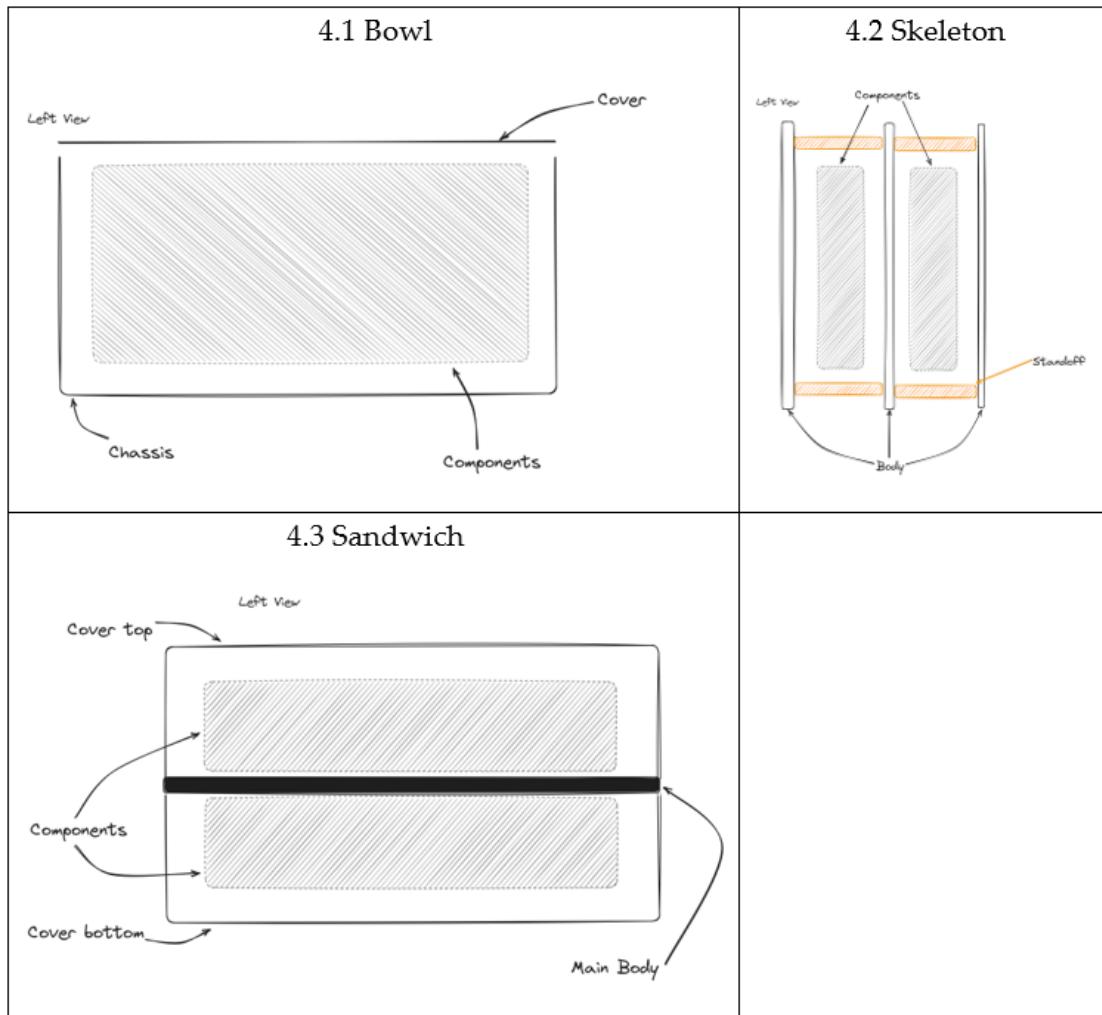


Table A.4: Body Type

A.1.5 Handling

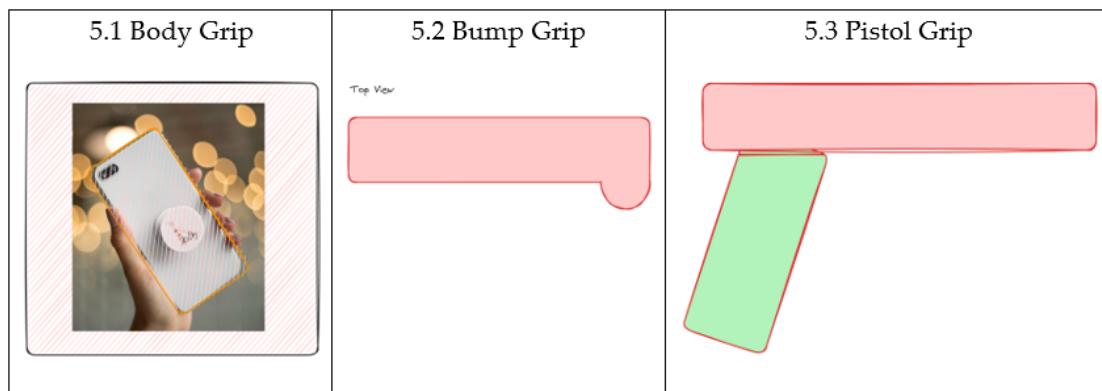


Table A.5: Handling

A.1.6 External Mounting

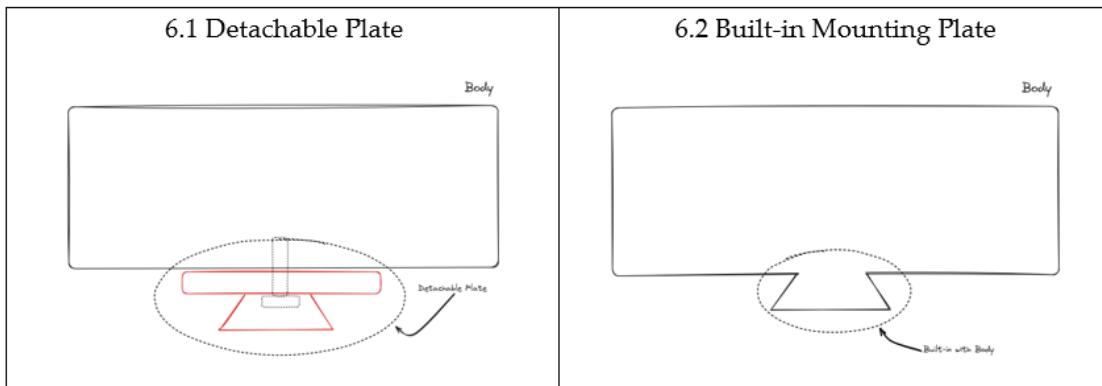


Table A.6: External Mounting

A.1.7 Control Mechanism

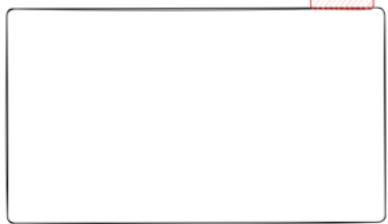
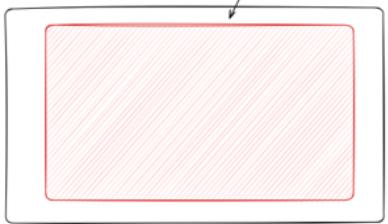
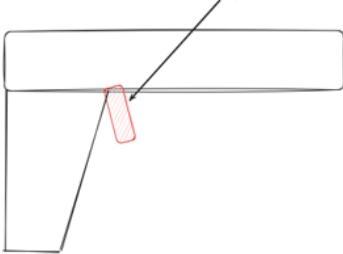
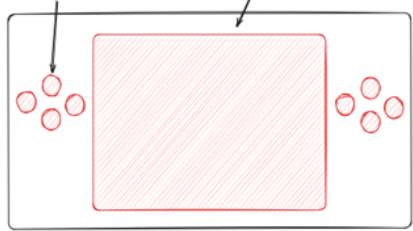
7.1 Button 	7.2 Touch Screen 
7.3 Trigger 	7.4 Touch and Button 

Table A.7: Control Mechanism

A.2 CAD Drawings

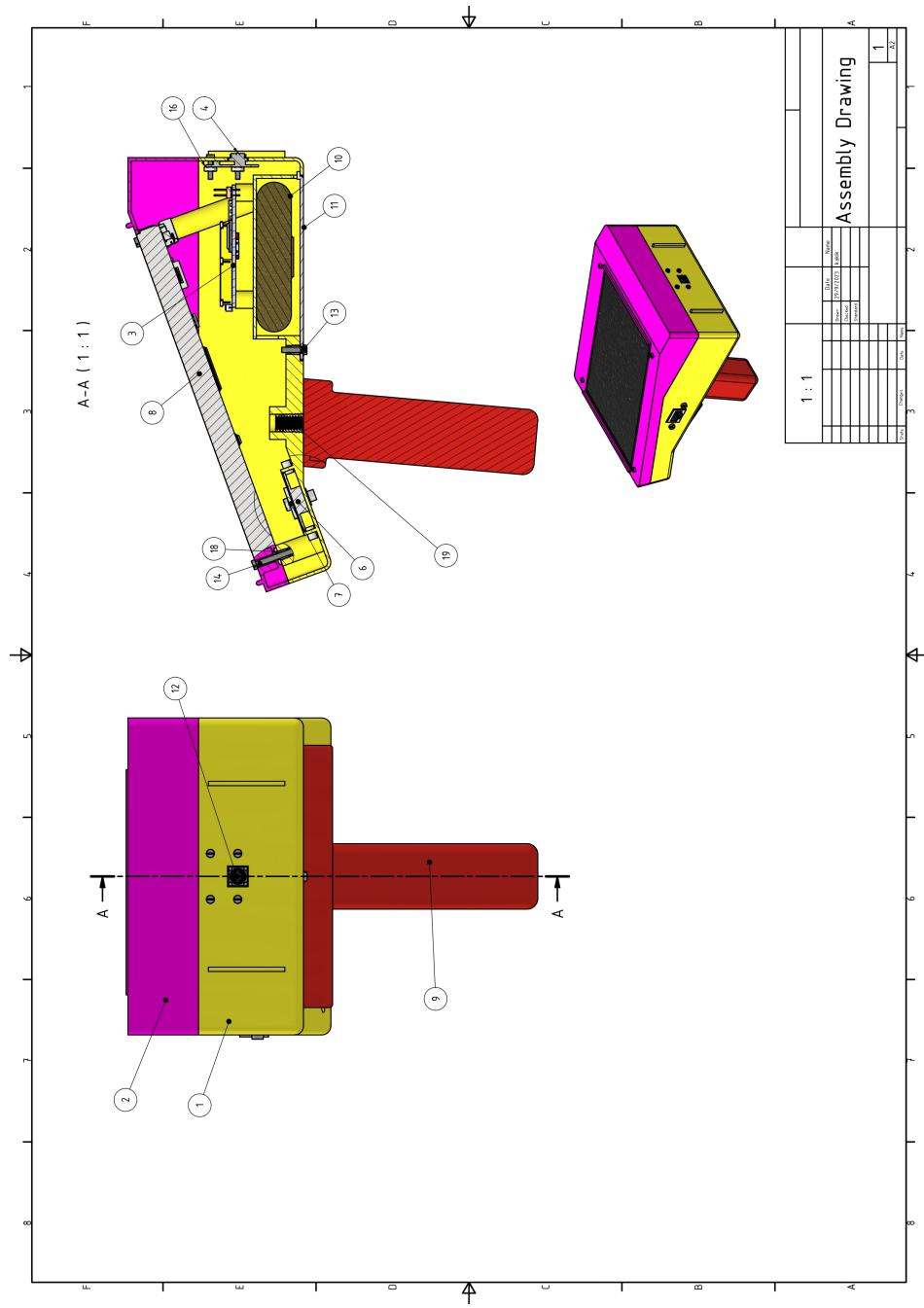


Figure A.1: Assembly Drawing

↓

PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	Main Body	
2	1	Top Cover	
3	1	Raspberry Pi 4	
4	1	Camera Module	
5	1	LAN Port	
6	1	Switch	
7	1	Switch Cover	
8	1	Screen	
9	1	Pistol Grip	
10	1	Battery	Veektomx
11	1	Battery Cover	
12	4	Screw DIN 84 M2x10	DIN 84 (ISO 1207) - A2-Cylinder head screws, ISO 1207
13	7	Screw DIN 84 M2.5x10	DIN 84 (ISO 1207) - A2-Cylinder head screws, ISO 1207
14	4	Screw DIN 84 M2.5x20	DIN 84 (ISO 1207) - A2-Cylinder head screws, ISO 1207
15	2	Screw DIN 84 M3x10	DIN 84 (ISO 1207) - A2-Cylinder head screws, ISO 1207
16	4	Nut M2	DIN 934
17	2	Nut M2.5	DIN 934
18	9	Threaded Insert M2.5	Ruthex
19	3	Threaded Insert 1/4"	Ruthex
20	2	Camera Screw 1/4"	

1 : 1						
Date	Name	Bill of Material			2 A4	
Drawn	29/9/2023 kakik					
Checked						
Standard						
State	Changes					Date

↑

Figure A.2: Bill of Materials

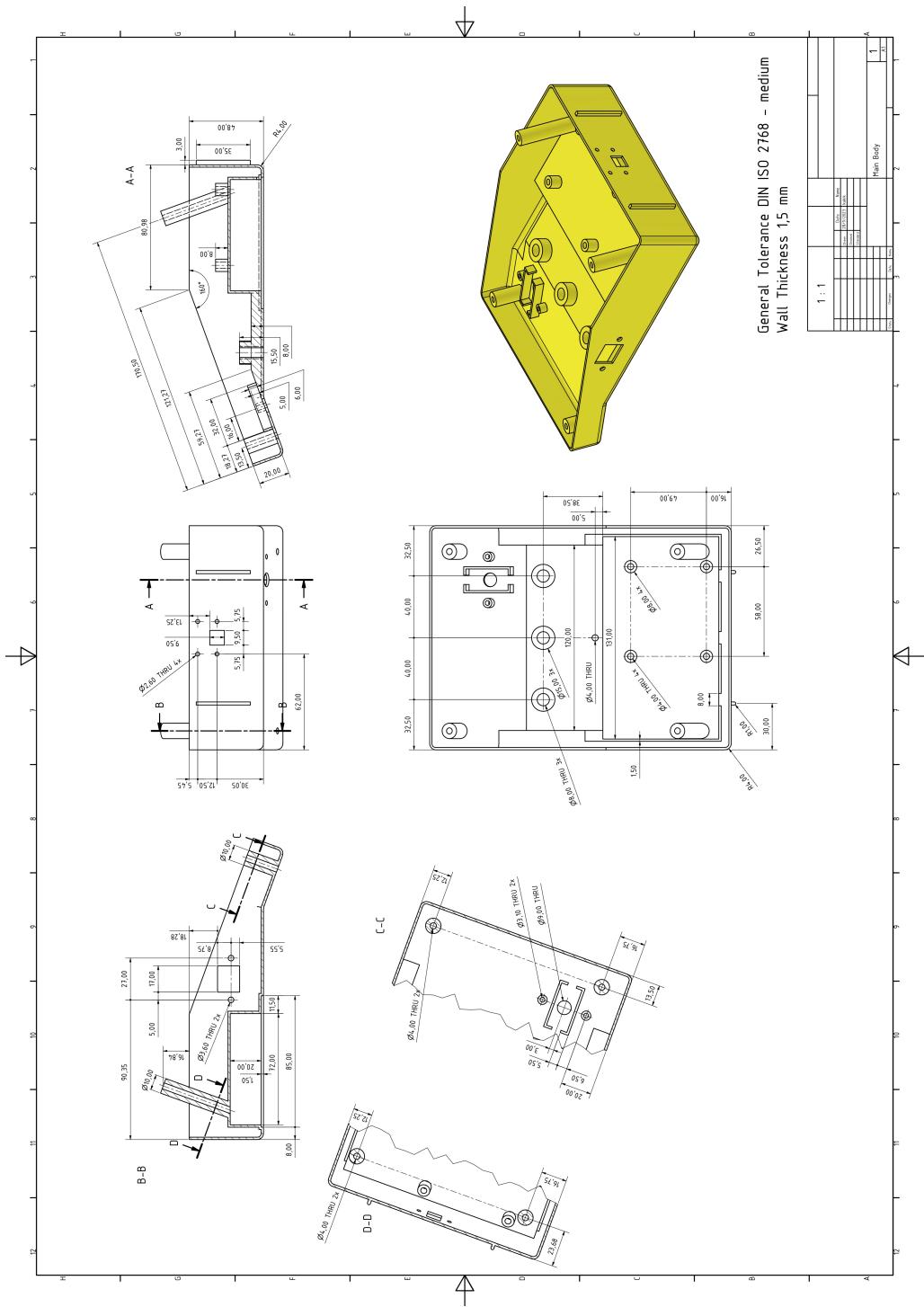


Figure A.3: Main Body Drawing

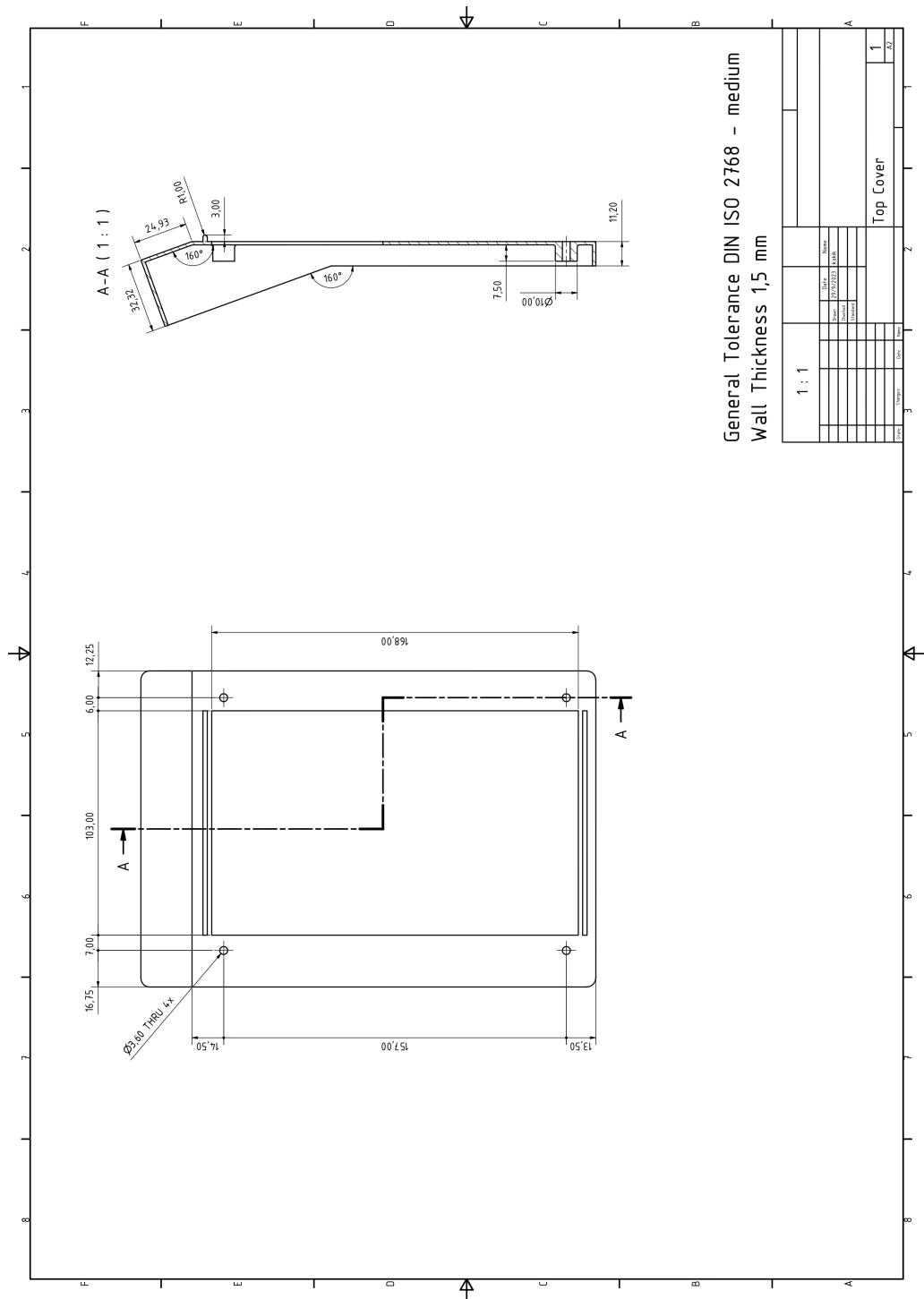


Figure A.4: Top Cover Drawing

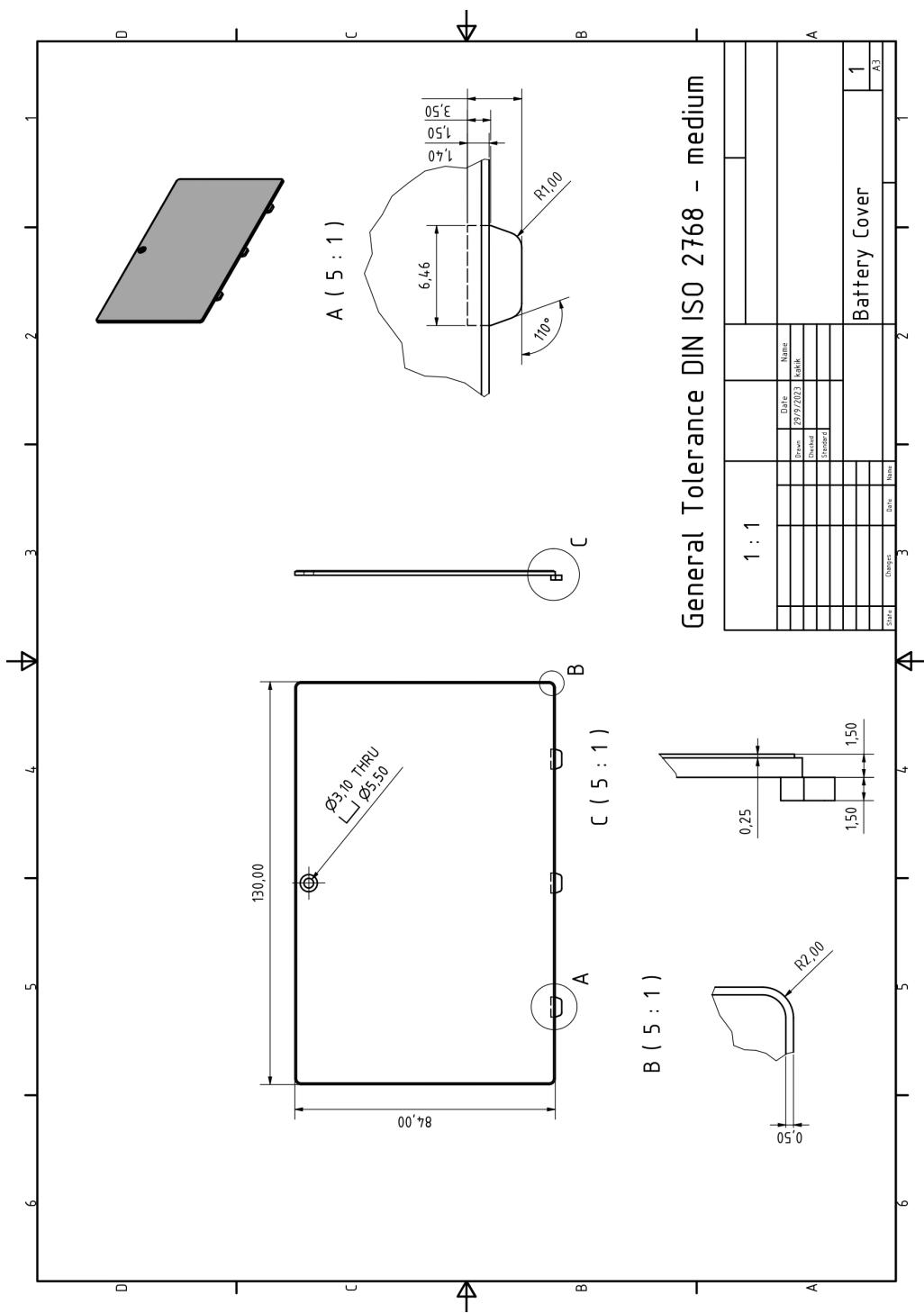


Figure A.5: Battery Cover Drawing

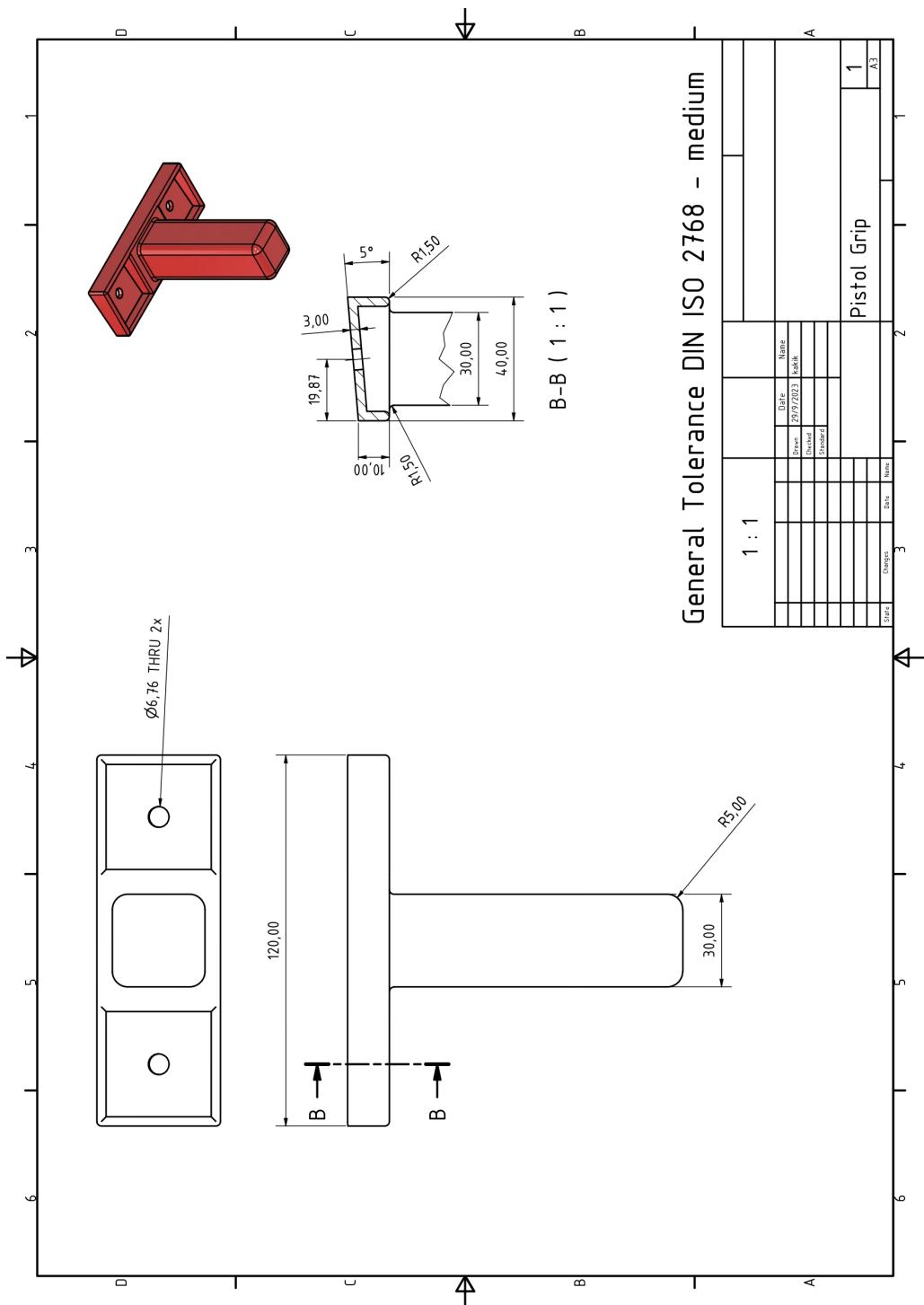


Figure A.6: Pistol Grip Drawing

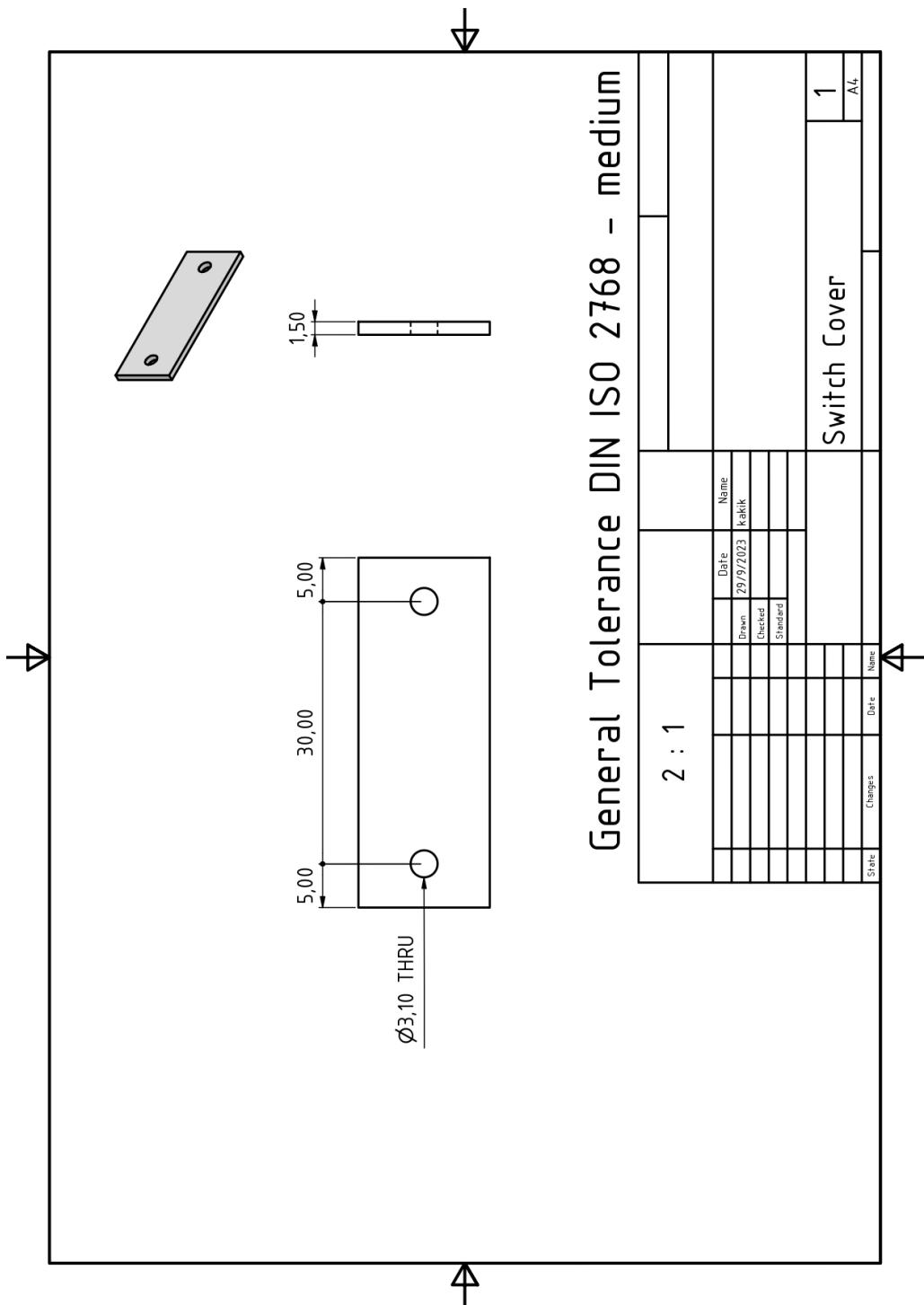


Figure A.7: Switch Cover Drawing

A.3 Technical Specifications

A.3.1 Original Prusa i3 MK3S+ 3D printer

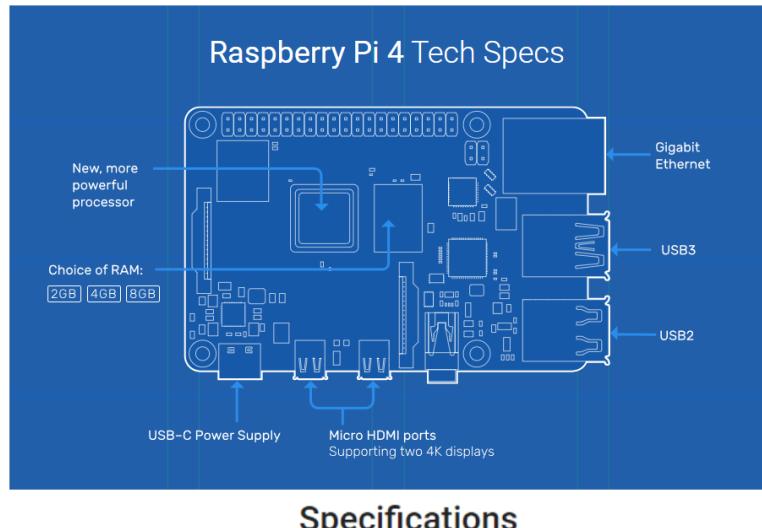
Technical Parameters

Build Volume	25×21×21 cm (9.84"×8.3"×8.3")
Layer height	0.05 - 0.35 mm
Nozzle	0.4mm default, wide range of other diameters/nozzles supported
Filament diameter	1.75 mm
Supported materials	Wide range of thermoplastics, including PLA, PETG, ASA, ABS, PC (Polycarbonate), CPE, PVA/BVOH, PVB, HIPS, PP (Polypropylene), Flex, nGen, Nylon, Carbon filled, Woodfill and other filled materials.
Max travel speed	200+ mm/s
Max nozzle temperature	300 °C / 572 °F
Max heatbed temperature	120 °C / 248 °F
Extruder	Direct Drive, Bondtech gears, E3D V6 hotend
Print surface	Removable magnetic steel sheets(*) with different surface finishes, heatbed with cold corners compensation
Printer dimensions (without spool)	7 kg, 500×550×400 mm; 19.6×21.6×15.7 in (X×Y×Z)
Power consumption	PLA settings: 80W / ABS settings: 120W



Figure A.8: Original Prusa i3 MK3S+ 3D printer

A.3.2 Raspberry Pi 4 Model B



Specifications

Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
 1GB, 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
 Gigabit Ethernet
 2 USB 3.0 ports; 2 USB 2.0 ports.
 Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
 2 x micro-HDMI® ports (up to 4kp60 supported)
 2-lane MIPI DSI display port
 2-lane MIPI CSI camera port
 4-pole stereo audio and composite video port
 H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
 OpenGL ES 3.1, Vulkan 1.0
 Micro-SD card slot for loading operating system and data storage
 5V DC via USB-C connector (minimum 3A*)
 5V DC via GPIO header (minimum 3A*)
 Power over Ethernet (PoE) enabled (requires separate PoE HAT)
 Operating temperature: 0 – 50 degrees C ambient

* A good quality 2.5A power supply can be used if downstream USB peripherals consume less than 500mA in total.

Figure A.9: Raspberry Pi 4 Model B Technical Specifications

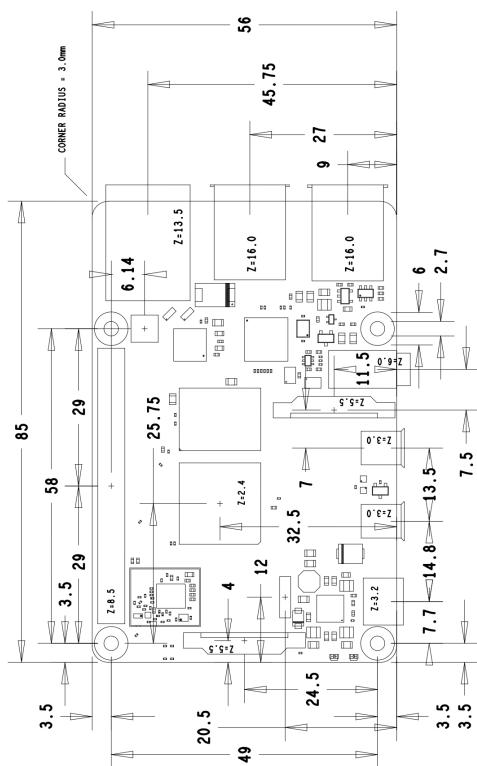


Figure A.10: Raspberry Pi 4 Model B Mechanical Drawing

A.3.3 Raspberry Pi Camera Module V2

	Camera Module v1	Camera Module v2	Camera Module 3	Camera Module 3 Wide	HQ Camera	GS Camera
Net price	\$25	\$25	\$25	\$35	\$50	\$50
Size	Around 25 × 24 × 9 mm	Around 25 × 24 × 9 mm	Around 25 × 24 × 11.5 mm	Around 25 × 24 × 12.4 mm	38 x 38 x 18.4mm (excluding lens)	38 x 38 x 19.8mm (29.5mm with adaptor and dust cap)
Weight	3g	3g	4g	4g	30.4g	34g (41g with adaptor and dust cap)
Still resolution	5 Megapixels	8 Megapixels	11.9 Megapixels	11.9 Megapixels	12.3 Megapixels	1.58 Megapixels
Video modes	1080p30, 720p60 and 640 × 480p60/90	1080p47, 1640 × 1232p41 and 640 × 480p206	2304 × 1296p56, 2304 × 1296p30 HDR, 1536 × 864p120	2304 × 1296p56, 2304 × 1296p30 HDR, 1536 × 864p120	2028 × 1080p50, 2028 × 1520p40 and 1332 × 990p120	1456 x 1088p60
Sensor	OmniVision OV5647	Sony IMX219	Sony IMX708	Sony IMX708	Sony IMX477	Sony IMX296
Sensor resolution	2592 × 1944 pixels	3280 × 2464 pixels	4608 × 2592 pixels	4608 × 2592 pixels	4056 × 3040 pixels	1456 × 1088 pixels
Sensor image area	3.76 × 2.74 mm	3.68 × 2.76 mm (4.6 mm diagonal)	6.45 × 3.63mm (7.4mm diagonal)	6.45 × 3.63mm (7.4mm diagonal)	6.287mm x 4.712 mm (7.9mm diagonal)	6.3mm diagonal
Pixel size	1.4 µm × 1.4 µm	1.12 µm × 1.12 µm	1.4 µm × 1.4 µm	1.4 µm × 1.4 µm	1.55 µm × 1.55 µm	3.45 µm × 3.45 µm
Optical size	1/4"	1/4"	1/2.43"	1/2.43"	1/2.3"	1/2.9"
Focus	Fixed	Adjustable	Motorized	Motorized	Adjustable	Adjustable
Depth of field	Approx 1 m to ∞	Approx 10 cm to ∞	Approx 10 cm to ∞	Approx 5 cm to ∞	N/A	N/A
Focal length	3.60 mm +/- 0.01	3.04 mm	4.74 mm	2.75 mm	Depends on lens	Depends on lens
Horizontal Field of View (FoV)	53.50 +/- 0.13 degrees	62.2 degrees	66 degrees	102 degrees	Depends on lens	Depends on lens
Vertical Field of View (FoV)	41.41 +/- 0.11 degrees	48.8 degrees	41 degrees	67 degrees	Depends on lens	Depends on lens
Focal ratio (F-Stop)	F2.9	F2.0	F1.8	F2.2	Depends on lens	Depends on lens
Maximum exposure times (seconds)	6 (legacy) / 0.97 (libcamera)	11.76	112	112	670.74	15.5
Lens Mount	N/A	N/A	N/A	N/A	C/CS- or M12-mount	C/CS
NoIR version available?	Yes	Yes	Yes	Yes	No	No

Figure A.11: Raspberry Pi Camera Module V2 Technical Specifications

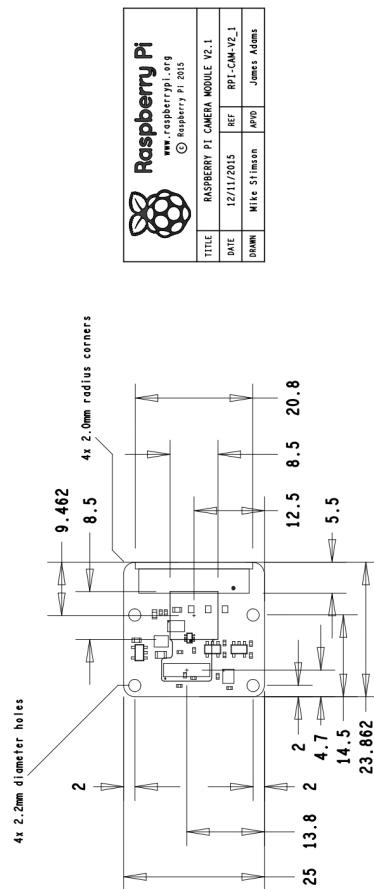


Figure A.12: Raspberry Pi Camera Module V2 Mechanical Drawing

A.3.4 Waveshare 7inch HDMI LCD (H)

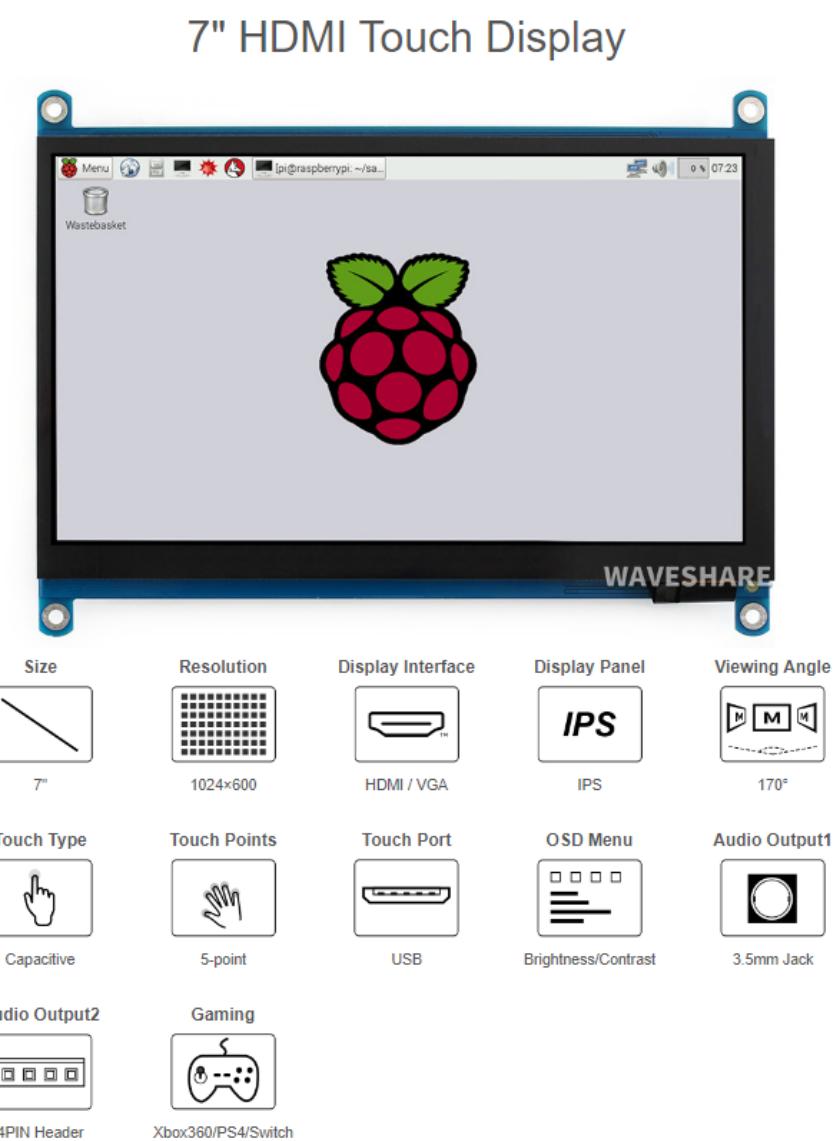


Figure A.13: Waveshare 7inch HDMI LCD (H) Technical Specifications -1

Connection Examples

Working With Raspberry Pi 4



Working With Raspberry Pi 3B+



Working With Raspberry Pi Zero W

Figure A.14: Waveshare 7inch HDMI LCD (H) Technical Specifications -2

Appearance And Dimensions

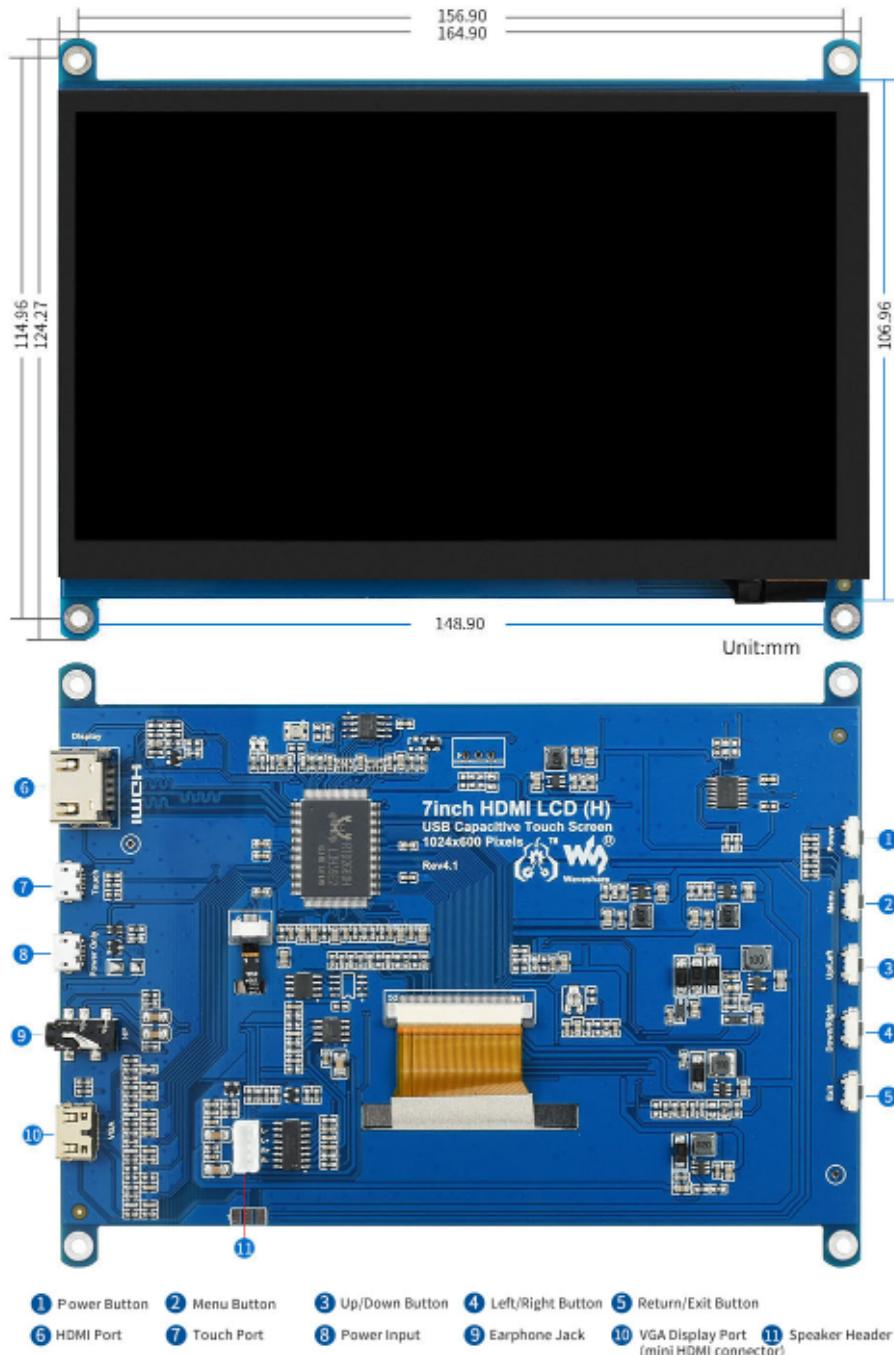


Figure A.15: Waveshare 7inch HDMI LCD (H) Technical Specifications -3

A.3.5 Veektomx VT103



Figure A.16: Veektomx VT103 Technical Specifications

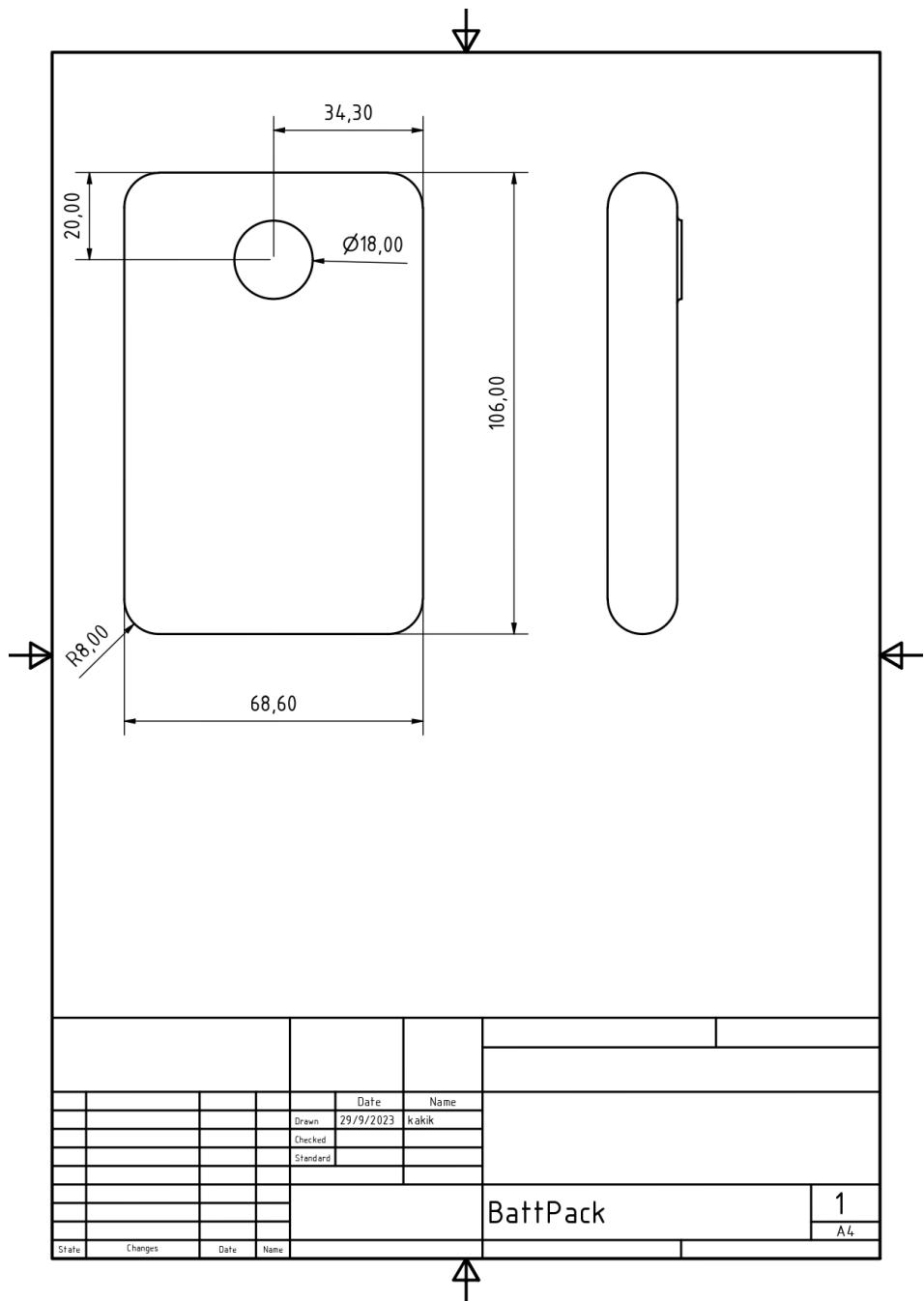


Figure A.17: Veektomx VT103 Mechanical Drawing

A.3.6 Ruthex Brass Inserts

STAFFELPREIS

VERGLEICH

RUTHEX®	ANDERE HERSTELLER
Cd FREE → FREI VON CADMIUM ✓	Cd Cadmium → ENTHALT CADMIUM X
Pb FREE → BLEIFREI ✓	Pb Lead → ENTHALT BLEI X
→ FÜR UNSERE UMWELT ✓	

RUTHEX® - PRODUKTIONSGEWINNIGKEIT

- Niedrige Drehzahl
- Scharfe Kanten
- Starkere Zugkraft

1 in sec.

KONKURRENZ - PRODUKTIONSGEWINNIGKEIT

- Höhe Drehzahl
- Unscharfe Kanten
- Schwache Zugkraft

1 in sec.

DEUTSCHE MARKE

ANWENDUNG

EINFACHES EINSETZEN DURCH WÄRME ODER ULTRASCHALL

STABILE GEWINDE FÜR IHR 3D DRUCK PRODUKT

GRÖÙE

Metrische ISO-Gewinde	Zollähnliche UNC-Gewinde	Ø d1	Ø d2	Ø d3	L	W
M2	#2-56	3,6	3,1	3,6	4,0	1,3
M2,5		4,6	3,9	4,0	5,7	1,6
M3 Short		4,6	3,9	4,0	4,0	1,6
M3x3x4 Voron		5,0	4,25	4,4	4,0	1,3
M3	#4-40	4,6	3,9	4,0	5,7	1,6
M4 Short		6,3	5,5	5,6	4,0	2,1
M4	#8-32	6,3	5,5	5,6	8,1	2,1
M5 Short		7,1	6,3	6,4	5,8	2,6
M5	#10-24	8,5	6,3	6,4	9,5	2,6
M6	#18-20	8,7	7,9	8,0	12,7	3,9
M8		10,1	9,5	9,6	12,7	4,5
	3/8"-16	12,6	11,8	11,9	12,7	6,0

Ø d1 Ø 43 min. w
Ø d2
Sack- oder DurchgangsgroÙe (Sack or Through hole)

Figure A.18: Ruthex Brass Inserts

A.4 Cost Calculation

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Cost Calculation

SetCurrencyUnits(*BaseCurrency*)
1 EUR = 1.0000 EUR

Constant

Filament cost per kg

$$C_{fil} := 29.99 \frac{\text{EUR}}{\text{kg}}$$

Printer power rating

$$P := 80 \frac{\text{W}}{\text{hr}}$$

Electricity Price

$$C_{el} := 0.23545 \frac{\text{EUR}}{\text{kW}}$$

Formula

Material Cost

$$C_m := m_{fil} \cdot C_{fil}$$

Electric Cost

$$C_e := t_p \cdot C_{el} \cdot P$$

Printing Cost

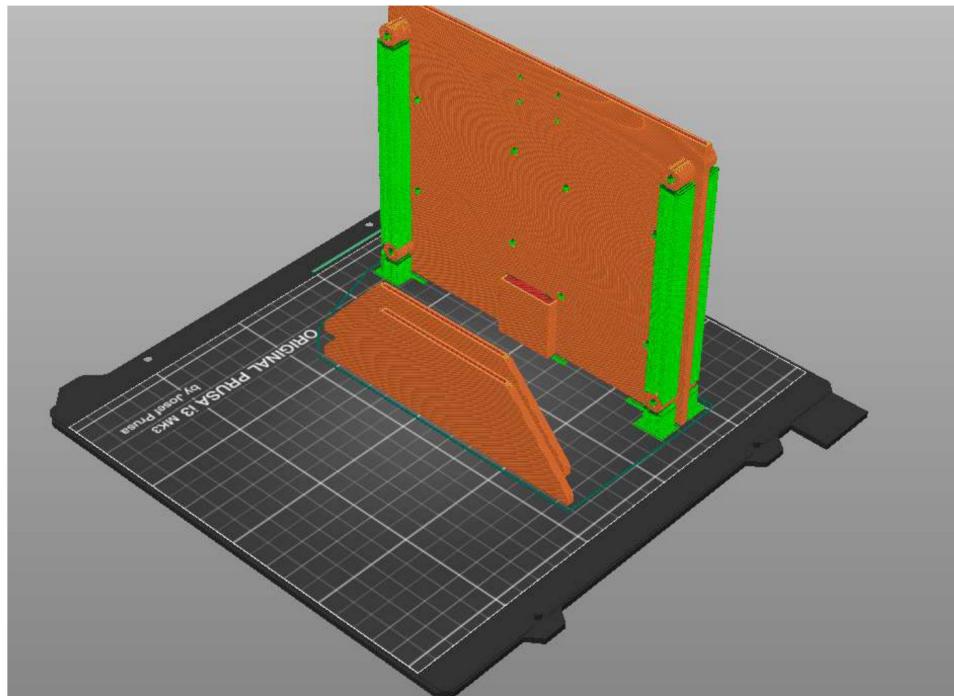
$$C_{print} := C_m + C_e$$

Figure A.19: Cost Calculation 1

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Variant 2

Base



$$m_{fil} := 175.07 \text{ g}$$

$$t_p := 6 \text{ hr} + 8 \text{ min}$$

$$C_m = 5.2503 \text{ EUR}$$

$$C_e = 0.1155 \text{ EUR}$$

$$C_{print} = 5.3659 \text{ EUR}$$

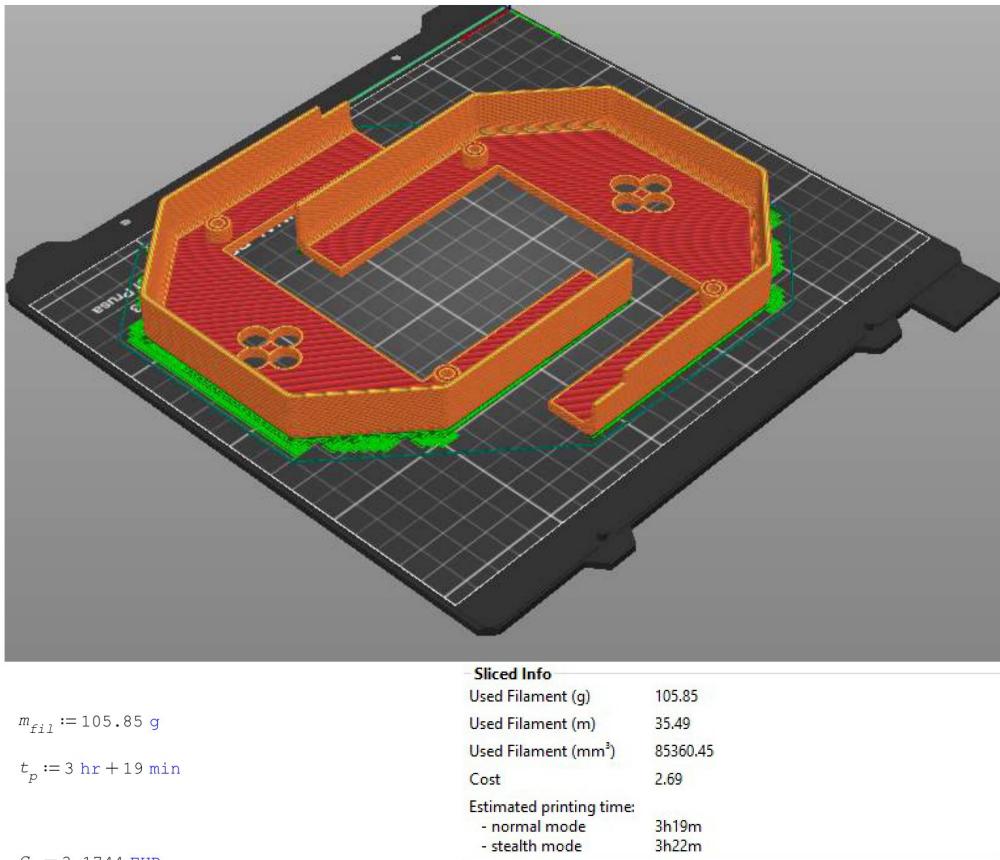
Sliced Info	
Used Filament (g)	175.07
Used Filament (m)	58.70
Used Filament (mm³)	141185.08
Cost	4.45
Estimated printing time:	
- normal mode	6h8m
- stealth mode	6h17m

Not for commercial use
2/19

Figure A.20: Cost Calculation 2

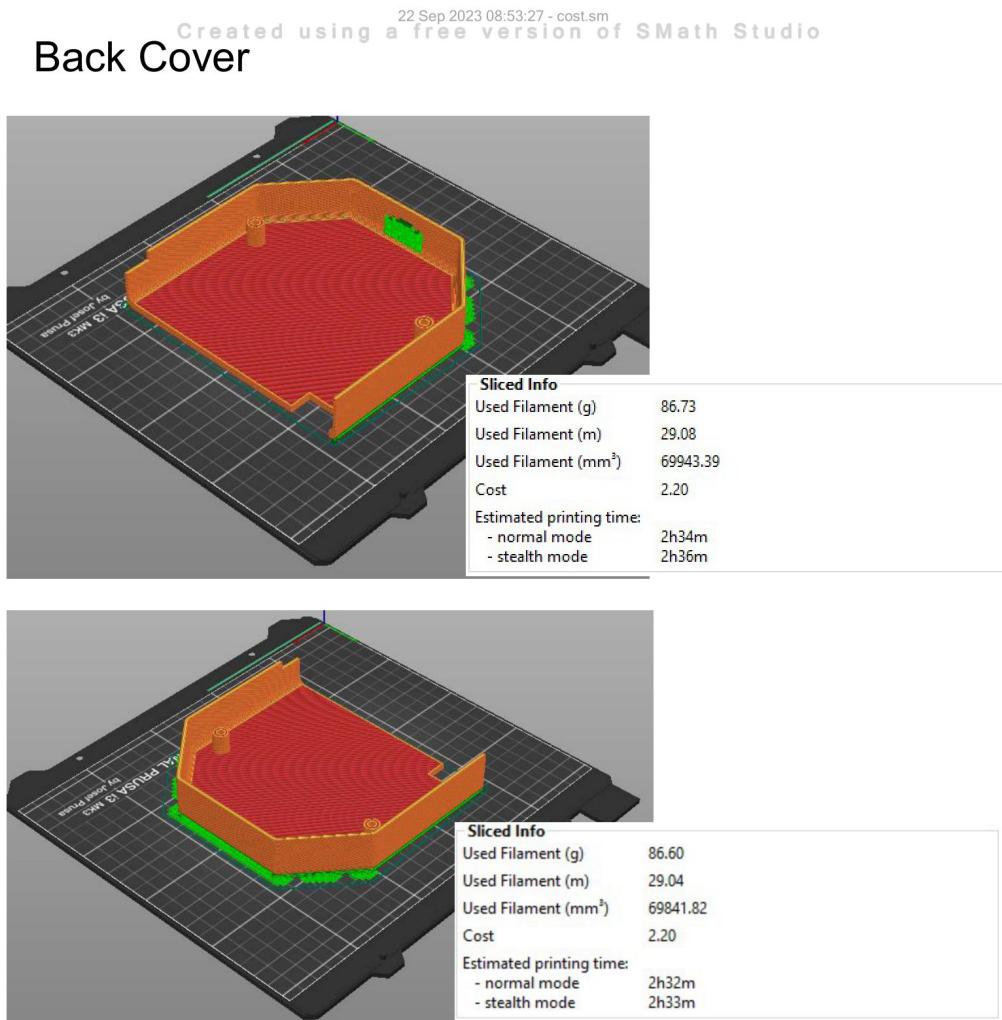
22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Top Cover



Not for commercial use
3/19

Figure A.21: Cost Calculation 3



$$m_{fil} := (86.73 + 86.6) \text{ g}$$

$$t_p := 2 \text{ hr} + 34 \text{ min} + 2 \text{ hr} + 32 \text{ min}$$

$$C_m = 5.1982 \text{ EUR}$$

$$C_e = 0.0961 \text{ EUR}$$

$$C_{print} = 5.2942 \text{ EUR}$$

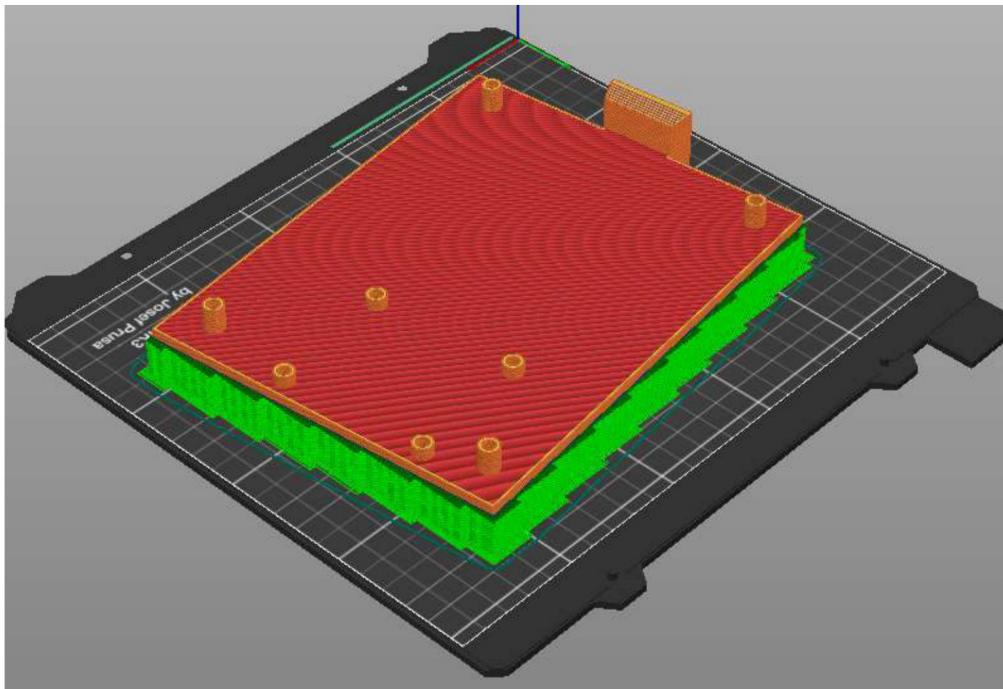
Not for commercial use
4/19

Figure A.22: Cost Calculation 4

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Variant 3

Base

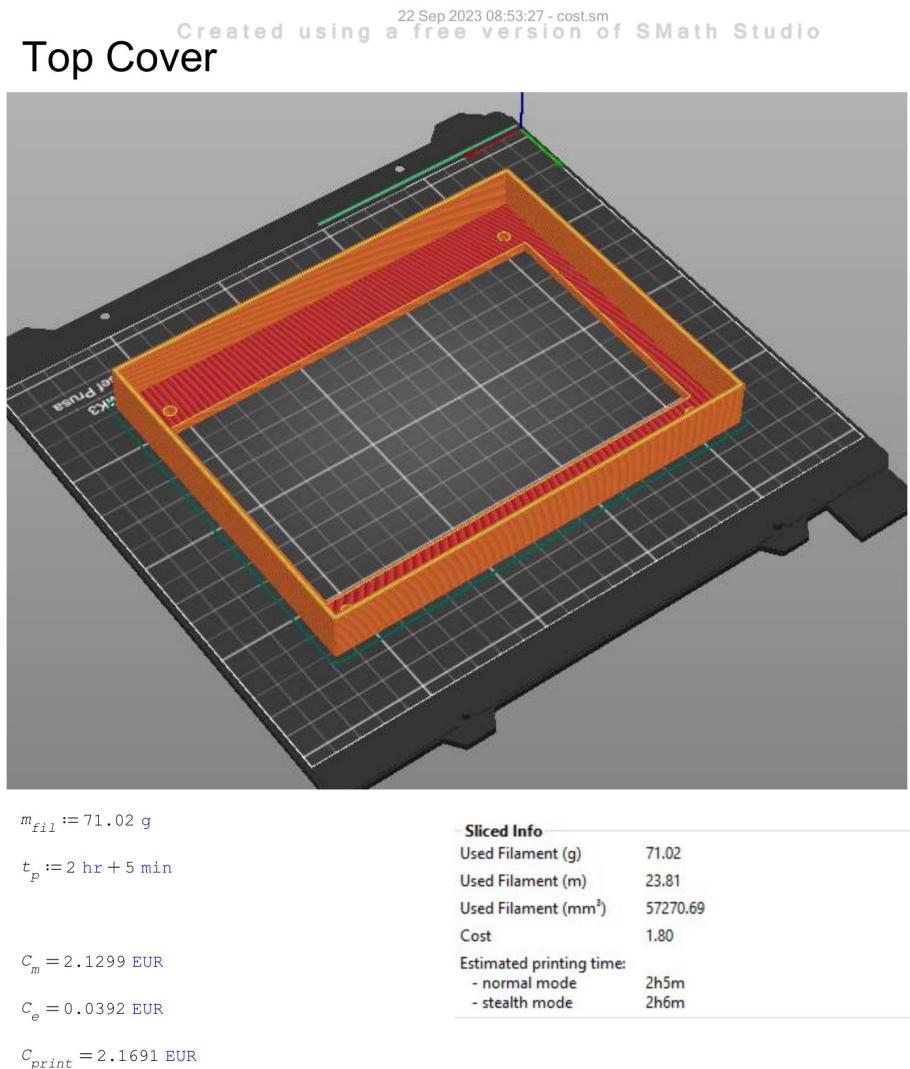


Sliced Info	
$m_{fil} := 222.96 \text{ g}$	Used Filament (g) 222.96
$t_p := 6 \text{ hr} + 40 \text{ min}$	Used Filament (m) 74.76
	Used Filament (mm^3) 179809.70
	Cost 5.66
	Estimated printing time:
	- normal mode 6h40m
	- stealth mode 6h44m

$C_m = 6.6866 \text{ EUR}$
 $C_e = 0.1256 \text{ EUR}$
 $C_{print} = 6.8121 \text{ EUR}$

Not for commercial use
5/19

Figure A.23: Cost Calculation 5

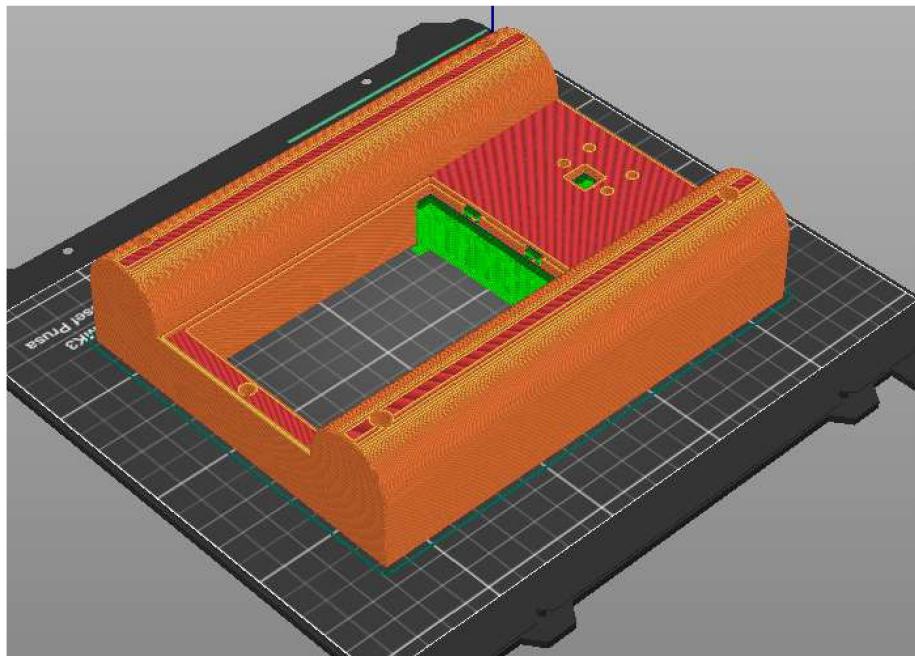


Not for commercial use
6/19

Figure A.24: Cost Calculation 6

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Back Cover



$m_{fil} := 303.02 \text{ g}$

$t_p := 9 \text{ hr} + 22 \text{ min}$

$C_m = 9.0876 \text{ EUR}$

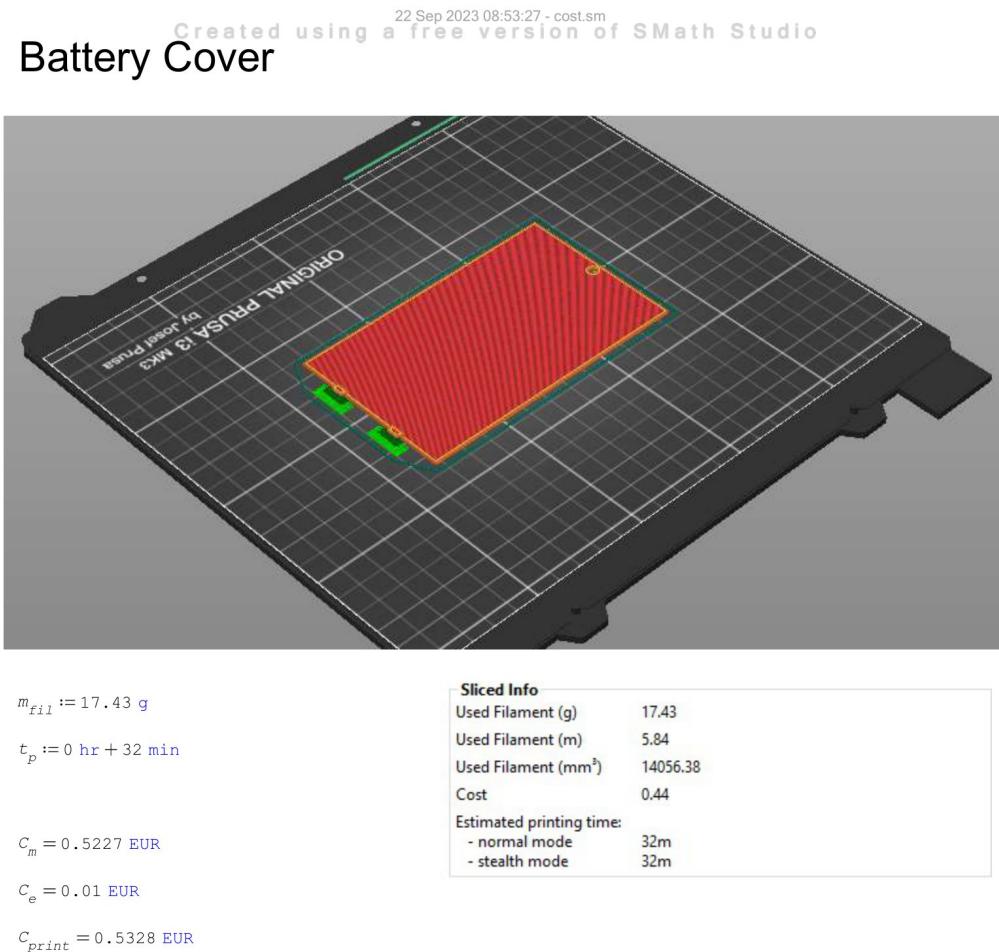
$C_e = 0.1764 \text{ EUR}$

$C_{print} = 9.264 \text{ EUR}$

Sliced Info	
Used Filament (g)	303.02
Used Filament (m)	101.60
Used Filament (mm^3)	244371.20
Cost	7.70
Estimated printing time:	
- normal mode	9h22m
- stealth mode	9h31m

Not for commercial use
7/19

Figure A.25: Cost Calculation 7



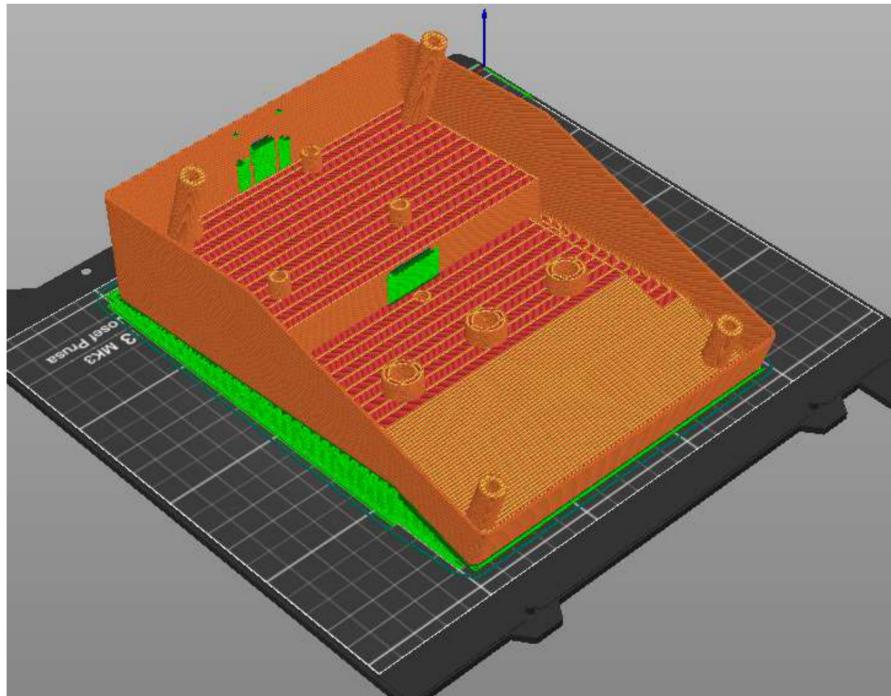
Not for commercial use
8 / 19

Figure A.26: Cost Calculation 8

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Variant 6

Main Body



$$m_{fil} := 240.3 \text{ g}$$

$$t_p := 8 \text{ hr} + 53 \text{ min}$$

$$C_m = 7.2066 \text{ EUR}$$

$$C_e = 0.1673 \text{ EUR}$$

$$C_{print} = 7.3739 \text{ EUR}$$

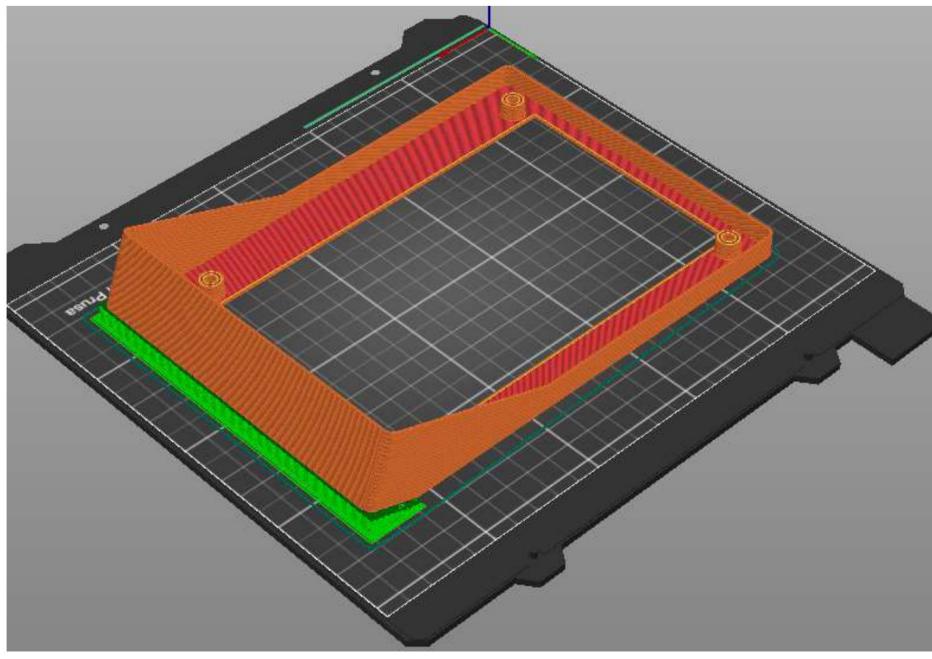
Sliced Info	
Used Filament (g)	240.30
Used Filament (m)	80.57
Used Filament (mm^3)	193788.10
Cost	6.10
Estimated printing time:	
- normal mode	8h53m
- stealth mode	9h3m

Not for commercial use
9/19

Figure A.27: Cost Calculation 9

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

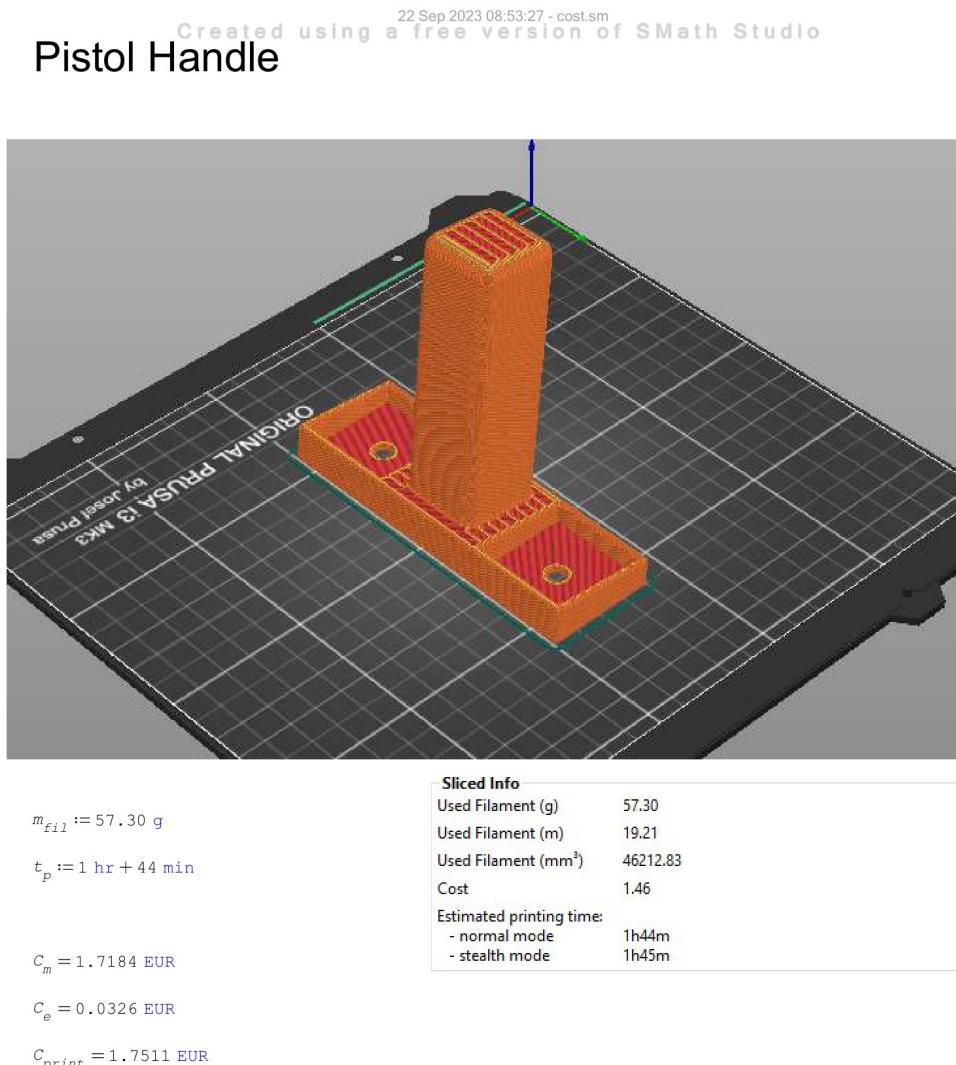
Top Cover



Sliced Info	
$m_{fil} := 55.32 \text{ g}$	Used Filament (g) 55.35
$t_p := 1 \text{ hr} + 52 \text{ min}$	Used Filament (m) 18.56
$C_m = 1.659 \text{ EUR}$	Used Filament (mm^3) 44638.88
$C_e = 0.0352 \text{ EUR}$	Cost 1.41
$C_{print} = 1.6942 \text{ EUR}$	Estimated printing time: - normal mode 1h52m - stealth mode 1h54m

Not for commercial use
10 / 19

Figure A.28: Cost Calculation 10

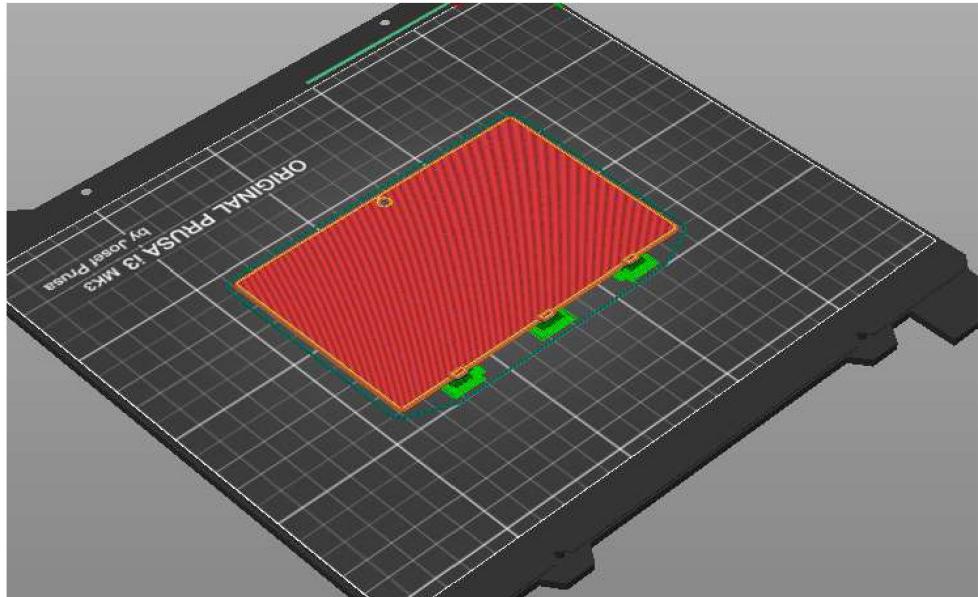


Not for commercial use
11 / 19

Figure A.29: Cost Calculation 11

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Battery Cover



Sliced Info	
Used Filament (g)	22.21
Used Filament (m)	7.45
Used Filament (mm³)	17910.60
Cost	0.56
Estimated printing time:	
- normal mode	40m
- stealth mode	40m

$C_m = 0.6661 \text{ EUR}$

$C_e = 0.0126 \text{ EUR}$

$C_{print} = 0.6786 \text{ EUR}$

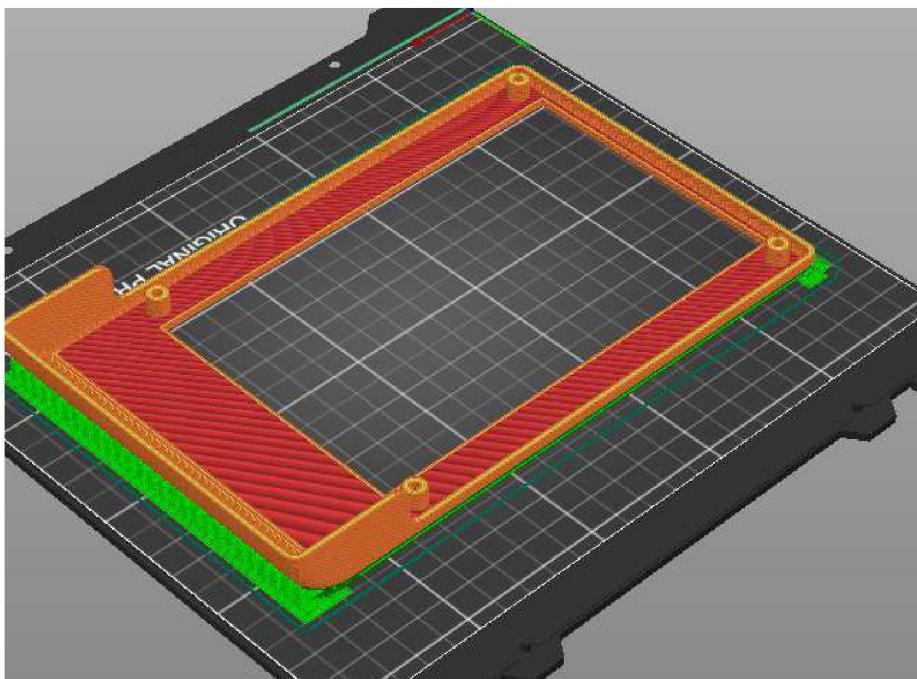
Not for commercial use
12 / 19

Figure A.30: Cost Calculation 12

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Variant 7

Top Cover



$m_{fil} := 61.5 \text{ g}$

$t_p := 2 \text{ hr} + 5 \text{ min}$

$C_m = 1.8444 \text{ EUR}$

$C_e = 0.0392 \text{ EUR}$

$C_{print} = 1.8836 \text{ EUR}$

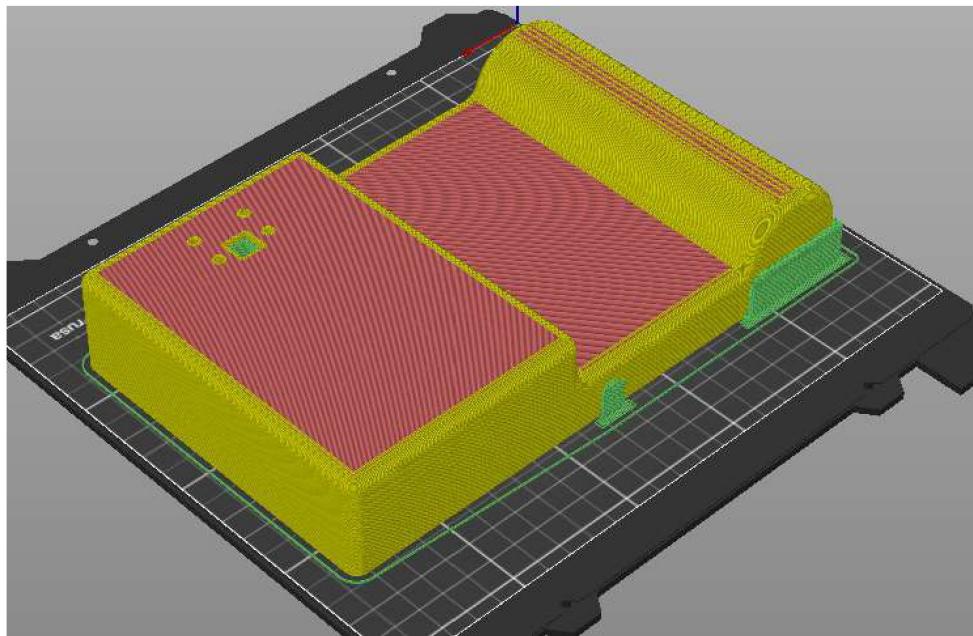
Sliced Info	
Used Filament (g)	61.50
Used Filament (m)	20.62
Used Filament (mm³)	49593.24
Cost	1.56
Estimated printing time:	
- normal mode	2h5m
- stealth mode	2h7m

Not for commercial use
13 / 19

Figure A.31: Cost Calculation 13

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Back Cover



$m_{fil} := 380.69 \text{ g}$

$t_p := 11 \text{ hr} + 49 \text{ min}$

$C_m = 11.4169 \text{ EUR}$

$C_e = 0.2226 \text{ EUR}$

$C_{print} = 11.6395 \text{ EUR}$

Sliced Into	
Used Filament (g)	380.69
Used Filament (m)	127.64
Used Filament (mm^3)	307011.60
Cost	9.67
Estimated printing time:	
- normal mode	11h49m
- stealth mode	11h57m

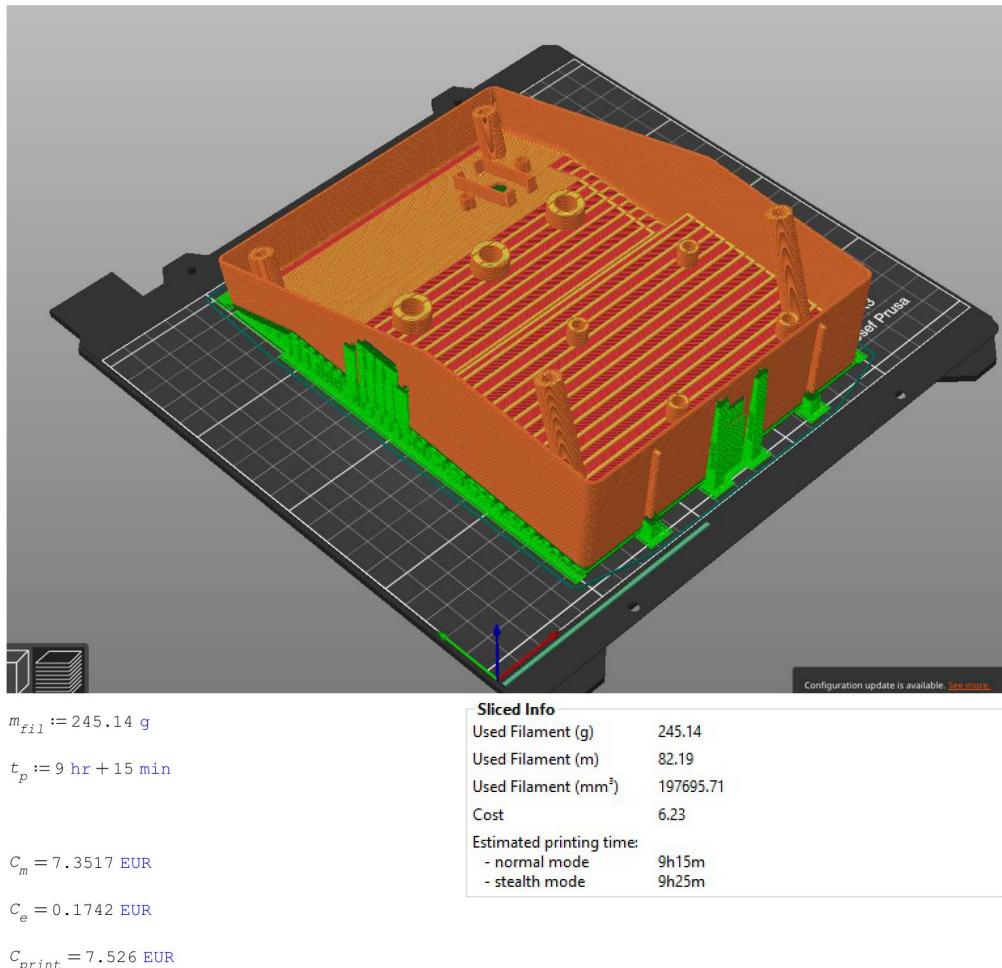
Not for commercial use
14 / 19

Figure A.32: Cost Calculation 14

22 Sep 2023 08:53:27 - cost.sm
Created using a free version of SMath Studio

Variant 6 Final

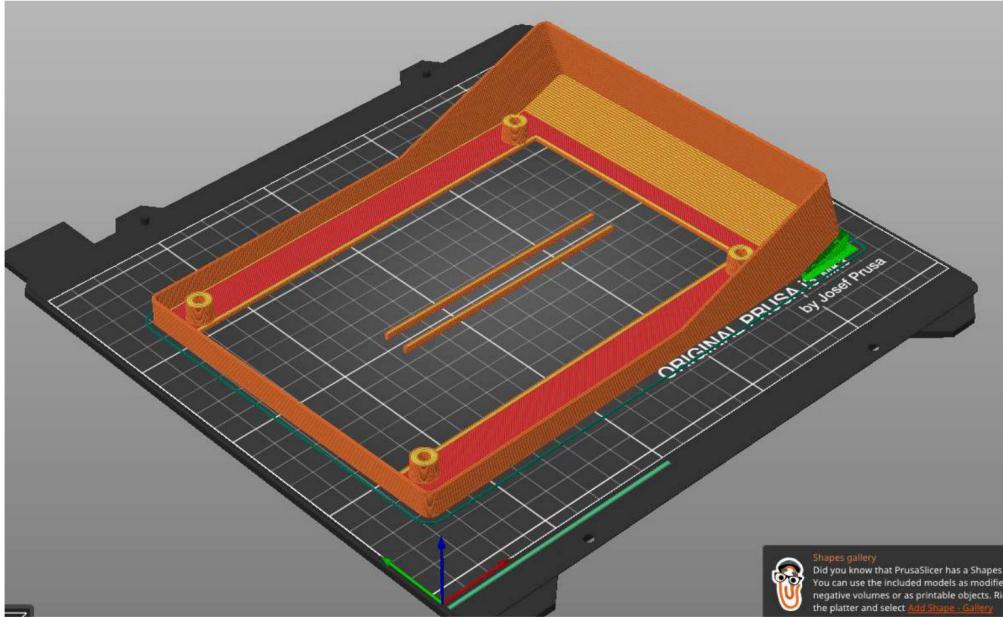
Main Body



Not for commercial use
15 / 19

Figure A.33: Cost Calculation 15

22 Sep 2023 08:53:27 - cost.sm
 Created using a free version of SMath Studio
Top Cover



- Sliced Info

Used Filament (g)	57.71
Used Filament (m)	19.35
Used Filament (mm^3)	46541.58
Cost	1.47
Estimated printing time:	
- normal mode	2h0m
- stealth mode	2h2m

$m_{fil} := 57.71 \text{ g}$

$t_p := 2 \text{ hr} + 0 \text{ min}$

$C_m = 1.7307 \text{ EUR}$

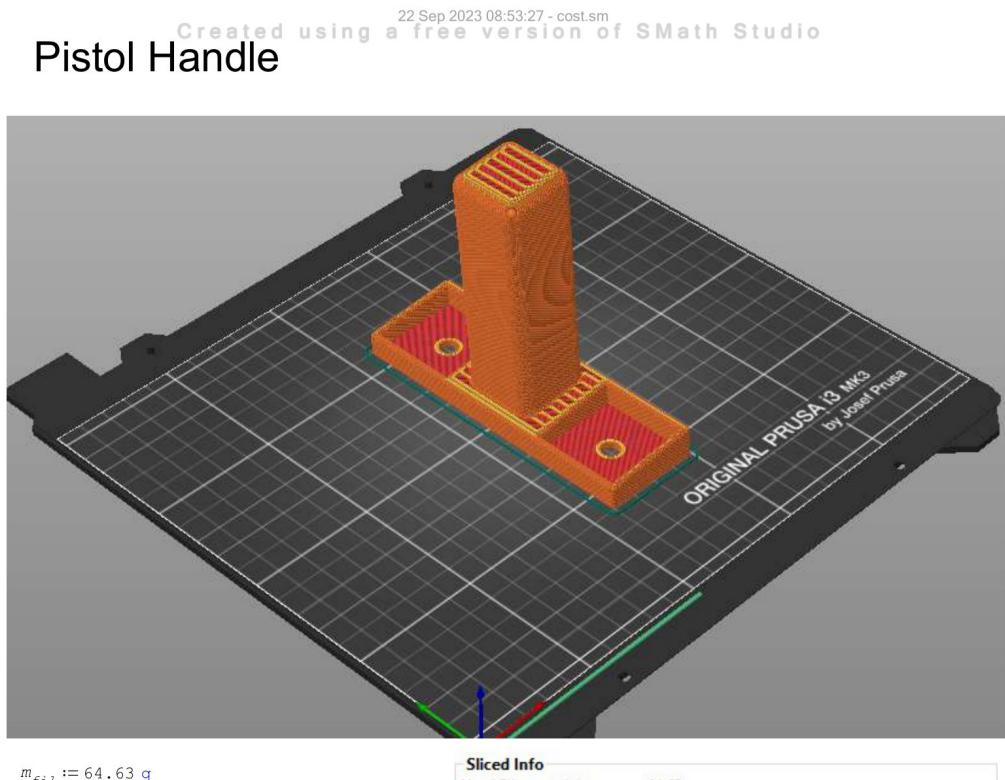
$C_e = 0.0377 \text{ EUR}$

$C_{print} = 1.7684 \text{ EUR}$



Not for commercial use
 16 / 19

Figure A.34: Cost Calculation 16



$$m_{fil} := 64.63 \text{ g}$$

$$t_p := 1 \text{ hr} + 59 \text{ min}$$

$$C_m = 1.9383 \text{ EUR}$$

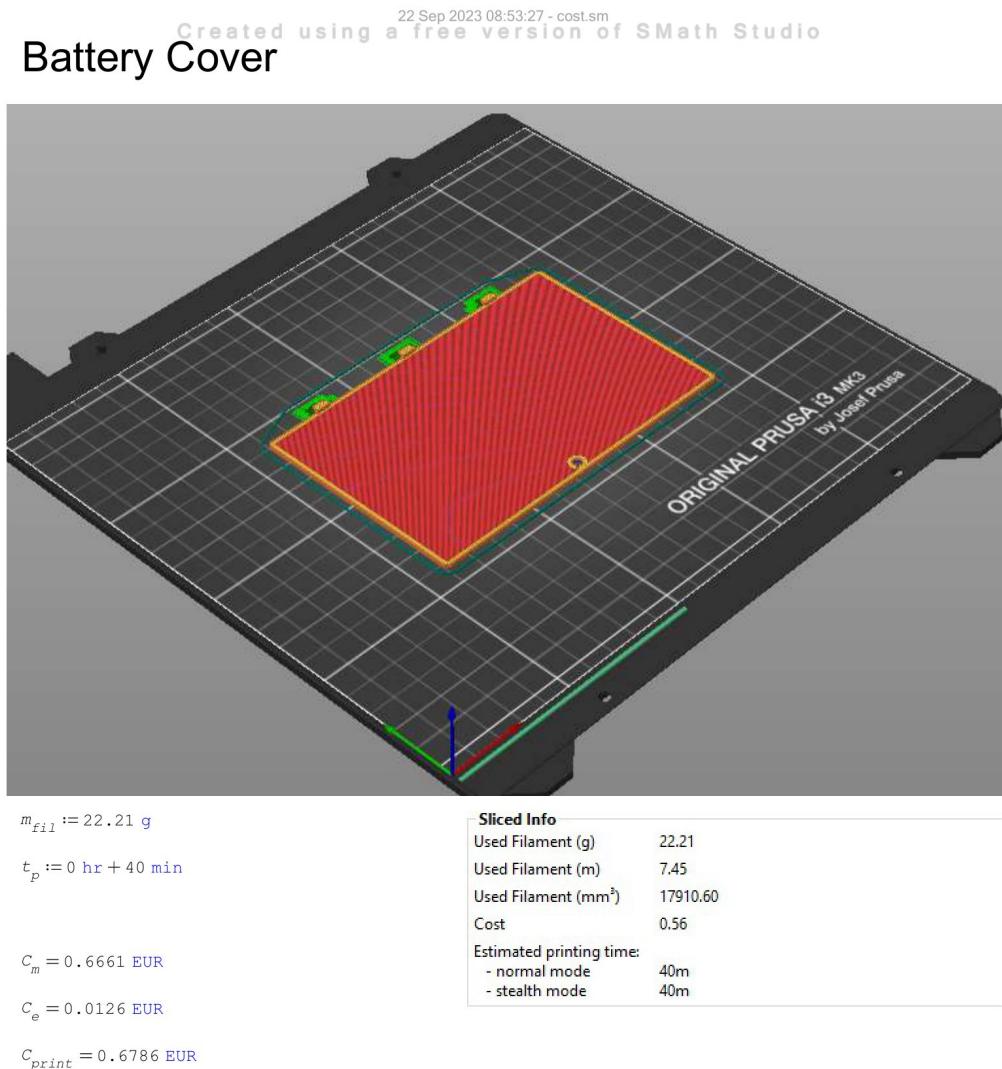
$$C_e = 0.0374 \text{ EUR}$$

$$C_{print} = 1.9756 \text{ EUR}$$

Sliced Info	
Used Filament (g)	64.63
Used Filament (m)	21.67
Used Filament (mm³)	52121.02
Cost	1.64
Estimated printing time:	
- normal mode	1h59m
- stealth mode	2h1m

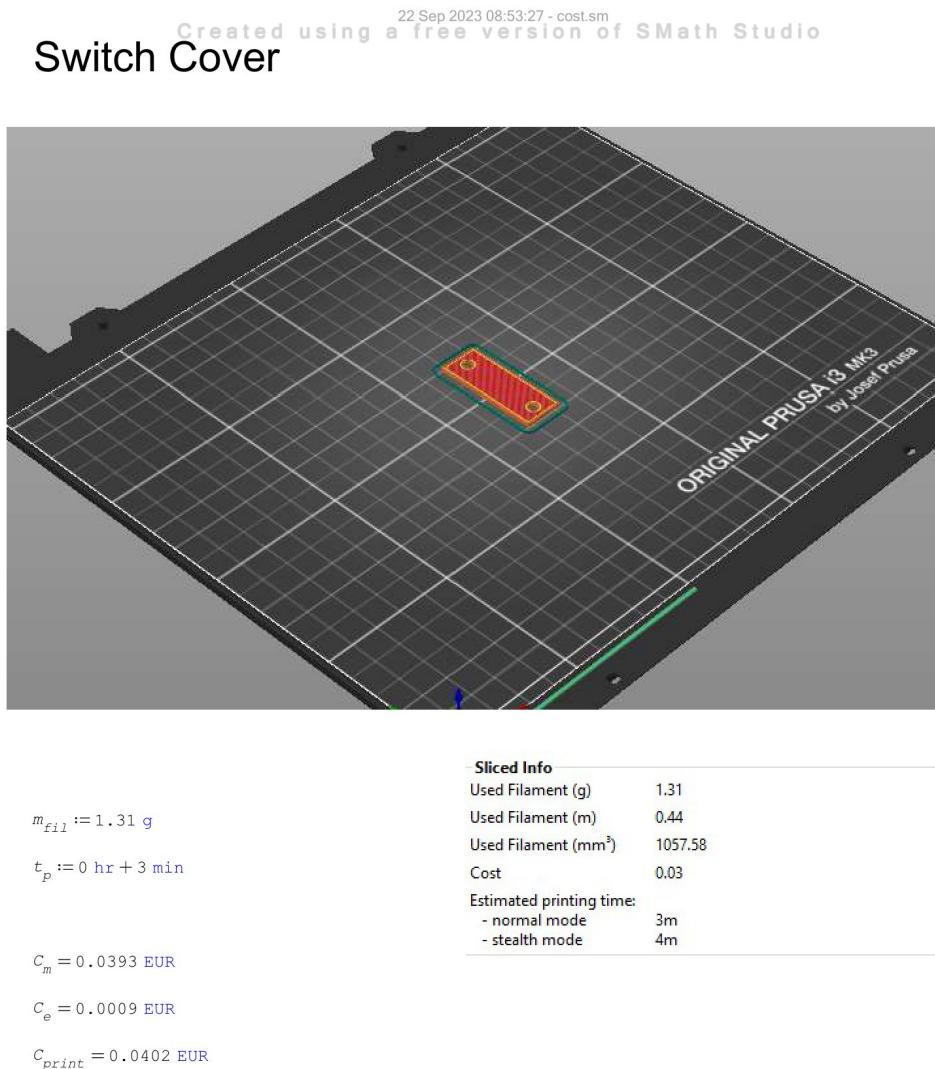
Not for commercial use
17 / 19

Figure A.35: Cost Calculation 17



Not for commercial use
18 / 19

Figure A.36: Cost Calculation 18



Not for commercial use
19 / 19

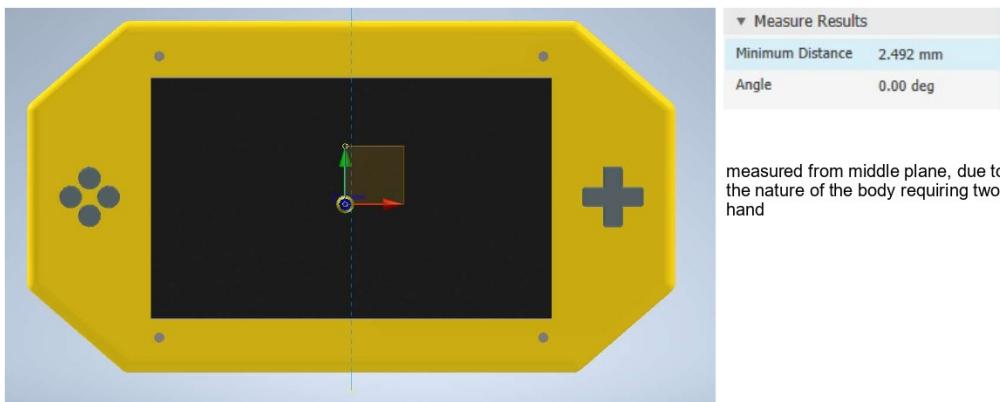
Figure A.37: Cost Calculation 19

A.5 Evaluation Data

Created using a free version of SMath Studio
1 Oct 2023 17:07:41 - eval.sm
Evaluation Data

Variant 2

Center of gravity



Device Weight



Ease of Assembly / Number of components

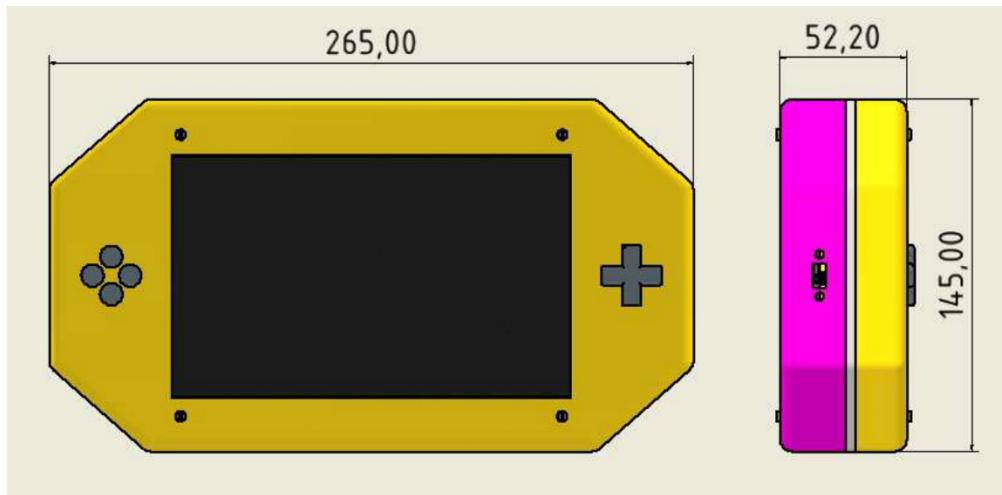


Swappable Parts

1. Quick change plate

Figure A.38: Evaluation 1

Created using a free version of SMath Studio
Device Size

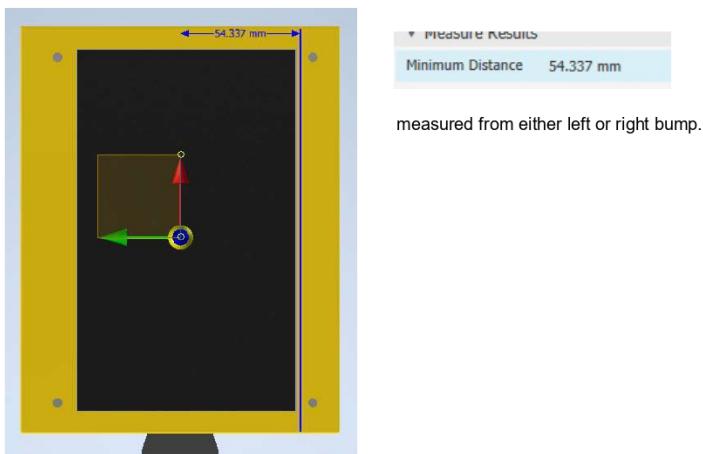


Not for commercial use
2/8

Figure A.39: Evaluation 2

Variant 3

Center of gravity



Device Weight



Ease of Assembly / Number of components



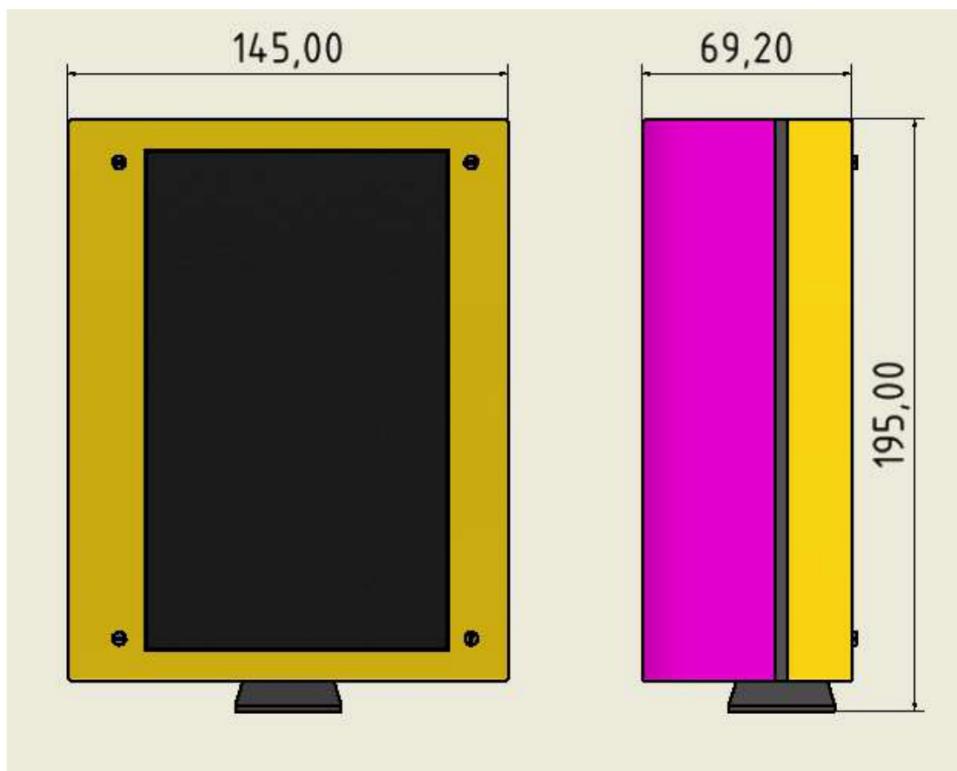
Swappable Parts

1. Battery

Not for commercial use
3/8

Figure A.40: Evaluation 3

Created using a free version of SMath Studio
Device Size

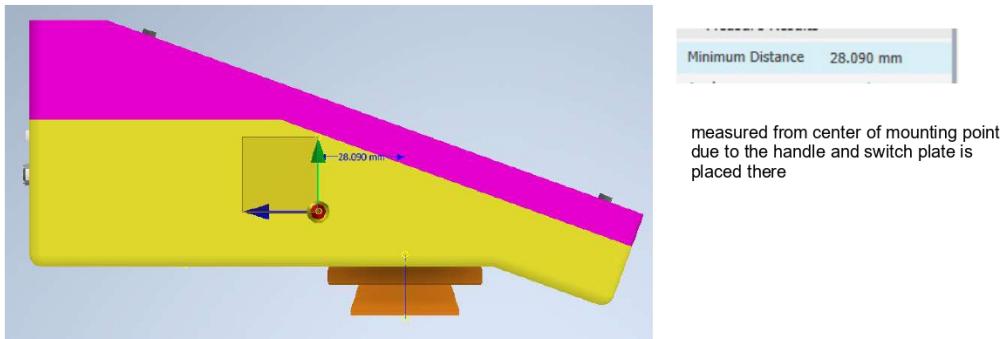


Not for commercial use
4 / 8

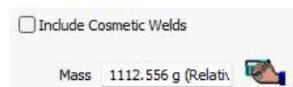
Figure A.41: Evaluation 4

Variant 6
Created using a free version of SMath Studio
1 Oct 2023 17:07:41 - eval.sm

Center of gravity



Device Weight



Ease of Assembly / Number of components



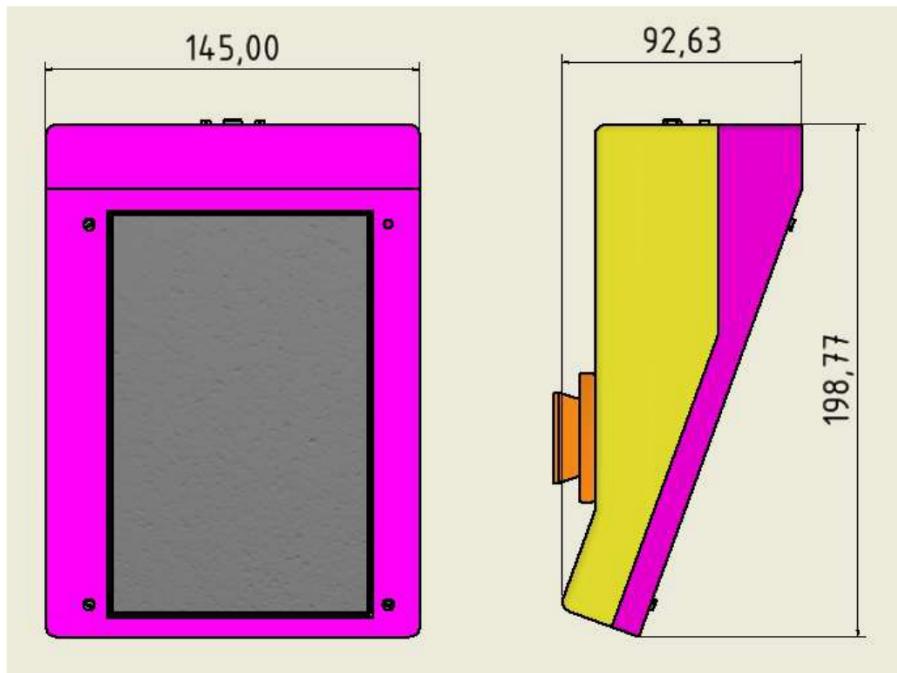
Swappable Parts

1. Battery
2. Switch plate
3. Handle

Not for commercial use
5/8

Figure A.42: Evaluation 5

1 Oct 2023 17:07:41 - eval.sm
Created using a free version of SMath Studio
Device Size

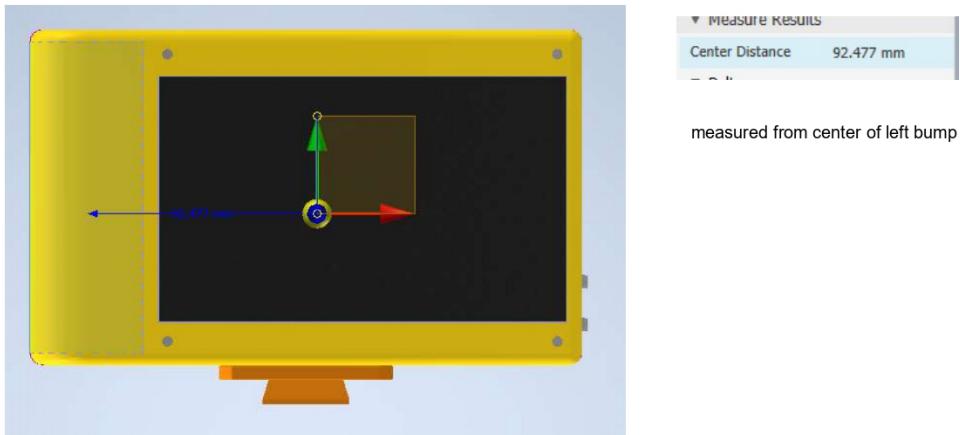


Not for commercial use
6/8

Figure A.43: Evaluation 6

Variant 7
Created using a free version of SMath Studio
1 Oct 2023 17:07:41 - eval.sm

Center of gravity



Device Weight



Ease of Assembly / Number of components



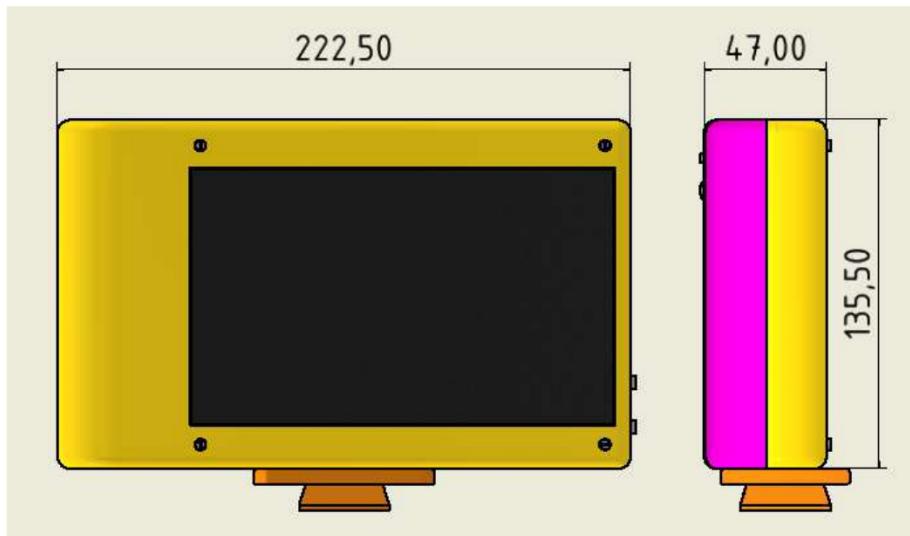
Swappable Parts

1. Switch plate

Not for commercial use
7/8

Figure A.44: Evaluation 7

Created using a free version of SMath Studio
Device Size



Not for commercial use
8/8

Figure A.45: Evaluation 8

A.6 Documentation

- Documentation
- Repository

A.7 C++ Unit Tests for Bank Account Management

Bank Account Class

```
1 class BankAccount {
2 private:
3     double balance;
4
5 public:
6     BankAccount() : balance(0.0) {}
7
8     void deposit(double amount) {
9         if (amount < 0) {
10             throw std::invalid_argument("Deposit amount
11                 must be positive");
12         }
13         balance += amount;
14     }
15
16     void withdraw(double amount) {
17         if (amount < 0) {
18             throw std::invalid_argument("Withdrawal amount
19                 must be positive");
20         }
21         if (amount > balance) {
```

```
20         throw std::runtime_error("Insufficient funds
21             for withdrawal");
22
23     balance -= amount;
24
25     double getBalance() const {
26         return balance;
27     }
28 }
```

Unit Tests

Test Case: Deposit

```
1 TEST(BankAccountTest, Deposit) {
2     BankAccount account;
3     account.deposit(100.0);
4     EXPECT_EQ(account.getBalance(), 100.0);
5 }
```

Explanation: This test verifies that funds can be successfully deposited into the account. It creates an instance of `BankAccount`, deposits 100.0 units, and then checks if the balance matches the expected value of 100.0.

Test Case: Withdraw

```
1 TEST(BankAccountTest, Withdraw) {
2     BankAccount account;
3     account.deposit(100.0);
4     account.withdraw(50.0);
5     EXPECT_EQ(account.getBalance(), 50.0);
6 }
```

Explanation: This test ensures that withdrawals are processed correctly. It first deposits 100.0 units into the account, then attempts to withdraw 50.0 units. The test verifies if the balance is now 50.0 units.

Test Case: Withdraw Too Much

```
1 TEST(BankAccountTest, WithdrawTooMuch) {
2     BankAccount account;
3     account.deposit(100.0);
4
5     EXPECT_THROW(account.withdraw(150.0), std:::
6         runtime_error);
7     EXPECT_EQ(account.getBalance(), 100.0);
}
```

Explanation: This test examines the scenario where an attempt is made to withdraw an amount greater than the available balance. It first deposits 100.0 units and then tries to withdraw 150.0 units. The test expects a `std::runtime_error` to be thrown, and ensures that the balance remains unchanged.

Test Case: Deposit Negative

```
1 TEST(BankAccountTest, DepositNegative) {
2     BankAccount account;
3
4     EXPECT_THROW(account.deposit(-50.0), std:::
5         invalid_argument);
6     EXPECT_EQ(account.getBalance(), 0.0);
7 }
```

Explanation: This test handles the scenario where an attempt is made to deposit a negative amount. It expects a `std::invalid_argument` exception to be thrown. The test also verifies that the account balance remains unaffected.

A.8 User Manual

Also available online at [Google Drive](#).



SpeedCameraPi

User Manuals

v1.3.0

last updated: October 24, 2023

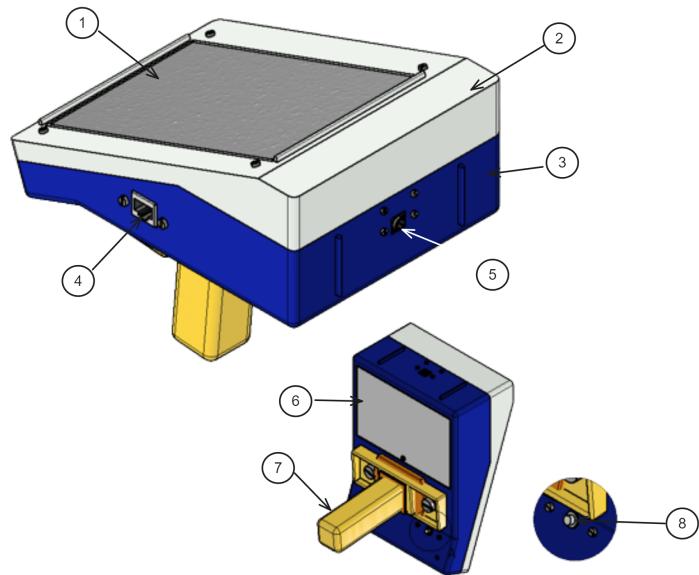
Contents

1	Hardware Overview	1
2	Quick Start	3
2.1	Power On	3
2.2	Capture Image	4
2.3	Calibrate	4
2.4	Select ROI	4
2.5	Measure	5
3	Panel Overview	6
3.1	Main Layout	7
3.2	Capture Panel	8
3.2.1	Instructions	9
3.3	Automatic Lane Calibration Panel	11
3.3.1	Instructions	12
3.4	Manual Lane Calibration Panel	13
3.4.1	Instructions	14
3.5	Distance Calibration Panel	15
3.5.1	Instructions	16
3.6	Roi Panel	17
3.6.1	Instructions	17
3.7	Result Panel	19
3.7.1	Instructions	20
3.8	Color Calibration Panel	21
3.8.1	Instructions	22
3.9	Trim Data Panel	23
3.9.1	Instructions	23

CONTENTS

3.10 Settings Panel	25
3.10.1 Camera	25
3.10.2 Sensor	26
3.10.3 Capture	26
3.10.4 Model	27
3.10.5 Optical Flow	27
3.10.6 Measurement	28
3.10.7 Preview	29
3.10.8 RANSAC	29
3.10.9 Blue HSV	30
3.10.10 Yellow HSV	30
3.10.11 Threads	31

1 Hardware Overview

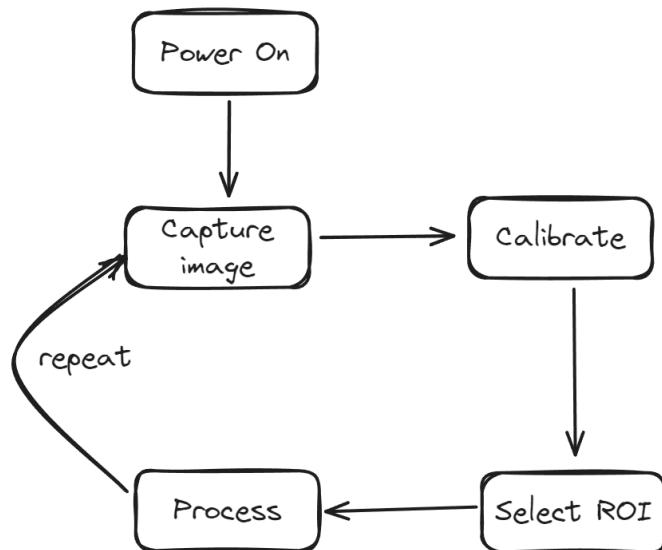


Hardware Overview

Number	Function
1	Touch Screen
2	Top Cover
3	Main Body
4	LAN Port
5	Camera
6	Battery Cover
7	Handle
8	Switch

2 Quick Start

This chapter describes how to use the device. Figure below shows the overall process of using the device.



2.1 Power On

1. Press the power button to turn on the device.
2. The device will boot up and the main interface will be displayed.

2.2 Capture Image

1. Make sure that you are on the Capture Panel.
2. Press the Capture button to capture an image.
3. The captured images will be displayed on the screen.

2.3 Calibrate

1. From the Capture Panel, press the Calibrate button to go to the Calibration Panel.
2. Once you are on the Calibration Panel, press the Start button to start the calibration process.
3. Depends on the calibration mode, the calibration process will be different.
4. Once the calibration process is done, press Accept to accept the calibration result.

2.4 Select ROI

1. From the Capture Panel, press the ROI button to go to the ROI Selection Panel.
2. On the ROI Panel, press the Start button to start the ROI selection process.
3. Within the touchscreen, draw a rectangle to select the region where the object is located.
4. Once the ROI selection process is done, press Accept to accept the ROI selection result.

2.5 Measure

1. From the Capture Panel, press the Measure button to go to the Measurement Panel.
2. On the Measurement Panel, press the Start button to start the measurement process.
3. The measurement result will be displayed on the screen.

3 Panel Overview

This chapter provides information on the available panels and their functions.

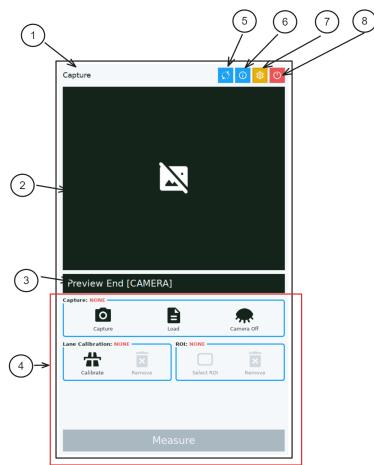
Following are the available panels:

- Capture Panel
- Automatic Lane Calibration Panel
- Manual Lane Calibration Panel
- Distance Calibration Panel
- ROI Panel
- Result Panel
- Color Calibration Panel
- Trim Data Panel
- Settings Panel

Panel Overview

3.1 Main Layout

This section provides information on the main layout of the device. Most of the available panels have the same layout.



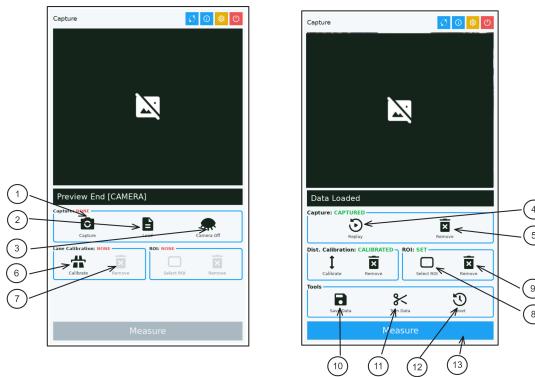
No.	Name	Function
1	Title	Show current panel
2	Image Panel	Show Image
3	Status Panel	Display status of application
4	Button Panel	Contains all buttons for processes and navigation
5	Switch Mode Button ¹	Switch between DISTANCE and LANE mode
6	Info Button	Show Info
7	Settings Button	Switch to Settings Panel
8	Exit Button	Exit the Application

¹This button only appear on Capture Panel

Panel Overview

3.2 Capture Panel

This section provides information on the Capture Panel. This Panel will handle the image capturing process. Additionally this panel also provides the ability to save and load captured data.



No	Function
1	Capture Image
2	Load Saved data
3	Toggle Camera On / Off (Preview)
4	Replay Captured Data
5	Clear/Remove Captured Data
6	Change to Calibration Panel
7	Remove Calibration Data
8	Change To Roi Panel
9	Remove Roi Data
10	Save Captured Data
11	Trim Captured Data
12	Reset Session (Remove All Data)
13	Change To Result Panel

3.2.1 Instructions

Capture Image

To capture image press the Capture button (1). The image will be captured and displayed in the panel. The image will also be saved in the program's memory.

Load Saved Data

To load saved data press the Load button (2). A file dialog will appear. Select the file you want to load. The file will be loaded and displayed in the panel. The file will also be saved in the program's memory.

Toggle Camera On / Off (Preview)

To toggle the camera on or off press the Camera button (3). The camera will be turned on or off. If the camera is on, the image will be displayed in the panel.

Replay Captured Data

To replay the captured data press the Replay button (4). Make sure that the data is loaded. The data will be replayed in the panel.

Clear/Remove Captured Data

To clear/remove the captured data press the Clear button (5). The data will be cleared/removed from the program's memory.

Change to Calibration Panel

To change to the Calibration Panel press the Calibration button (6). The Calibration Panel will be displayed. Refer [Automatic Lane Calibration Panel](#) and [Distance Calibration Panel](#) for more information.

Remove Calibration Data

To remove the calibration data press the Remove Calibration button (7). The calibration data will be removed from the program's memory. If ROI data is

Panel Overview

present, it will also be removed.

Change To Roi Panel

To change to the Roi Panel press the Roi button (8). The Roi Panel will be displayed. Refer [Roi Panel](#) for more information.

Remove Roi Data

To remove the Roi data press the Remove Roi button (9). The Roi data will be removed from the program's memory.

Save Captured Data

To save the captured data press the Save button (10). A file dialog will appear. If you press accept, the data will be saved to the file you selected.

Trim Captured Data

To trim the captured data press the Trim button (11). A dialog will appear. The TrimDataPanel will be displayed. Refer [Trim Data Panel](#) for more information.

Reset Session (Remove All Data)

To reset the session press the Reset button (12). All the data will be removed from the program's memory.

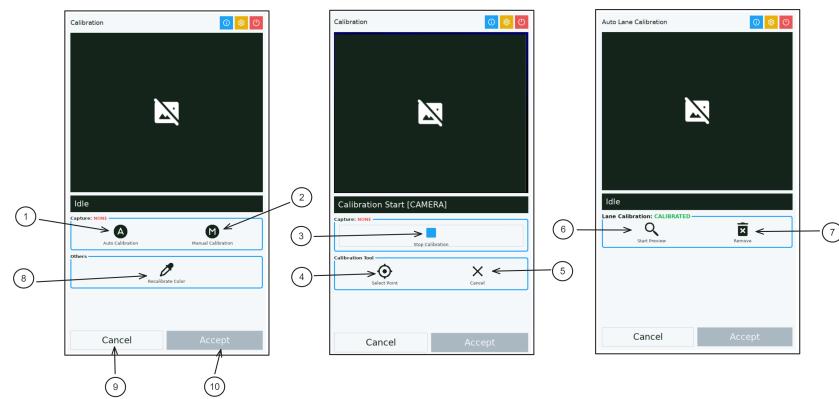
Change To Result Panel

To change to the Result Panel press the Result button (13). The Result Panel will be displayed. Refer [Result Panel](#) for more information.

Panel Overview

3.3 Automatic Lane Calibration Panel

The Calibration Panel is where you can perform automatic lane calibration. This calibration type is done with the custom mat.¹²



No.	Function
1	Start Automatic Lane Calibration
2	Change To Manual Lane Calibration Panel
3	Stop Calibration
4	Select Point
5	Cancel / Remove Calibration
6	Start Preview / Show Calibrated Data
7	Remove Calibrated Data
8	Change To Color Calibration Panel
9	Cancel Calibration
10	Accept Calibration

¹Please make sure that you adjust the object size in the Settings Panel. Go to settings panel by pressing the Settings button. Locate the **Object Width** variable and adjust it to the width of the mat (420 mm).

²To perform this calibration type, make sure the application is in **LANE MEASUREMENT MODE** by pressing the **Toggle Mode** button. Refer to [Main Layout](#) for more information.

3.3.1 Instructions

Performing Calibration

To perform automatic lane calibration, press the Start Calibration button (1). The program will start the calibration process. The program will display the image in the panel. Begin with selecting point by pressing the Select Point button (4). After clicking the point, select on the Image Panel, the position where the mat is located. The program will automatically detect line and display it in the panel. If the line is not detected, try to select another point. If the line is detected, press the Accept Calibration button (10). The program will save the calibration data in the program's memory.

Start Preview / Show Calibrated Data

To start preview or show calibrated data press the Preview button (6). The program will display the image in the panel. If the calibration data is present, the program will display the calibrated data in the panel.

Change To Manual Lane Calibration Panel

To change to the Manual Lane Calibration Panel press the Manual Calibration button (2). The Manual Lane Calibration Panel will be displayed. Refer [Manual Lane Calibration Panel](#) for more information.

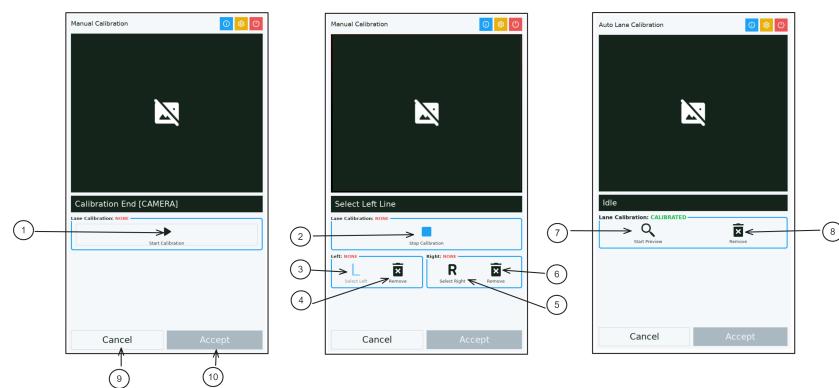
Change To Color Calibration Panel

To change to the Color Calibration Panel press the Color Calibration button (8). The Color Calibration Panel will be displayed. Refer [Color Calibration Panel](#) for more information.

Panel Overview

3.4 Manual Lane Calibration Panel

The Manual Lane Calibration Panel is where you can perform manual lane calibration.³⁴



No.	Function
1	Start Manual Lane Calibration
2	Stop Calibration
3	Select Left Line
4	Remove Selected Left Line
5	Select Right Line
6	Remove Selected Right Line
7	Start Preview / Show Calibrated Data
8	Remove Calibrated Data
9	Cancel Calibration
10	Accept Calibration

³⁴Please make sure that you adjust the object size in the Settings Panel. Go to settings panel by pressing the Settings button. Locate the **Object Width** variable and adjust it to the width of the mat (420 mm) or width of road (3500 mm)

⁴To perform this calibration type, make sure the application is in **LANE MEASUREMENT MODE** by pressing the **Toggle Mode** button. Refer to [Main Layout](#) for more information.

3.4.1 Instructions

Performing Calibration

To perform manual lane calibration, press the Start Calibration button (1). The program will start the calibration process. The program will display the image in the panel. Begin with selecting the left line by pressing the Select Left Line button (3). Line selection is done via point and drag method. Place the finger on the Image Panel where the starting point of the object is located. Adjust the position by dragging the finger. Repeat the process for the right line by pressing the Select Right Line button (5). After selecting both lines, press the Accept Calibration button (10). The program will save the calibration data in the program's memory.

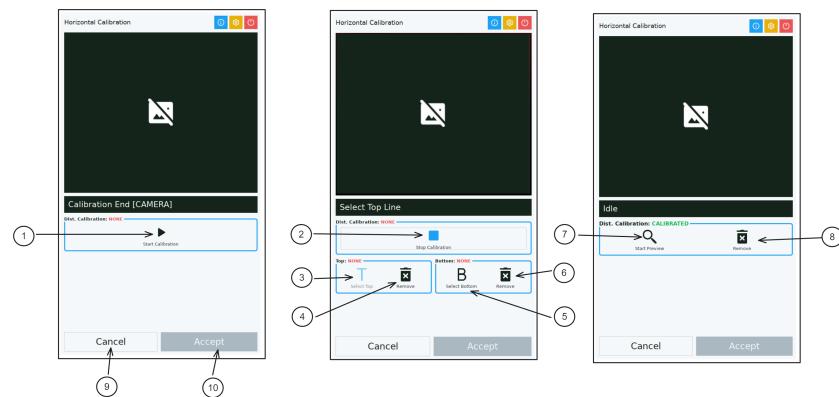
Start Preview / Show Calibrated Data

To start preview or show calibrated data press the Preview button (7). The program will display the image in the panel. If the calibration data is present, the program will display the calibrated data in the panel.

Panel Overview

3.5 Distance Calibration Panel

The Distance Calibration Panel is where you can perform distance calibration.
This calibration is done with an object of known height.⁵⁶



No.	Function
1	Start Distance Calibration
2	Stop Calibration
3	Select Top Line
4	Remove Selected Top Line
5	Select Bottom Line
6	Remove Selected Bottom Line
7	Start Preview / Show Calibrated Data
8	Remove Calibrated Data
9	Cancel Calibration
10	Accept Calibration

⁵⁵Please make sure that you adjust the object size in the Settings Panel. Go to settings panel by pressing the Settings button. Locate the **Object Height** variable and adjust it to the height of the object.

⁵⁶To perform this calibration type, make sure the application is in **DISTANCE MEASUREMENT MODE** by pressing the **Toggle Mode** button. Refer to [Main Layout](#) for more information.

3.5.1 Instructions

Performing Calibration

To perform distance calibration, press the Start Calibration button (1). The program will start the calibration process. The program will display the image in the panel. Begin with selecting the top line by pressing the Select Top Line button (3). Line selection is done via point and drag method. Place the finger on the Image Panel where the starting point of the object is located. Adjust the position by dragging the finger. Repeat the process for the bottom line by pressing the Select Bottom Line button (5). After selecting both lines, press the Accept Calibration button (10). The program will save the calibration data in the program's memory.

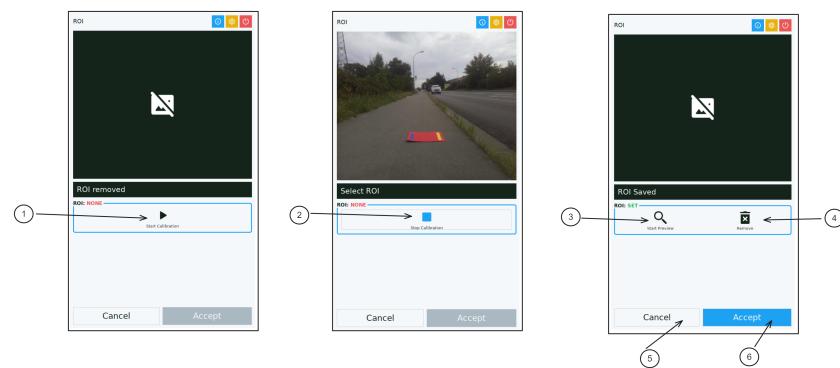
Start Preview / Show Calibrated Data

To start preview or show calibrated data press the Preview button (7). The program will display the image in the panel. If the calibration data is present, the program will display the calibrated data in the panel.

Panel Overview

3.6 Roi Panel

The Roi Panel is where you can perform region of interest (ROI) selection.



No.	Function
1	Start Roi
2	Stop Roi
3	Start Preview
4	Remove Roi
5	Cancel
6	Accept

3.6.1 Instructions

Performing ROI Selection

To perform ROI selection, press the Start Roi button (1). The program will start the ROI selection process. The program will display the image in the panel. Begin with selecting the top left point by pressing the Select Top Left Point button (3). Point selection is done via point and drag method. Place the finger on the Image Panel where the starting point of the ROI is located. Drag the finger to

Panel Overview

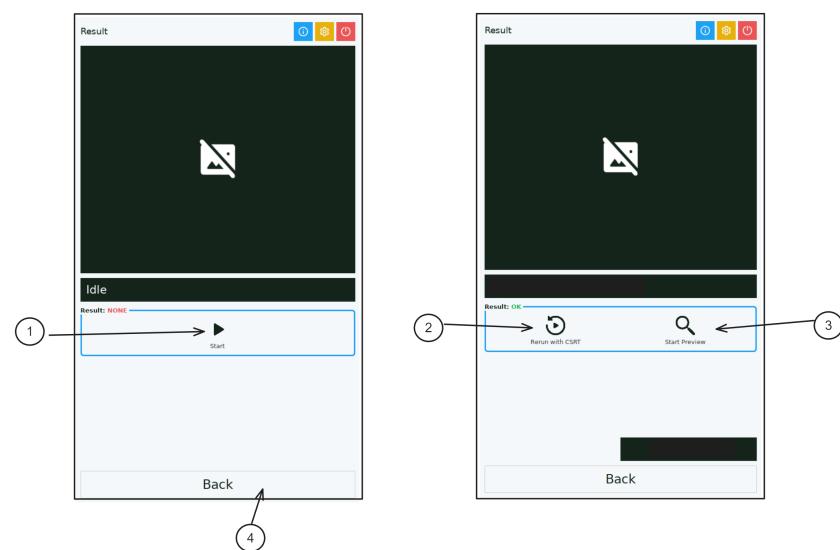
the end of the ROI. Press the Accept button (6) to accept the ROI. The program will save the ROI data in the program's memory.

Start Preview

To start preview the ROI press the Preview button (3). The program will display the image in the panel. If the ROI data is present, the program will display the ROI in the panel.

3.7 Result Panel

The Result Panel is where you can perform data processing and view the result of the measurement.



No.	Function
1	Start Process Data with Optical Flow Tracker
2	Rerun Process With CSRT Tracker
3	Preview Processed Data
4	Change To Capture Panel

3.7.1 Instructions

Start Process Data with Optical Flow Tracker

The program is set to use the Optical Flow Tracker by default. To start processing the data, press the Process Data button (1). The program will start processing the data. The program will display the processed data in the panel.

Rerun Process With CSRT Tracker

In some cases, where the Optical Flow Tracker is not accurate, you can use the CSRT Tracker. To rerun the process with CSRT Tracker, press the Rerun Process button (2). The program will start processing the data with CSRT Tracker. The program will display the processed data in the panel.

Preview Processed Data

To preview the processed data press the Preview button (3). The program will display the processed data in the panel.

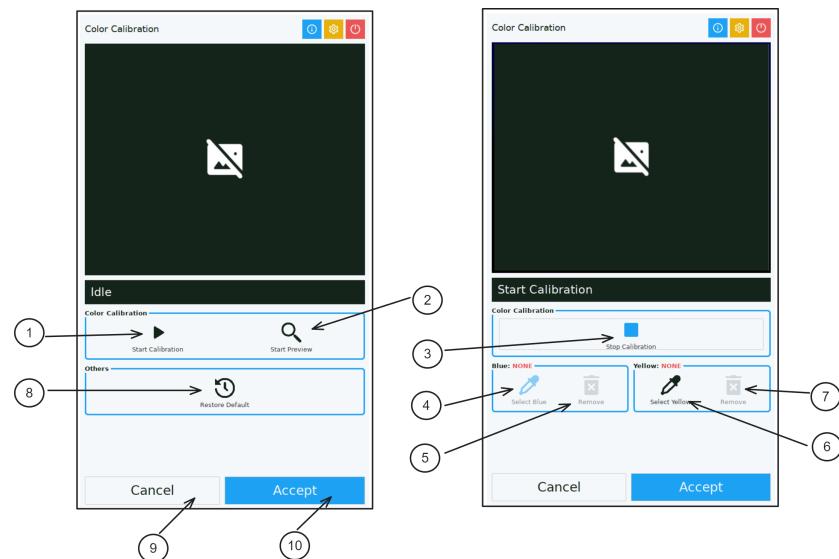
Change To Capture Panel

After processing the data, you can change to the Capture Panel by pressing the Back button (4). The Capture Panel will be displayed.

Panel Overview

3.8 Color Calibration Panel

The Color Calibration Panel is where you can perform color calibration. The purpose of this calibration is to detect the color of the mat, which is used within the [Automatic Lane Calibration Panel](#).



No.	Function
1	Start Color Calibration
2	Preview Current Calibrated Color
3	Stop Calibration
4	Select Blue Color
5	Remove Selected Blue Range
6	Select Yellow Color
7	Remove Selected Yellow Range
8	Restore Default Range
9	Cancel Calibration
10	Accept Calibration

3.8.1 Instructions

Performing Calibration

To perform color calibration, press the Start Calibration button (1). The program will start the calibration process. The program will display the image in the panel. Begin with selecting the blue color by pressing the Select Blue Color button (4). Color selection is done via point and drag method. Place the finger on the Image Panel where the blue color is located. The program will display the blue color range in the panel. If the range seems inaccurate, try to select another point. Repeat the process for the yellow color by pressing the Select Yellow Color button (6). After selecting both colors, press the Accept Calibration button (10). The program will save the calibration data in the program's memory.

Preview Current Calibrated Color

To preview the current calibrated color press the Preview button (2). The program will display the image in the panel. If the calibration data is present, the program will display the calibrated data in the panel.

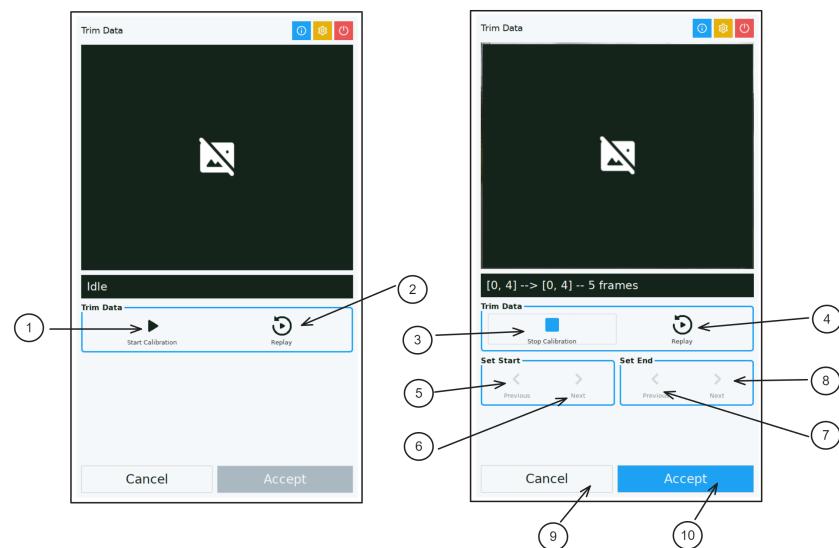
Restore Default Range

To restore the default range press the Restore Default Range button (8). The program will restore the default range.

Panel Overview

3.9 Trim Data Panel

The Trim Data Panel is where you can trim the captured data. In cases where the captured data is too long, you can trim the data to a certain length.⁷⁸



3.9.1 Instructions

Performing Trim

To perform trim, press the Start Trim button (1). The program will start the trim process. The program will display the image in the panel. Begin with selecting the start position by pressing the Decrement Start Position button (5)

⁷⁸There is a limit to the minimum length of data (5 frames). Buttons will be disabled if the minimum length is reached.

⁸By performing the trim, the program will automatically remove the calibration data and ROI data. This is done to avoid any errors.

Panel Overview

No.	Function
1	Start Trim Data
2	Replay Current Data
3	Stop Trim Data
4	Replay Current Data
5	Decrement Start Position
6	Increment Start Position
7	Decrement End Position
8	Increment End Position
9	Cancel
10	Accept

or Increment Start Position button (6). The program will display the current start position in the panel. Repeat the process for the end position by pressing the Decrement End Position button (7) or Increment End Position button (8). After selecting both positions, press the Accept button (10). The program will save the trimmed data in the program's memory.

Replay Current Data / Preview

To replay the current data or preview the trimmed data press the Replay button (2). The program will replay the data in the panel.

3.10 Settings Panel

The Settings Panel is where you can adjust the settings of the application.

3.10.1 Camera

Camera ID

The Camera ID is the ID of the camera that will be used by the application. The default value is **0**. If you have multiple cameras, you can change the value to the ID of the camera you want to use.⁹

Camera Width

The Camera Width is the width of the camera. The default value is **1280**. If you have a camera with a different width, you can change the value to the width of the camera.⁹

Camera Height

The Camera Height is the height of the camera. The default value is **960**. If you have a camera with a different height, you can change the value to the height of the camera.⁹

Camera FPS

The Camera FPS is the FPS of the camera. The default value is **10**. If you want to increase or decrease the FPS of the camera, you can change the value to the FPS of the camera.

⁹Changing the value of this variable is disabled. However, the value can be manipulated directly by accessing the config file or through source code.

3.10.2 Sensor

Sensor Width

The Sensor Width is the width of the sensor. The default value is **3.68** mm. If you have a sensor with a different width, you can change the value to the width of the sensor.¹⁰

Sensor Focal Length

The Sensor Focal Length is the focal length of the sensor. The default value is **3.04** mm. If you have a sensor with a different focal length, you can change the value to the focal length of the sensor.¹⁰

3.10.3 Capture

Max Frame

The number of frames that will be captured. The default value is **10**. If you want to increase or decrease the number of frames, you can change the value to the number of frames.

Debug - Show Image

This variable is used to show image during capturing process. There seems to be a bug where when showing images during capturing will cause inaccuracy in timing of the capturing process. The default value is **false**. If you want to show the image during capturing process, you can change the value to true.

Debug - Save Data

This variable is used to save data during capturing process. The default value is **false**. If the variable is enabled, the data will always be automatically saved for each capturing process.

¹⁰Variable is in mm.

3.10.4 Model

Maximum Thread Pool

The maximum number of threads that will be used by the application. The default value is **2**. If you want to increase or decrease the number of threads, you can change the value to the number of threads.

3.10.5 Optical Flow

Refer to OpenCV documentation for more information.

Max Corners

The maximum number of corners that will be detected by the Optical Flow Tracker. The default value is **1000**. If you want to increase or decrease the number of corners, you can change the value to the number of corners.

Quality Level

The quality level of the corners that will be detected by the Optical Flow Tracker. The default value is **0.05**. If you want to increase or decrease the quality level, you can change the value to the quality level.

Minimum Distance

The minimum distance between corners that will be detected by the Optical Flow Tracker. The default value is **7**. If you want to increase or decrease the minimum distance, you can change the value to the minimum distance.

Block Size

The block size of the corners that will be detected by the Optical Flow Tracker. The default value is **3**. If you want to increase or decrease the block size, you can change the value to the block size.

Use Harris Detector

This variable is used to enable or disable the Harris Detector. The default value is **false**. If you want to enable the Harris Detector, you can change the value to true.

K

The K value of the Harris Detector. The default value is **0.04**. If you want to increase or decrease the K value, you can change the value to the K value.

Minimum Point Distance

The minimum distance between points that will be detected by the Optical Flow Tracker. Any value below this value will be ignored. The default value is **0.2**. If you want to increase or decrease the minimum distance, you can change the value to the minimum distance.

Threshold

The minimum distance between points that will be detected by the Optical Flow Tracker. Any value below this value will be ignored. The default value is **0.2**. If you want to increase or decrease the minimum distance, you can change the value to the minimum distance.

3.10.6 Measurement

Object Width

The width of the object that will be used for measurement. The default value is **3500 mm**. If you have an object with a different width, you can change the value to the width of the object.¹¹

¹¹This variable is used together with the **DISTANCE MEASUREMENT MODE**. Refer to [Automatic Lane Calibration Panel](#) and [Manual Lane Calibration Panel](#) for more information.

Object Height

The height of the object that will be used for measurement. The default value is **2000 mm**. If you have an object with a different height, you can change the value to the height of the object.¹²

3.10.7 Preview

Preview Width

The width of the preview image. The default value is **640**. If you want to increase or decrease the width of the preview image, you can change the value to the width of the preview image.¹³

Preview Height

The height of the preview image. The default value is **480**. If you want to increase or decrease the height of the preview image, you can change the value to the height of the preview image.¹³

3.10.8 RANSAC

Threshold

The minimum distance to consider a point as an inlier. The default value is **6.0**. If you want to increase or decrease the threshold, you can change the value to the threshold.

Minimum Point

The minimum number of data within the datasets to enable line searching. Default value is **50**. If you want to increase or decrease the minimum point, you

¹²This variable is used together with the **DISTANCE MEASUREMENT MODE**. Refer to [Distance Calibration Panel](#) for more information.

¹³This variable is used to display image within Image Panel. Lowering the value will increase the performance of the application.

Panel Overview

can change the value to the minimum point.

Maximum Iteration

The maximum number of iteration to find the best line. Default value is **500**. If you want to increase or decrease the maximum iteration, you can change the value to the maximum iteration.

3.10.9 Blue HSV

Hue

The hue value of the blue color. The default value is **130** for Upper and **100** for Lower.

Saturation

The saturation value of the blue color. The default value is **255** for Upper and **100** for Lower.

Value

The value of the blue color. The default value is **255** for Upper and **100** for Lower.

3.10.10 Yellow HSV

Hue

The hue value of the yellow color. The default value is **35** for Upper and **20** for Lower.

Saturation

The saturation value of the yellow color. The default value is **255** for Upper and **100** for Lower.

Value

The value of the yellow color. The default value is **255** for Upper and **100** for Lower.

3.10.11 Threads

Variable to enable automatic process starting when switching panel.

Auto Calibration

This variable is used to enable or disable automatic calibration when switching to the Calibration Panel. The default value is **false**. If you want to enable automatic calibration, you can change the value to true.¹⁴

Auto Manual Calibration

When enabled, the program will automatically start manual calibration when switching to the Manual Lane Calibration Panel. The default value is **false**. If you want to enable automatic manual calibration, you can change the value to true.

Auto ROI

When enabled, the program will automatically start ROI selection when switching to the ROI Panel. The default value is **false**. If you want to enable automatic ROI selection, you can change the value to true.

Auto Result

When enabled, the program will automatically start processing data when switching to the Result Panel. The default value is **false**. If you want to enable automatic processing data, you can change the value to true.

¹⁴This variable is unused.

A.9 Developer Manual

Also available online at [Google Drive](#).



SpeedCameraPi

Developer Manuals

v1.3.0

last updated: October 24, 2023

Contents

I Application Compiling Guide	1
1 Acquiring the Source Code	2
1.1 Downloading the Source Code	2
1.2 Cloning the GitHub Repository	3
2 Installing Required Dependencies	4
2.1 Preparing the system	4
2.2 Preparing the system	5
2.2.1 Explanation of script	5
2.3 Installing OpenCV	6
2.4 Explanation of script	6
2.5 Installing wxWidget	8
2.5.1 Explanation of script	9
2.5.2 Installing libcamera	10
2.5.3 Explanation of script	11
2.5.4 Clean up	12
3 Compiling the Application	13
4 Utility Scripts	14
4.1 Hotspot	14
4.2 Startup	15
4.3 Touchscreen	15
II Setting up the Development Environment	17
5 Introduction	19

CONTENTS

6 Cross-compilation	20
6.1 Steps to setup Cross-Compilation	20
6.2 Fixing OpenCV linker problem	22
6.3 Installing Visual Studio Code	23
6.4 Accessing WSL from Visual Studio Code	23
6.5 Installing Required Extensions	24
6.6 Opening SpeedCameraPi project	24
6.7 Configuring C/C++ Extension	25
6.8 Building the project	26
6.9 Debugging the project	27
6.9.1 Preparations	27
6.9.2 Debugging	28
6.9.3 Known Issues	28
6.10 Extra Features	29
6.10.1 VNC Viewer	29
6.10.2 VSCode Extensions	33
7 Native Compilation	34
7.1 Installing required Extensions	34
7.2 Opening SpeedCameraPi project	34
7.3 Configuring C/C++ extension	36
7.4 Building the project	36
7.5 Debugging	37
III Getting Started with Development	38
8 Introduction	39
9 Add new Panel	40
9.1 Define a new PanelID Enum	40
9.2 Create a new Controller class	41
9.2.1 Define a new Controller class	41
9.2.2 Add method to ControllerFactory	42
9.3 Create a new Panel class	43

CONTENTS

9.3.1	Define a new Panel class	43
9.3.2	Create ButtonPanel	44
9.3.3	Implement the Panel class	44
9.3.4	Add method to PanelFactory	45
9.4	Define object in MainFrame	46
10	Custom Event	47
10.1	DataEvent	47
10.1.1	Example Classes	49
10.2	EmptyEvent	49
10.3	Bind Event	50
10.4	Submit Event	50
11	Thread	51
11.1	Define unique ThreadID	51
11.2	Create a new Thread class	51
11.3	Add method to ThreadController	53
12	Task for ThreadPool	54
12.1	Define unique TaskType	54
12.2	Create a new Task class	54

Part I

Application Compiling Guide

1 Acquiring the Source Code

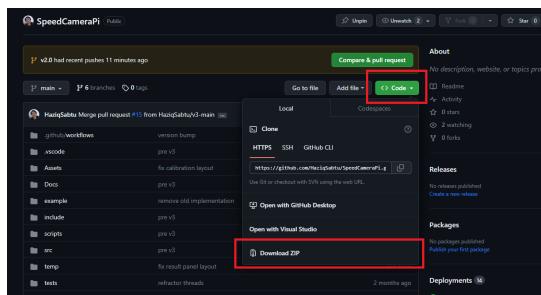
There are two ways to acquire the source code for the project. They are:

1. Downloading the source code from the GitHub repository
2. Cloning the GitHub repository

1.1 Downloading the Source Code

Alternatively, you can download the source code as a zip file from the GitHub repository page and extract it to your home directory.

- Go to the GitHub repository page:
1 <https://github.com/HaziqSabtu/SpeedCameraPi.git>
- Click on the green button labeled **Code** and select **Download ZIP**.



- Extract the downloaded zip file to your **home** directory.

Acquiring the Source Code

1.2 Cloning the GitHub Repository

Run the following commands to clone the repository:

```
1 cd ~  
2 git clone https://github.com/HaziqSabtu/SpeedCameraPi.git
```

2 Installing Required Dependencies

In this chapter, we will be installing the required dependencies for the project. The dependencies are:

1. OpenCV v4.5.5
2. wxWidgets
3. libcamera v0.0.4

2.1 Preparing the system

Before installing the required dependencies, following steps are required to be done:

- Navigate to scripts directory:

```
1 cd ~/SpeedCameraPi/scripts/Installation
```

- Allow script to be executed:

```
1 sudo chmod +x DI_Scripts1.sh
```

- Run script:

```
1 sudo ./DI_Scripts1.sh
```

Installing Required Dependencies

2.2 Preparing the system

Before installing the required dependencies, following steps are required to be done:

- Navigate to scripts directory:

```
1 cd ~/SpeedCameraPi/scripts/Installation
```

- Allow script to be executed:

```
1 sudo chmod +x DI_Scripts1.sh
```

- Run script:

```
1 sudo ./DI_Scripts1.sh
```

2.2.1 Explanation of script

- Update the package list and upgrade all packages to their latest versions:

```
1 sudo apt-get update && sudo apt-get upgrade
```

- Increase swap size to 4096MByte to prevent the system from running out of memory during the compilation process:

```
1 NEW_SWAPSIZE=4096
2 sudo sed -i "s/CONF_SWAPSIZE=.*$/CONF_SWAPSIZE=$NEW_SWAPSIZE/
   "/etc/dphys-swapfile
3 sudo sed -i "s/CONF_MAXSWAP=.*$/CONF_MAXSWAP=$NEW_SWAPSIZE/"
   /sbin/dphys-swapfile
4 sudo systemctl restart dphys-swapfile
```

Installing Required Dependencies

2.3 Installing OpenCV

OpenCV is a powerful computer vision library that provides a vast range of image and video processing capabilities. This project uses OpenCV to perform image processing tasks such as detecting and tracking objects in the video stream.

Steps to install OpenCV on Raspberry Pi 4 running Debian: [\[source\]](#)

- Navigate to scripts directory:

```
1 cd ~/SpeedCameraPi/scripts/Installation
```

- Allow script to be executed:

```
1 sudo chmod +x DI_Scripts2.sh
```

- Run script:

```
1 sudo ./DI_Scripts2.sh
```

2.4 Explanation of script

- Update the package list and upgrade all packages to their latest versions:

```
1 sudo apt-get update && sudo apt-get upgrade
```

- Install the required dependencies for building OpenCV:

```
1 sudo apt-get install -y build-essential cmake git unzip pkg-config
2 sudo apt-get install -y libjpeg-dev libtiff-dev libpng-dev
3 sudo apt-get install -y libavcodec-dev libavformat-dev
   libswscale-dev
4 sudo apt-get install -y libgtk2.0-dev libcanberra-gtk*
   libgtk-3-dev
5 sudo apt-get install -y libgstreamer1.0-dev gstreamer1.0-
   gtk3
```

Installing Required Dependencies

```
6   sudo apt-get install -y libgstreamer-plugins-base1.0-dev
7     gstreamer1.0-gl
8   sudo apt-get install -y libxvidcore-dev libx264-dev
9   sudo apt-get install -y python3-dev python3-numpy python3-
10    pip
11   sudo apt-get install -y libtbb2 libtbb-dev libdc1394-22-dev
12   sudo apt-get install -y libv4l-dev v4l-utils
13   sudo apt-get install -y libopenblas-dev libatlas-base-dev
14     libblas-dev
15   sudo apt-get install -y liblapack-dev gfortran lib hdf5-dev
16   sudo apt-get install -y libprotobuf-dev libgoogle-glog-dev
17     libgflags-dev
18   sudo apt-get install -y protobuf-compiler
```

- Download OpenCV version 4.5.5 from the repository:

```
1 cd ~
2 sudo rm -rf opencv*
3 wget -O opencv.zip https://github.com/opencv/opencv/archive/
4   /4.5.5.zip
5 wget -O opencv_contrib.zip https://github.com/
6   opencv/opencv_contrib/archive/4.5.5.zip
7 unzip opencv.zip
8 unzip opencv_contrib.zip
9 mv opencv-4.5.5 opencv
10 mv opencv_contrib-4.5.5 opencv_contrib
11 rm opencv.zip
12 rm opencv_contrib.zip
```

- Create a build directory and navigate to it:

```
1 cd ~/opencv
2 mkdir build
3 cd build
```

- Configure the build process by running the following command:

```
1 cmake -D CMAKE_BUILD_TYPE=RELEASE \
2 -D CMAKE_INSTALL_PREFIX=/usr/local \
3 -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
4 -D ENABLE_NEON=ON \
```

Installing Required Dependencies

```
5   -D WITH_OPENMP=ON \
6   -D WITH_OPENCL=OFF \
7   -D BUILD_TIFF=ON \
8   -D WITH_FFMPEG=ON \
9   -D WITH_TBB=ON \
10  -D BUILD_TBB=ON \
11  -D WITH_GSTREAMER=ON \
12  -D BUILD_TESTS=OFF \
13  -D WITH_EIGEN=OFF \
14  -D WITH_V4L=ON \
15  -D WITH_LIBV4L=ON \
16  -D WITH_VTK=OFF \
17  -D WITH_QT=ON \
18  -D OPENCV_ENABLE_NONFREE=ON \
19  -D INSTALL_C_EXAMPLES=OFF \
20  -D INSTALL_PYTHON_EXAMPLES=OFF \
21  -D OPENCV_FORCE_LIBATOMIC_COMPILER_CHECK=1 \
22  -D PYTHON3_PACKAGES_PATH=/usr/lib/python3/dist-packages \
23  -D OPENCV_GENERATE_PKGCONFIG=ON \
24  -D BUILD_EXAMPLES=OFF ..
```

- Compile and install the library:

```
1 make -j1
2 sudo make install
3 sudo ldconfig
```

2.5 Installing wxWidget

wxWidgets is a cross-platform GUI library that provides a set of C++ classes for creating graphical user interfaces. This project uses wxWidgets to create the graphical user interface for the application.

Steps to install wxWidget on Raspberry Pi 4 running Debian: [\[source\]](#)

- Navigate to scripts directory:

```
1 cd ~/SpeedCameraPi/scripts/Installation
```

Installing Required Dependencies

- Allow script to be executed:

```
1 sudo chmod +x DI_Scripts3.sh
```

- Run script:

```
1 sudo ./DI_Scripts3.sh
```

2.5.1 Explanation of script

- Update the package list and upgrade all packages to their latest versions:

```
1 sudo apt-get update && sudo apt-get upgrade
```

- Install the required dependencies for building wxWidgets:

```
1 sudo apt-get install libgtk-3-dev build-essential  
checkinstall
```

- Download the latest stable version of wxWidgets from the official website:

```
1 wget https://github.com/wxWidgets/wxWidgets/releases/  
download/v3.2.2.1/wxWidgets-3.2.2.1.tar.bz2
```

- Extract the downloaded archive and navigate to the extracted directory:

```
1 tar -xvf wxWidgets-3.2.2.1.tar.bz2
```

- Change directory to the extracted directory:

```
1 cd wxWidgets-3.2.2.1
```

- Create a build directory and navigate to it:

```
1 mkdir gtk-build  
2 cd gtk-build
```

- Configure the build process by running the following command:

```
1 .../configure --disable-shared --enable-unicode --disable-  
debug
```

Installing Required Dependencies

- Compile and install the library:

```
1 make  
2 sudo make install
```

2.5.2 Installing libcamera

Libcamera is an open-source software stack designed to support complex camera configurations on Raspberry Pi and other embedded devices. It provides a unified interface for multiple camera modules, allowing developers to efficiently harness their full potential in various applications, from computer vision projects to advanced photography.

As of now, libcamera is still under development and is still in alpha stage. However, it is already being used by the Raspberry Pi Foundation to support the Raspberry Pi High-Quality Camera Module. It is recommended to use libcamera in the future as it provides a more flexible and powerful interface for controlling the camera module.

For this project, we will be using v0.0.4. As of now, a higher version is available, however, as it is still in the alpha stage, a lot of changes are still being made. When a more stable release is available, we will update this guide.

Steps to install Libcamera v0.0.4 on Raspberry Pi 4 running Debian:

- Navigate to scripts directory:

```
1 cd ~/SpeedCameraPi/scripts/Installation
```

- Allow script to be executed:

```
1 sudo chmod +x DI_Scripts4.sh
```

- Run script:

```
1 sudo ./DI_Scripts4.sh
```

Installing Required Dependencies

2.5.3 Explanation of script

- Update the package list and upgrade all packages to their latest versions:

```
1 sudo apt-get update && sudo apt-get upgrade
```

- Install the required dependencies for building libcamera:

```
1 sudo apt install -y libboost-dev
2 sudo apt install -y libgnutls28-dev openssl libtiff5-dev
   pybind11-dev
3 sudo apt install -y qtbase5-dev libqt5core5a libqt5gui5
   libqt5widgets5
4 sudo apt install -y meson
5 sudo apt install -y cmake
6 sudo pip3 install pyyaml ply
7 sudo pip3 install --upgrade meson
8 sudo apt install -y libglib2.0-dev libgstreamer-plugins-
   base1.0-dev
```

- Download the repository and checkout to v0.0.4:

```
1 cd ~
2 git clone https://github.com/raspberrypi/libcamera.git
3 cd libcamera
4 git checkout v0.0.4
```

- Configure and build the library:

```
1 meson build --buildtype=release -Dpipelines=raspberrypi -
   Dipas=raspberrypi -Dv4l2=true -Dgstreamer=enabled -Dtest=
   false -Dlc-compliance=disabled -Dcam=disabled -Dqcam=
   enabled -Ddocumentation=disabled -Dpycamera=enabled
2 ninja -C build -j2    # use -j 2 on Raspberry Pi 3 or earlier
   devices
3 sudo ninja -C build install
```

Installing Required Dependencies

2.5.4 Clean up

After installing the required dependencies, you may want to clean up the unnecessary files to free up some disk space. To do so, run the following commands:

- Navigate to scripts directory:

```
1 cd ~/SpeedCameraPi/scripts/Installation
```

- Allow script to be executed:

```
1 sudo chmod +x DI_Scripts5.sh
```

- Run script:

```
1 sudo ./DI_Scripts5.sh
```

3 Compiling the Application

To build the application, follow these steps:

- Create a build directory and navigate to it:

```
1 cd ~/SpeedCameraPi  
2 mkdir build  
3 cd build
```

- Configure the build process by running the following command:

```
1 cmake -B . -H ..
```

- Compile the application:

```
1 make -j3
```

- After finished compiling, the executable file is located in the build directory.

```
1 ./build
```

- To run the application, simply double-click the executable file or run the following command:

```
1 ./SpeedCameraPi
```

4 Utility Scripts

This section will provide some useful scripts. These scripts are optional and are not required to run the application. However, they are useful for managing the application. The scripts are:

- **Hotspot.sh** - This script will create a hotspot on the Raspberry Pi. This is useful if you want to connect to the Raspberry Pi without a router.
- **Startup.py** - This script will run the application on startup. This is useful if you want the application to run automatically when the Raspberry Pi boots up.
- **Touchscreen.sh** - This script will fix the Touchscreen rotation on the Raspberry Pi.

4.1 Hotspot

This script will configure the Raspberry Pi to enable the hotspot feature whenever it is not connected to the local network. This will allow you to connect to the Raspberry Pi directly when you are not at home.

- Navigate to scripts directory:

```
1 cd ~/SpeedCameraPi/scripts/Installation
```

- Allow script to be executed:

```
1 sudo chmod +x Hotspot.sh
```

- Run script:

```
1 sudo ./Hotspot.sh
```

4.2 Startup

This script will configure the Raspberry Pi to automatically start the Speed-CameraPi application whenever it is booted up. This will allow you to use the Raspberry Pi as a standalone device without the need to manually start the application.

- Navigate to scripts directory:

```
1 cd ~/SpeedCameraPi/scripts/Installation
```

- Allow script to be executed:

```
1 sudo chmod +x Startup.py
```

- Run script:

```
1 sudo ./Startup.py
```

4.3 Touchscreen

This script will fix the touchscreen rotation issue. This will allow you to use the touchscreen in portrait mode.

- Navigate to scripts directory:

```
1 cd ~/SpeedCameraPi/scripts/Installation
```

- Allow script to be executed:

```
1 sudo chmod +x Touchscreen.sh
```

- Run script:

Utility Scripts

```
1  sudo ./Touchscreen.sh
```

Part II

Setting up the Development Environment

For future developers, this section will help you properly set up your development environment. In general, the development is done with Visual Studio Code with cross-compilation to Raspberry Pi 4. A guide to setting up the development environment is provided below. Alternatively, a native development setup will also be provided.

5 Introduction

C++ is a programming language that is designed to be compiled, meaning that it is generally translated into machine language that can be understood directly by the system, making the generated program highly efficient. To compile C++ code, you need a set of tools known as the development toolchain, whose core are a compiler and its linker. A compiler is a program that converts your C++ code into machine code, while a linker is a program that combines the machine code with other libraries and resources to create an executable file. There are different compilers and linkers available for C++, such as Visual Studio, Clang, mingw, etc. Some of them are integrated with IDEs (Integrated Development Environments) that provide features such as syntax highlighting, debugging, code completion, etc.

To cross-compile C++ code, you need to use a cross-compiler that can generate executable code for a different platform than the one you are running on. For example, if you want to compile C++ code for ARM architecture on a Windows PC, you need to use a cross-compiler like `arm-none-linux-gnueabi-gcc`. Depending on the tool you are using, you may need to configure some settings to enable cross compilation. For example, if you are using Visual Studio Code, you can edit the Compiler path and Compiler arguments settings in the C/C++ extension to specify the cross-compiler and the target triplet. You can also use other tools like CMake or Makefile to automate the cross-compilation process.

6 Cross-compilation

Cross-compilation is the process of compiling code on one platform (host) to run on another platform (target). In this case, we will be compiling the code on a Windows 11 machine running Windows Subsystem for Linux (WSL) to run on Raspberry Pi 4 running Raspberry Pi OS (Bullseye 32-bit).

This method of development is preferred as it allows for faster compilation time and easier debugging, which allows for better development experience.

Upon further reading, there are two important terms to take note of:

- **Host** - The machine that you are developing on. In this case, it is the Windows 11 machine.
- **Target** - The machine that you are developing for. In this case, it is the Raspberry Pi 4.

This method has been tested numerous times and is proven to work. However, it is not guaranteed to work on all machines. To reproduce the development environment, following Build number of Windows 11 and WSL is required:

- Windows 11 Pro Version 22H2 OS Build 22621.2428
- WSL 2 with Ubuntu 22.04 LTS

6.1 Steps to setup Cross-Compilation

For this section, it is assumed that you have already installed WSL 2 with Ubuntu 22.04 LTS. If not, please refer to the [official documentation](#) to install WSL 2 with

Cross-compilation

Ubuntu 22.04 LTS. Most of the process are taken from [this guide](#).

- To begin with, it is important that required dependencies are installed on **Target**. Refer to the Section 2 to install the required dependencies.
- Next, we need to prepare the **Target** to allow ssh connection from the host. To do so, run the following command on **Host**:

```
1 cd ~/SpeedCameraPi/scripts/crosscompile  
2 sudo chmod +x setup_ssh.py  
3 ./setup_ssh.py hostname username
```

Type yes when prompted. When successful, you should now be able to ssh into the target from the host with the following command:

```
1 ssh RPi0
```

- Next, we need to prepare the **target** for cross-compilation. To do so, run the following command on **Host**:

```
1 cd ~/SpeedCameraPi/scripts/crosscompile  
2 sudo chmod +x Target_setup.py  
3 ./Target_setup.py credential password
```

- Next, we need to prepare the **host** for cross-compilation. To do so, run the following command on **Host**:

```
1 cd ~/SpeedCameraPi/scripts/crosscompile  
2 sudo chmod +x Host_setup.py  
3 sudo ./Host_setup.py
```

- Lastly, now we need to sync the files structure of **Target** with **Host** on the staging directory (sysroot). To do so run following command:

```
1 cd ~/SpeedCameraPi/scripts/crosscompile  
2 sudo chmod +x Host_postSetup.py  
3 ./Host_postSetup.py
```

- Now, you should be able to perform cross-compilation. However, as of now, there is a linker problem with OpenCV libraries. To fix this, please refer to the next section.

6.2 Fixing OpenCV linker problem

Now, we will once again compile OpenCV. There seems to be a problem using the already compiled library of OpenCV as done previously on the **Target**. Some *opencv_contrib* libraries cannot be found during runtime. Possible linker error? Still investigating. However, the current fix is to cross compile the library on the **Host** and transfer the compiled libs to the **Target**.

- To do so, follow these steps:

```
1 cd ~/SpeedCameraPi/scripts/crosscompile  
2 sudo chmod +x Target_cvinstall.py  
3 ./Target_cvinstall.py credential
```

- Now, we will compile OpenCV on **host** machine:

```
1 cd ~/SpeedCameraPi/scripts/crosscompile  
2 sudo chmod +x Host_cvcompile.py  
3 sudo ./Host_cvcompile.py
```

In some cases, during compilation of OpenCV, it will produce error *jconfig.h not found*. To fix this, run the following command:

```
1 cd ~/SpeedCameraPi/scripts/crosscompile  
2 sudo chmod +x fix_jconfig.py  
3 ./fix_jconfig.py
```

- Now, when the library is compiled, we can transfer the compiled library to the **target** machine. To do so, run the following command on **Host**:

```
1 cd ~/SpeedCameraPi/scripts/crosscompile  
2 sudo chmod +x Host_cvinstall.py  
3 ./Host_cvinstall.py
```

Now, the machine is ready for cross-compilation. Now we will explore steps to install Visual Studio Code.

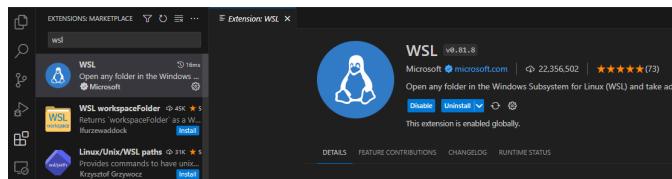
6.3 Installing Visual Studio Code

Visual Studio Code is a free source-code editor made by Microsoft for Windows, Linux, and macOS. It is a very popular code editor among developers. This section will guide you through the process of installing Visual Studio Code for cross-compilation.

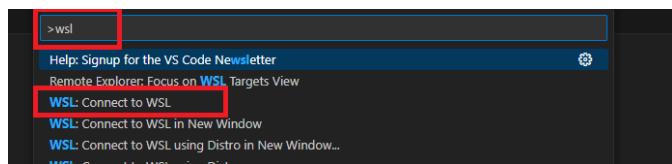
If you are running Linux on WSL, it is recommended to install Visual Studio Code directly on Windows. Download the installer from [here](#) and install it on Windows.

6.4 Accessing WSL from Visual Studio Code

First, we need to install the WSL Extension for Visual Studio Code. To do so, open the Extensions pane (Ctrl+Shift+X) and search for WSL. Click on the Install button to install the extension.

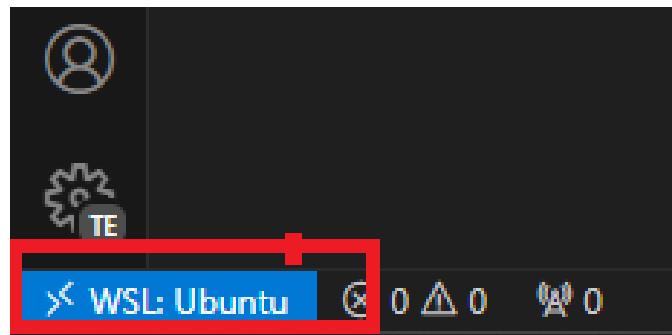


To access WSL from Visual Studio Code, open the Command Palette (Ctrl+Shift+P) and type `WSL: Connect to wsl`. This will open a new Visual Studio Code window with WSL as the default shell. You can also open a folder in WSL by right-clicking on the folder in the File Explorer and selecting Open in WSL.



Cross-compilation

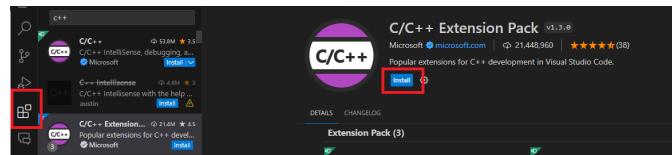
You should see that you are currently in the WSL shell. You can also check the bottom left corner of the window to see if you are in the WSL shell.



6.5 Installing Required Extensions

To install extensions, open Visual Studio Code and press Ctrl+Shift+X to open the Extensions pane. Search for the following extensions and install them:

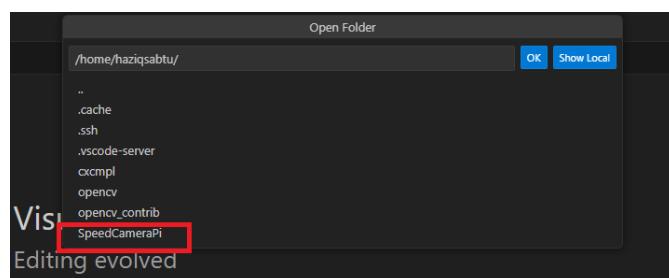
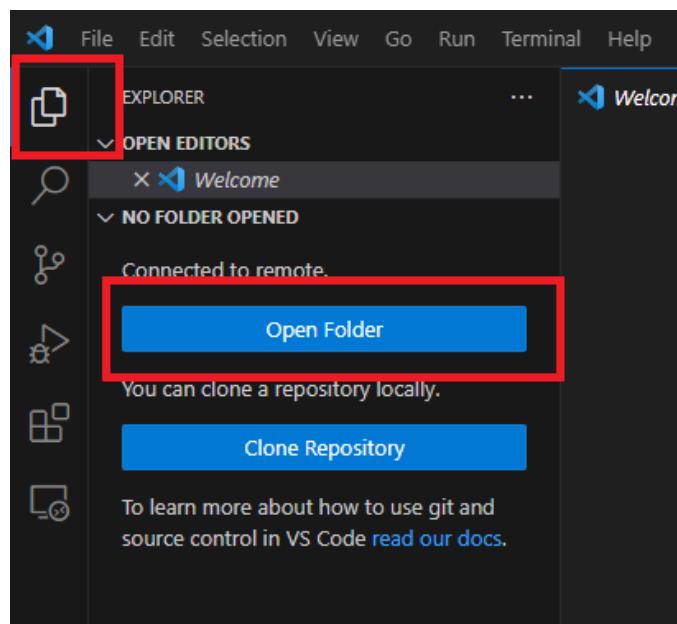
- C/C++ Extension Pack



6.6 Opening SpeedCameraPi project

To open the SpeedCameraPi project, open the Explorer pane (Ctrl+Shift+E) and click on the Open Folder button. Navigate to the SpeedCameraPi folder and click on the OK button. You should now see the SpeedCameraPi project in the Explorer pane.

Cross-compilation



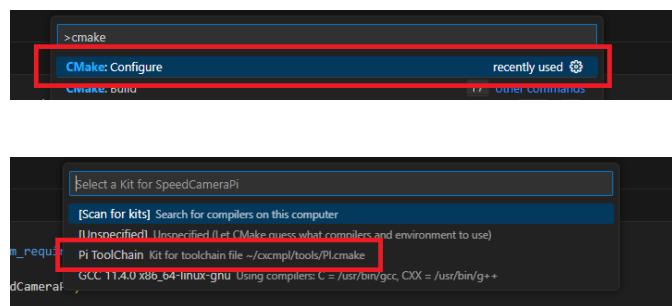
6.7 Configuring C/C++ Extension

To configure the C/C++ extension, open the Command Palette (Ctrl+Shift+P) and type `CMake:Configure` and select the `Pi Toolchain` option. This will

Cross-compilation

create a build folder in the project directory and generate the CMake cache.

If the Pi Toolchain option is not available, you may need to check if the cross-compilation setup is done correctly. Additionally, in some cases the path may be different. If so, you may need to edit the `.vscode/cmake-kits.json` file to change the path to the cross compiler.

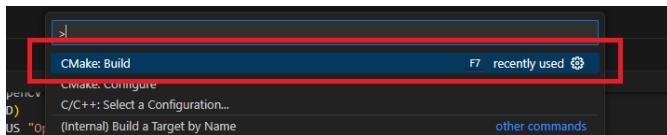


If the configuration process is successful, you should see the following message in the terminal:

```
[cmake] -- Check if compiler accepts -pthread - yes
[cmake] -- Found Threads: TRUE
[cmake] -- Threads found
[cmake] -- libcamera library found:
[cmake] --   version: 0.0.4
[cmake] --   libraries: /home/haziqsabtu/cxcmpl/sysroot/usr/local/lib/arm-linux-gnueabihf/
[cmake] --   include path: /home/haziqsabtu/cxcmpl/sysroot/usr/local/include/libcamera
[cmake] --   Library found in directory: /home/haziqsabtu/cxcmpl/sysroot/usr/local/lib/arm-
[cmake] --   libcamera include dir: /home/haziqsabtu/cxcmpl/sysroot/usr/local/include/libcamera
[cmake] --   path to libcamera: /home/haziqsabtu/cxcmpl/sysroot/usr/local/lib/arm-linux-gnueabi
[cmake] --   path to libcamera-base: /home/haziqsabtu/cxcmpl/sysroot/usr/local/lib/arm-linux-gnueabi
[cmake] --   Configuring done
[cmake] -- Generating done
[cmake] -- Build files have been written to: /home/haziqsabtu/SpeedCameraPi/build
```

6.8 Building the project

To build the project, open the Command Palette (Ctrl+Shift+P) and type `CMake:Build`. This will build the project and generate the executable file in the `build` folder.



Within the `CMakeLists.txt` file, the executable file is by default configured to be automatically copied to the **Target** machine. You can disable this by editing out the command. However, it is recommended to keep it as it is. The build application is located at `/Target/SpeedCameraPi`.

6.9 Debugging the project

6.9.1 Preparations

Debugging is crucial when developing a project. These steps need to be done at least once. To perform debugging, we first need to configure the `launch.json` script. To do so, navigate to the script folder:

```
1 cd ~/SpeedCameraPi/scripts/debug
```

Next, run the following command:

```
1 sudo chmod +x launch.py
```

Lastly, run the script with the following parameters:

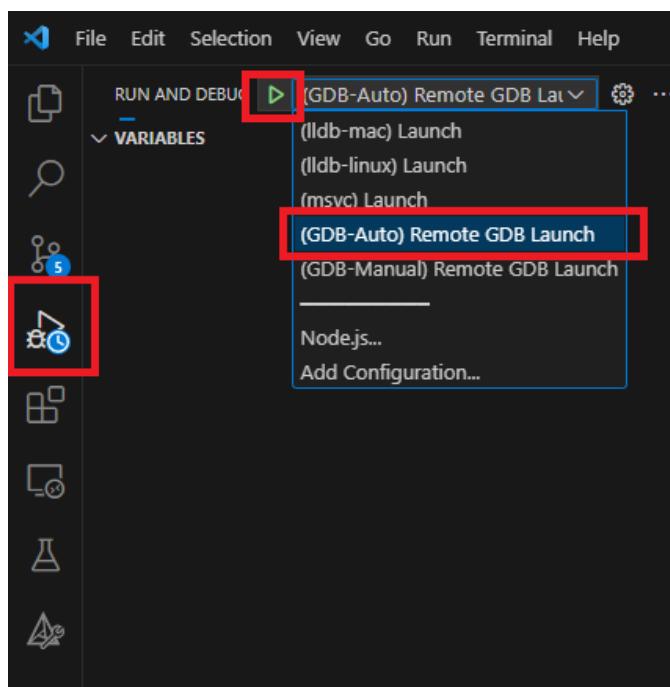
```
1 sudo ./launch.py ipaddress
```

This will generate the `launch.json` file in the `.vscode` folder. Now we need to install `gdbserver` on the **Target** machine. To do so, run the following command on **Host**:

```
1 ssh RPi0
2 sudo apt-get install gdbserver
```

6.9.2 Debugging

To debug the project, open the Debugger Tab (`Ctrl + Shift + D`) and select the profile (GDB-Auto) Remote GDB Launch and click on the Run button. This will start the `gdbserver` on the **Target** machine and connect to it. You should now be able to debug the project.



6.9.3 Known Issues

In some cases, certain libraries cannot be found. To fix this, run the following command on **Host**:

- `libncurses.so`

```
1 sudo apt install libncurses5
```

- `libpython2.7.so`

```
1 sudo apt install libpython2.7
```

6.10 Extra Features

In this section, we will explore some extra features that can be used to improve the development experience.

6.10.1 VNC Viewer

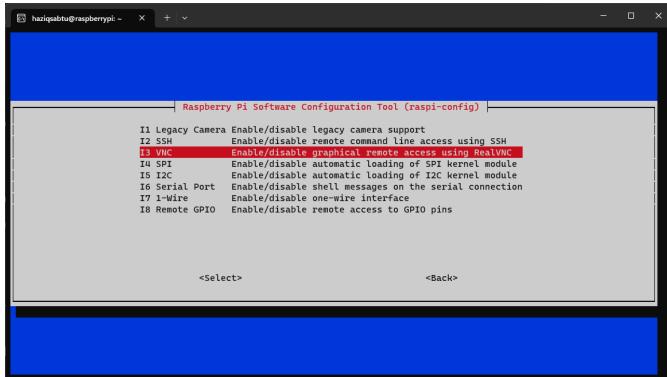
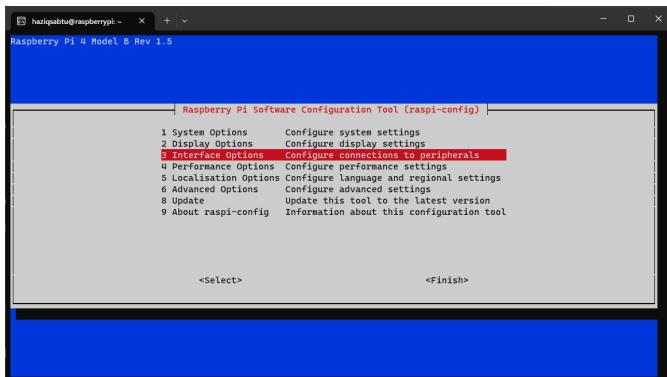
VNC Viewer is a remote desktop application that allows you to remotely control a computer. It is useful for accessing the **Target** machine remotely. By properly integrating the VNC, your Raspberry Pi can be used without a monitor, keyboard, or mouse.

To download VNC Viewer, go to [this link](#) and download the installer for your operating system. Once installed, you may need to create an account to use the application. Once done you may need to prepare the **Target** machine to allow VNC connection. To do so, run the following command on **Host**:

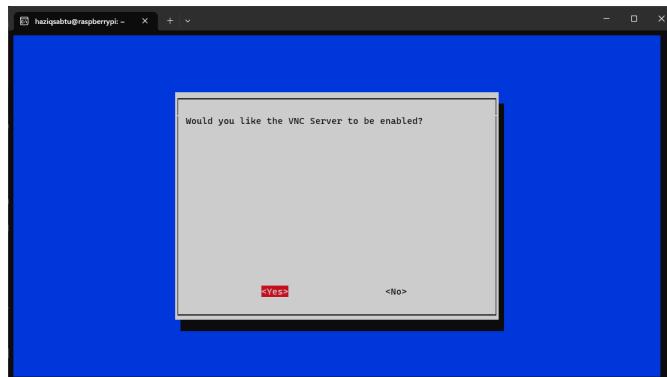
```
1 ssh RPi0
2 sudo raspi-config
```

Navigate to **Interface Options** and enable **VNC**.

Cross-compilation



Cross-compilation



Now, you should be able to access the **Target** machine via VNC Viewer. It is recommended to reboot the **Target** machine after enabling VNC.

To access the **Target** machine via VNC Viewer, you need to know the IP address of the **Target** machine. To do so, run the following command on **Host**:

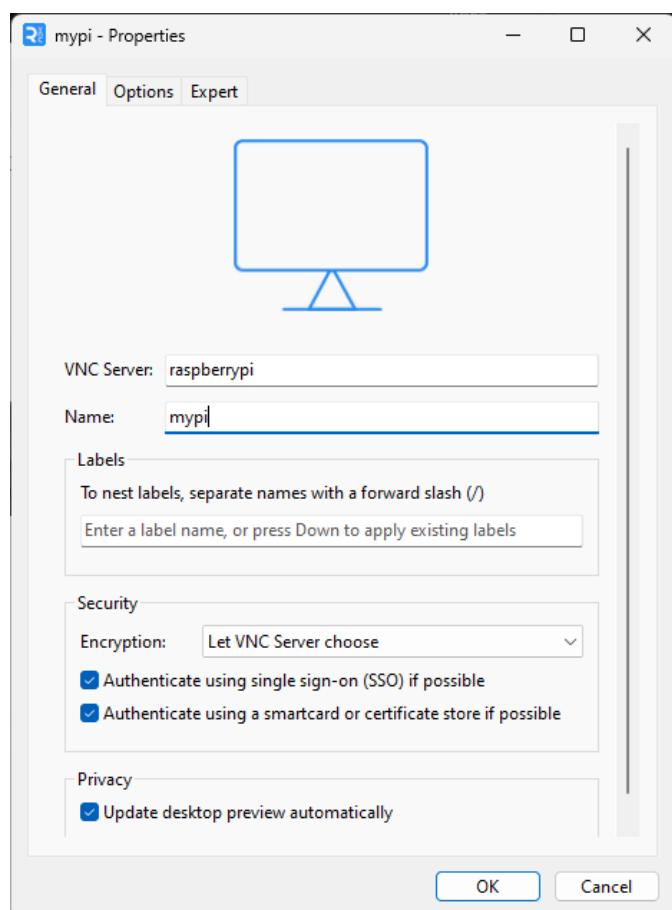
```
1 hostname -I
```

Alternatively, it is defaulted to `raspberrypi`.

Now, you can access the **Target** machine via VNC Viewer. To do so, open VNC Viewer and create a new connection (`Ctrl + N`) and set the following parameters:

- **VNC Server** - IP address of the **Target** machine
- **Name** - Name of the connection, can be anything

Cross-compilation



Now, you should be able to access the **Target** machine via VNC Viewer. You may need to enter the password of the **Target** machine.

6.10.2 VSCode Extensions

Following are some useful extensions that can be used to improve the development experience:

- [Clang-Format](#): Clang-Format is a tool that formats C/C++/Obj-C code according to a set of style options, similar to the way that clang-format formats code.
- [Github Copilot](#): GitHub Copilot is an AI pair programmer that helps you write code faster and with less work.
- [Todo Tree](#): Todo Tree is a task manager that helps you manage your TODOs.

7 Native Compilation

For those who prefer to develop natively on the Raspberry Pi, this section will guide you through the process of setting up the development environment.

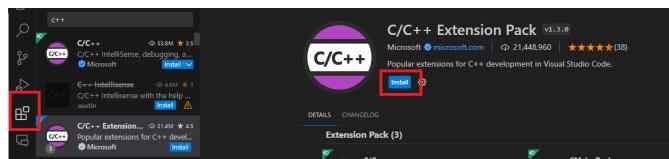
For native compilation, the steps are much simpler. Simply install the required dependencies on the **Target** machine and you are good to go. Refer to [this section](#) for more information.

The installation of VSCode on Raspberry Pi is done with [this](#) instruction.

7.1 Installing required Extensions

To install extensions, open Visual Studio Code and press **Ctrl+Shift+X** to open the Extensions pane. Search for the following extensions and install them:

- C/C++ Extension Pack

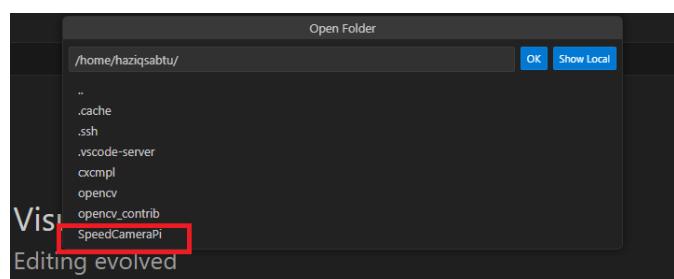
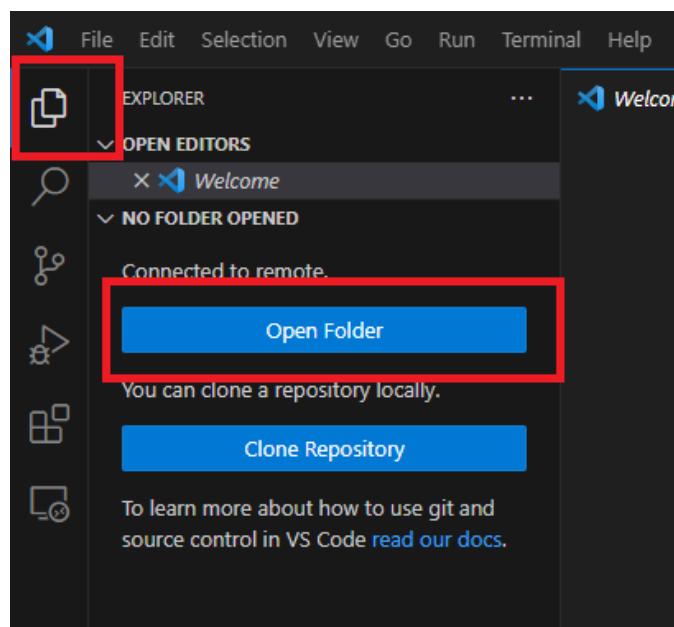


7.2 Opening SpeedCameraPi project

To open the SpeedCameraPi project, open the Explorer pane (**Ctrl+Shift+E**) and click on the Open Folder button. Navigate to the SpeedCameraPi folder

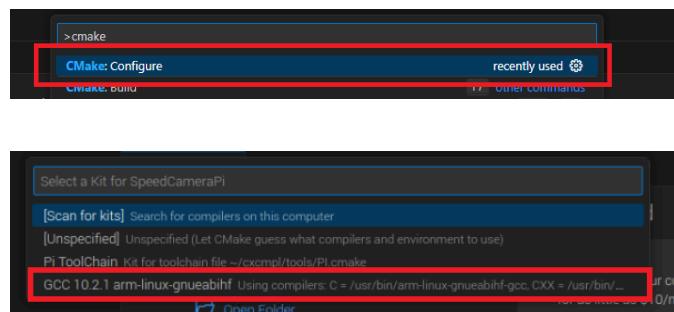
Native Compilation

and click on the OK button. You should now see the SpeedCameraPi project in the Explorer pane.



7.3 Configuring C/C++ extension

To configure the C/C++ extension, open the Command Palette (**Ctrl+Shift+P**) and type **CMake : Configure** and select the default compiler. This will create a **build** folder in the project directory and generate the CMake cache.



If the configuration process is successful, you should see the following message in the terminal:

```
[PROBLEMS] [OUTPUT] DEBUG CONSOLE TERMINAL PORTS
[cmake] -- Check if compiler accepts -pthread - yes
[cmake] -- Found Threads: TRUE
[cmake] -- Threads found
[cmake] -- libcamera library found:
[cmake] --   version: 0.0.4
[cmake] --   libraries: /home/haziqsabtu/cxcmpl/sysroot/usr/local/lib/arm-linux-gnueabihf/libso
[cmake] --   include path: /home/haziqsabtu/cxcmpl/sysroot/usr/local/include/libcamera
[cmake] --   Library found in directory: /home/haziqsabtu/cxcmpl/sysroot/usr/local/lib/arm-linux-gnueabihf/libcamera
[cmake] -- libcamera include dir: /home/haziqsabtu/cxcmpl/sysroot/usr/local/include/libcamera
[cmake] -- path to libcamera: /home/haziqsabtu/cxcmpl/sysroot/usr/local/lib/arm-linux-gnueabihf/libcamera
[cmake] -- path to libcamera-base: /home/haziqsabtu/cxcmpl/sysroot/usr/local/lib/arm-linux-gnueabihf/libcamera-base
[cmake] -- Configuring done
[cmake] -- Generating done
[cmake] -- Build files have been written to: /home/haziqsabtu/SpeedCameraPi/build
```

7.4 Building the project

To build the project, open the Command Palette (**Ctrl+Shift+P**) and type **CMake:Build**. This will build the project and generate the executable file in

Native Compilation

the build folder.

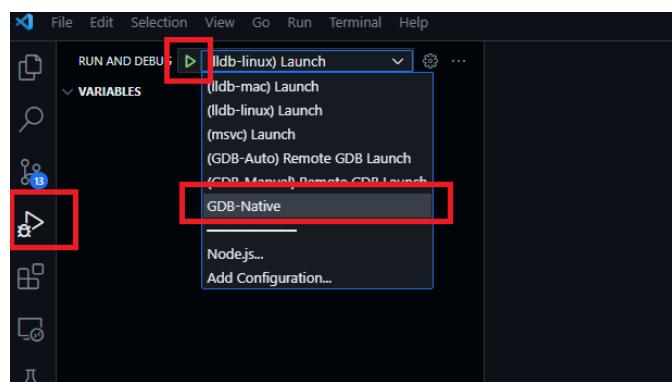


If the UI is unresponsive, it is highly likely that the building process is using all of the resources available. Either wait until the process completes, or reduce the number of parallel build jobs. To do so, open the Command Palette (`Ctrl+Shift+P`) and type `Settings: Open Settings (UI)`. Navigate to `CMake: Parallel Jobs` and set the value to 1.



7.5 Debugging

To debug the project, open the Debugger Tab (`Ctrl + Shift + D`) and select the profile `GDB - Native` and click on the Run button. This will compile the project and start the debugger. You should now be able to debug the project.



Part III

Getting Started with Development

8 Introduction

This section will guide you through the process of developing the application. It is assumed that you have already set up the development environment. If not, please refer to the previous section. This section will cover the following topics:

- **Adding a new Panel** - This section will guide you through the process of adding a new panel to the application.
- **Custom Event** - This section will guide you through the process of defining, triggering, and handling custom events.
- **Defining Process Threads** - This section will guide you through the process of defining process threads.
- **Creating Tasks** - This section will guide you through the process of creating tasks for ThreadPool.

9 Add new Panel

This chapter will provide description on how to add new panel to the application. To add a new Panel to the Application, following steps are required:

1. Define a new PanelID Enum
2. Create a new Controller class
3. Create a new Panel class
4. Define object in MainFrame

9.1 Define a new PanelID Enum

The first step is to register a unique enum for the Panel. This enum will be used to identify the Panel in the application. To add a new Panel enum, modify the PanelID enum in the `include/Model/Enum.hpp` file. Following example will provide a simple example of a PanelID enum:

Listing 9.1: PanelID enum example

```
1 enum PanelID {  
2     // ... other enum  
3     PANEL_INFO,  
4     PANEL_MY_PANEL, // Add new PanelID enum here  
5 };
```

9.2 Create a new Controller class

9.2.1 Define a new Controller class

To create a new controller class, you first need to decide either the Panel will require a touch input or not. If touch input is required you can create a custom controller class that extends the BaseControllerWithTouch class. Otherwise you can create a custom controller class that extends the BaseController class. Following example will provide a simple example of a Controller class that extends the BaseControllerWithTouch class:

Listing 9.2: MyController class example

```
1 class MyController : public BaseControllerWithTouch {
2     public:
3         MyController(ResourcePtr shared);
4         ~MyController();
5
6     private:
7         static const PanelID currentPanelID = PanelID::PANEL_MY_PANEL; // Add new PanelID enum here
8
9     private:
10        void throwIfAnyThreadIsRunning() override;
11
12        void killAllThreads(wxEvtHandler *parent) override;
13
14        void leftDownHandler(wxEvtHandler *parent, cv::Point point)
15            override;
16        void leftMoveHandler(wxEvtHandler *parent, cv::Point point)
17            override;
18        void leftUpHandler(wxEvtHandler *parent, cv::Point point)
19            override;
20    };
```

Make sure that you implemented the pure virtual functions from the BaseControllerWithTouch class. The pure virtual functions are:

Add new Panel

- void throwIfAnyThreadIsRunning() override;
- void killAllThreads(wxEvtHandler *parent) override;
- void leftDownHandler(wxEvtHandler *parent, cv::Point point) override;
- void leftMoveHandler(wxEvtHandler *parent, cv::Point point) override;
- void leftUpHandler(wxEvtHandler *parent, cv::Point point) override;

See the source code for `LaneCalibrationController` for example.

To make the code more readable, it is advised to define the controller shared pointer to a shorter name. Following example will provide a simple example of defining a shorter name for the Controller shared pointer:

Listing 9.3: Shorter name for Controller shared pointer

```
1 #define MCPtr std::shared_ptr<MyController>
```

9.2.2 Add method to ControllerFactory

The `ControllerFactory` is used to connect the Controller with the Model component. Following example can be used:

Listing 9.4: ControllerFactory method example

```
1 class ControllerFactory {
2     public:
3         ControllerFactory(wxWindow *parent);
4         ~ControllerFactory();
5
6         ResourcePtr getSharedModel();
7
8         // ... other methods
9         MyPtr createMyController();
10
```

Add new Panel

```
11     private:  
12         ResourcePtr shared;  
13     };
```

Listing 9.5: Implementation of createMyController method

```
1 MyPtr ControllerFactory::createMyController() {  
2     return std::make_shared<MyController>(shared);  
3 }
```

9.3 Create a new Panel class

9.3.1 Define a new Panel class

To create a new Panel class, you need to create a new class that extends either the BasePanel or the BasePanelWithTouch class. The BasePanel class has already implemented the basic functionality of a Panel, while the BasePanelWithTouch class has already implemented the basic functionality of a Panel with touch support. The BasePanelWithTouch class is recommended to be used if the Panel will be used in a touch screen device. Alternatively you can also create a whole new class that extends the wxPanel class from wxWidgets. Following example will provide a simple example of a Panel class that extends the BasePanelWithTouch class:

Listing 9.6: MyPanel class example

```
1 class MyPanel : public BasePanelWithTouch {  
2     public:  
3         MyPanel(wxWindow *parent, wxWindowID id, MyPtr controller);  
4         ~MyPanel();  
5  
6     private:  
7         const PanelID panel_id = PANEL_MY_PANEL;  
8  
9         MyPtr controller;  
10  
11     DECLARE_EVENT_TABLE()
```

Add new Panel

```
12 };
```

Make sure that if you are using the BasePanelWithTouch class, you are also using the BaseControllerWithTouch class. Otherwise you will get a compilation error.

9.3.2 Create ButtonPanel

Now create a ButtonPanel for the Panel. The ButtonPanel is a panel that contains buttons to perform certain actions. To create a ButtonPanel, you need to create a new class that extends the BaseButtonPanel class. Following example will provide a simple example of a ButtonPanel class that extends the BaseButtonPanel class:

Listing 9.7: CustomButtonPanel class example

```
1 class CustomButtonPanel : public BaseButtonPanel {
2     public:
3         CustomButtonPanel(wxWindow *parent, wxWindowID id);
4
5         void update(const AppState &state) override;
6
7         // define all buttons here
8
9         DECLARE_EVENT_TABLE();
10    };
```

Make sure that the `update()` method is implemented. The `update()` method will be called by the Panel to update the state of the buttons.

9.3.3 Implement the Panel class

Now that you have created the ButtonPanel, you can now implement the Panel class. Following example can be used as a reference to implement the Panel class:

Listing 9.8: Implementation of MyPanel class

```
1 MyPanel::MyPanel(wxWindow *parent, wxWindowID id,
2                               MyPtr controller)
3     : BasePanelWithTouch(parent, id, controller), controller(
4         controller) {
5
5     button_panel =
6         new CustomButtonPanel(this, wxID_ANY);
7     title_panel = new TitlePanel(this, panel_id);
8
9     size();
10 }
```

Make sure the `size()` method is called at the end of the constructor. The `size()` method will set the sizer of the Panel.

9.3.4 Add method to PanelFactory

In this step, you need to modify the PanelFactory class. Following example can be used:

Listing 9.9: Create Panel method in PanelFactory

```
1 wxPanel *createPanel(wxWindow *parent, PanelID panelID) {
2     // ... other if case
3
4     if (panelID == PANEL_MY_PANEL) {
5         return createMyPanel(parent);
6     }
7 }
8
9 MyPanel *createMyPanel(wxWindow *parent) {
10     return new MyPanel(parent, wxID_ANY, controllerFactory->
11                         createMyController());
11 }
```

9.4 Define object in MainFrame

The last step is to define the Panel in the MainFrame class. Following example can be used:

Listing 9.10: MainFrame class example

```
1 class MainFrame : public wxFrame {
2     // ... other methods
3
4     private:
5     // ... other methods
6     MyPanel *myPanel;
7     // ... other methods
8 };
9
10 MainFrame::MainFrame() : wxFrame(NULL, wxID_ANY, Data::AppName) {
11
12     // ... other methods
13
14     // ... other registerPanel
15     registerPanel(PANEL_MY_PANEL);
16
17     showFirstPanel();
18 }
```

Now you have successfully added a new Panel to the application.

10 Custom Event

Events are used as a response mechanism for the View component. There are two types of events in this application:

- DataEvent
- EmptyEvent

10.1 DataEvent

DataEvent is an event that contains data. The data can be any type of data. To create a new DataEvent, you need to create a new class that extends the wxCommandEvent class. Following example will provide a simple example of a DataEvent class:

Listing 10.1: CustomEvent class example

```
1 class CustomEvent;
2 wxDECLARE_EVENT(c_CUSTOM_EVENT, CustomEvent);
3
4 enum CUSTOM_EVENT_TYPE {
5     CUSTOM_EVENT_TYPE_1 = 1,
6     CUSTOM_EVENT_TYPE_2,
7     CUSTOM_EVENT_TYPE_3,
8 };
9
10 class CustomEvent : public wxCommandEvent {
11 public:
12     CustomEvent(wxEventType type, int id = 1);
13
14     CustomEvent(const CustomEvent &event);
```

Custom Event

```
15     virtual wxEvent *Clone() const override;
16
17     // Define set and get method for the data here
18     void setData(const std::string &data);
19     std::string getData() const;
20
21     private:
22         std::string data;
23     };
24
25     // this part is important
26     #typedef void (wxEvtHandler::*CustomFunction)(CustomEvent &);
27     #define CustomHandler(func)
28
29             \\\n
30             wxEVENT_HANDLER_CAST(CustomFunction, func)
31     #define EVT_CUSTOM(id, func)
32
33             \\\n
34             wx__DECLARE_EVT1(c_CUSTOM_EVENT, id, CustomHandler(func))
```

Listing 10.2: CustomEvent continued

```
1 wxDEFINE_EVENT(c_CUSTOM_EVENT, CustomEvent);
2
3 CustomEvent::CustomEvent(wxEventType type, int id)
4     : wxCommandEvent(type, id) {
5 }
6
7 CustomEvent::CustomEvent(const CustomEvent &event)
8     : wxCommandEvent(event) {
9     this->setData(event.getData());
10 }
11
12 wxEvent *UpdatePreviewEvent::Clone() const {
13     return new CustomEvent(*this);
14 }
15
16 void CustomEvent::setData(const std::string &data) {
17     this->data = data;
18 }
19
20 std::string CustomEvent::getData() const {
```

Custom Event

```
21     return data;  
22 }
```

10.1.1 Example Classes

The following classes can be referred to as examples:

- UpdatePreviewEvent
- UpdateStateEvent
- ErrorEvent

10.2 EmptyEvent

EmptyEvent is an event that does not contain any data. It is normally used to signal the View component to perform certain action. To create a new EmptyEvent, you need to create a new class that extends the wxCommandEvent class. Following example will provide a simple example of an EmptyEvent class:

Listing 10.3: EmptyEvent class example

```
1 wxDECLARE_EVENT(c_CUSTOM_EVENT, wxCommandEvent);  
2  
3 enum CUSTOM_EVENT_TYPE {  
4     CUSTOM_EVENT_TYPE_1 = 1,  
5     CUSTOM_EVENT_TYPE_2,  
6     CUSTOM_EVENT_TYPE_3,  
7 };
```

Listing 10.4: EmptyEvent continued

```
1 wxDEFINE_EVENT(c_CUSTOM_EVENT, wxCommandEvent);
```

10.3 Bind Event

To bind an event to a method, you need to use the following syntax in the Panel class:

Listing 10.5: Binding Event in Panel class

```
1 BEGIN_EVENT_TABLE(BasePanel, wxPanel)
2     // ... other event
3
4     // For EmptyEvent
5     EVT_COMMAND(wxID_ANY, c_CUSTOM_EVENT, BasePanel::OnCustomEvent)
6
7     // For DataEvent
8     EVT_CUSTOM(c_CUSTOM_EVENT, BasePanel::OnCustomEvent)
9 END_EVENT_TABLE()
```

Make sure the function to handle the event is implemented.

10.4 Submit Event

To submit an event, you need to use the following syntax:

Listing 10.6: Submitting Event

```
1 // For EmptyEvent
2 wxCommandEvent event(c_CUSTOM_EVENT, CUSTOM_EVENT_TYPE_1);
3 wxPostEvent(this, event);
4
5 // For DataEvent
6 CustomEvent event(c_CUSTOM_EVENT, CUSTOM_EVENT_TYPE_1);
7 event.setData("data");
8 wxPostEvent(this, event);
```

11 Thread

Thread is used to perform a long-running task.

11.1 Define unique ThreadID

To define a unique ThreadID, you need to modify the ThreadID enum in the include/Thread/Thread_ID.hpp file. Following example will provide a simple example of a ThreadID enum:

Listing 11.1: ThreadID enum example

```
1 enum ThreadID {
2     // ... other enum
3     THREAD_MY_THREAD, // Add new ThreadID enum here
4 };
```

11.2 Create a new Thread class

Now you need to create a new class that extends the BaseThread class. Following example will provide a simple example of a Thread class:

Listing 11.2: CustomThread class example

```
1 class CustomThread : public BaseThread {
2     public:
3         CustomThread(wxEvtHandler *parent, DataPtr data);
4         ~CustomThread();
5
6         ThreadID getID() const override;
```

Thread

```
7
8     protected:
9         virtual ExitCode Entry() override;
10
11    private:
12        // ... other methods
13        const ThreadID thread_id = ThreadID::THREAD_MY_THREAD; // Add new
14        ThreadID enum here
15    };
16
17 CustomThread::CustomThread(wxEvtHandler *parent, DataPtr data)
18     : BaseThread(parent, data) {
19 }
20
21 CustomThread::~CustomThread() {
22 }
23
24 ThreadID CustomThread::getID() const {
25     return thread_id;
26 }
27
28 wxThread::ExitCode CustomThread::Entry() {
29     // ... other code, do process
30     return (wxThread::ExitCode)0;
31 }
```

Additionally, following classes can be inherited to add more functionality:

Class	Description
PreviewableThread	Enable the thread to send image to ImagePanel
ImageSizeDataThread	Add variable <code>imageSize</code> , which is the size of the captured image
ImageDataThread	Add variable <code>image</code> , which is the camera
CameraAccessor	Enable camera access

11.3 Add method to ThreadController

In this step, you need to modify the ThreadController class. Following example can be used:

Listing 11.3: ThreadController class example

```
1 class ThreadController {
2     public:
3         ThreadController(wxEvtHandler *parent, DataPtr data);
4         ~ThreadController();
5
6         // ... other methods
7
8         virtual void startCustomThread(wxEvtHandler *parent, PanelID
9             panelID);
10
11        void endCustomThread();
12
13        CustomThread *getCustomThread();
14
15    private:
16        // ... other variables
17        CustomThread *custom_thread;
18 }
```

12 Task for ThreadPool

ThreadPool enables parallel processing, which can be used to speed up the application. To add a new Task to the ThreadPool, following steps are required:

12.1 Define unique TaskType

To define a unique TaskType, you need to modify the TaskType enum in the include/Thread/Task/Task.hpp file. Following example will provide a simple example of a TaskType enum:

Listing 12.1: TaskType enum example

```
1 enum TaskType {
2     // ... other enum
3     TASK_MY_TASK, // Add new TaskType enum here
4 };
```

12.2 Create a new Task class

Now you need to create a new class that extends the Task class. Following example will provide a simple example of a Task class:

Listing 12.2: CustomTask class example

```
1 class CustomTask : public Task {
2     public:
3         CustomTask(wxEvtHandler *parent, DataPtr data);
4         void Execute() override;
```

Task for ThreadPool

```
5     private:
6         // ... other methods
7         const std::string currentName = "CustomTask";
8         const TaskType currentType = TaskType::TASK_MY_TASK; // Add new
9             TaskType enum here
10    };
11
12 CustomTask::CustomTask(wxEvtHandler *parent, DataPtr data)
13     : Task(parent, data) {
14     property = TaskProperty(currentType);
15     name = currentName;
16 }
17
18 void CustomTask::Execute() {
19     // ... other code, do process
20 }
```