A Low-Cost, Off-Grid Camera-Based System for Water Level Monitoring

Development and Field Validation of a Proof-of-Concept for Remote River Monitoring using Edge Computing

AEC (Bjorn) Rens

Delft University of Technology

TU Delft

A Low-Cost, Off-Grid Camera-Based System for Water Level Monitoring

Development and Field Validation of a Proof-of-Concept for Remote River Monitoring using Edge Computing

by

AEC (Bjorn) Rens

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday June 18, 2025 at 14:00 AM.

Student number:	5654254	
Project duration:	November 1, 2024 – June 18, 2025	
Thesis committee:	Prof. dr. ir. N.C. van de Giesen,	TU Delft, Water Resource Engineering
	Dr. R. Taormina,	TU Delft, Sanitary Engineering
	Dr. ir. R.W. Hut,	TU Delft, Water Resource Engineering
	Dr. H.C. Winsemius,	Rainbow Sensing

TU Delft

Preface

This thesis marks the end of an exciting and rewarding journey through my Master's degree in Environmental Engineering at the Delft University of Technology. The core inspiration for this project stemmed not only from my academic curiosity but also from my internship experience in Nepal—a resource-constrained environment where local engineers were incredibly driven yet faced significant barriers due to high equipment costs. Witnessing their challenges firsthand underscored the importance of creating accessible and sustainable monitoring technologies.

Developing this low-cost, off-grid, camera-based water-level monitoring system has significantly tested and expanded my technical skills, ranging from computer vision and embedded systems to practical field experimentation. Additionally, I developed essential hardware skills, such as soldering and hands-on tinkering with electronics.

I would like to sincerely thank my supervisor, Prof. dr. ir. Nick van de Giesen, not only for his guidance throughout the research but also for his support and good company during fieldwork. A huge thanks to Dr. ir. Rolf Hut for his creative insights and hands-on help with hardware tinkering, and to Dr. Hessel Winsemius for generously providing access to his OpenRiverCam platform and sharing his extensive experience. Special thanks also go to Dr. Riccardo Taormina, whose feedback, though he was less directly involved, significantly contributed to my research.

I am deeply grateful for the unwavering support from my girlfriend Daphne, who stood by me through the ups and downs of this academic journey. To my family, thank you for continually encouraging me as I pursued my academic goals. I am also immensely appreciative of the friendly and enjoyable environment created by everyone in the "afstudeer hok". Special thanks to Mannes, Christine, Karen, and Sophie for enduring the struggles with me.

I hope this work contributes to more accessible and effective water management solutions, sparking further innovation in environmental engineering.

AEC (Bjorn) Rens
Delft, June 2025

Abstract

Accurate water-level monitoring is vital for effective river management, flood prevention, and environmental conservation efforts. The increasing frequency and severity of flooding events, driven by climate change, highlight the critical need for reliable, robust, and sustainable water-level measurement systems. Such systems are particularly necessary in remote and resource-constrained settings where conventional infrastructure is lacking or insufficient.

Traditional water-level measurement methods, including radar, ultrasonic, pressure sensors, and conventional imaging systems, encounter significant limitations. These include high acquisition and maintenance costs, susceptibility to environmental damage, vandalism risks due to conspicuous placement, and dependence on stable, continuous power sources. Consequently, there remains an unmet demand for affordable, resilient, and autonomous monitoring devices capable of functioning reliably in challenging and off-grid environments.

To address this gap, this study presents the development and validation of an innovative water-level monitoring prototype as a prove of concept. The proposed system integrates a low-cost Raspberry Pi-based imaging sensor equipped with infrared illumination to enable accurate measurements both during the day and at night. An onboard processing unit autonomously captures and analyses images in real-time using one of two distinct image-processing algorithms: the mean-difference method and the Kolmogorov–Smirnov (KS) test method. These algorithms quantify water levels by detecting pixel-intensity contrasts, thus eliminating the need for direct physical contact with the water body.

The prototype underwent field validation in a natural river environment, demonstrating robust and consistent performance. The system achieved an accuracy within ±4.6 cm at a 95% confidence interval. Notably, it exhibited stable performance under varying environmental conditions, with moderate bias differences between daytime and nighttime scenarios, and minimal sensitivity to precipitation effects. The mean-difference algorithm demonstrated superior precision, while the KS-test method offered enhanced robustness against environmental variability.

This research underscores the practical feasibility and significant potential of low-cost, autonomous camera-based systems for sustainable water-level monitoring. Such solutions can substantially improve disaster preparedness and environmental management, especially in remote or economically disadvantaged regions lacking traditional monitoring infrastructure. Future research directions include integrating renewable energy solutions such as solar power to enhance operational autonomy, exploring advanced image-processing algorithms for increased accuracy, and conducting extended validations across a broader range of hydrological and environmental scenarios.

Keywords: *Water-level monitoring; Autonomous sensors; Computer vision; Raspberry Pi; Off-grid operation; Flood management; Low-cost hardware; Open-source hardware*

Contents

Preface i

Summary ii

Nomenclature viii

1 Hardware in context 1
- 1.1 State of the art . . . 1
- 1.2 Research gap and objectives . . . 2
- 1.3 Introducing the proof of concept . . . 2

2 Hardware description 3
- 2.1 Imaging module . . . 3
- 2.2 Illumination system . . . 3
- 2.3 Onboard processing unit . . . 4
- 2.4 Data storage and communication . . . 4
- 2.5 Power supply and energy management . . . 4
- 2.6 Enclosure and mounting assembly . . . 4
- 2.7 Reference object for waterline detection . . . 5

3 Design files summary 8

4 Bill of materials summary 10

5 Build instructions 11
- 5.1 Hardware . . . 11
  - 5.1.1 Preparing power supply and PCB . . . 13
  - 5.1.2 Mounting mechanism & modular design . . . 14
  - 5.1.3 Installing camera module . . . 15
  - 5.1.4 Installing IR-illumination module . . . 16
  - 5.1.5 Device assembly . . . 17
  - 5.1.6 Reference object . . . 18
- 5.2 Software . . . 19
  - 5.2.1 System setup summary . . . 19
  - 5.2.2 Remote access and data retrieval . . . 20
- 5.3 Water level detection algorithm . . . 20
  - 5.3.1 Image pre-processing . . . 20
  - 5.3.2 Mean-difference method . . . 21
  - 5.3.3 KS-test method . . . 22

6 Operation instructions 24
- 6.1 Setup and orientation . . . 24
- 6.2 Calibration and verification . . . 25
- 6.3 Operation . . . 25

7 Validation and characterization 26
- 7.1 Experimental setup . . . 26
- 7.2 Data acquisition . . . 27
- 7.3 Results . . . 28
  - 7.3.1 Raw data and outliers . . . 28
  - 7.3.2 Diurnal patterns . . . 29
  - 7.3.3 Influence of precipitation . . . 30
  - 7.3.4 Performance statistics . . . 31

7.4 Power assessment . . . 33

8 Discussion **35**
8.1 Interpretation of field data . . . 35
8.2 Comparison to expectations & literature . . . 36
8.2.1 Comparison to traditional sensors . . . 36
8.2.2 Expected diurnal influence versus observed performance . . . 36
8.2.3 Expected influence of precipitation versus observed performance . . . 37
8.3 Limitations . . . 37

9 Conclusion **40**
9.1 Key findings . . . 40
9.2 Practical implications . . . 41

10 Future work **42**

References **43**

A Design Methodology **46**
A.1 Design Perspective . . . 46
A.2 Function Analysis . . . 47
A.3 Basic Design Cycle . . . 49
A.4 List of Requirements . . . 50
A.5 Morphological Chart . . . 52
A.6 SCAMPER Method . . . 53

B Design choices **54**
B.1 Operating system Raspberry Pi . . . 54
B.2 Reference object . . . 54
B.3 Algorithm development . . . 55
B.4 Hardware development . . . 56

C Criteria evaluation **57**
C.1 Measurement Performance . . . 57
C.2 Operational Aspects . . . 58
C.2.1 Summary of Performance . . . 59

D System Setup Manual **60**
D.1 Operating System . . . 60
D.2 First-Boot Configuration: Swapfile and Updates . . . 61
D.3 Package Installation . . . 62
D.4 Auto-Start Service . . . 63

E Remote access manual **64**
E.1 Remote access GUI mode; RealVNC . . . 64
E.2 Remote access CLI mode; SSH & PuTTY . . . 65
E.3 SFTP file transfer via WinSCP . . . 68

F Operation and calibration manual **69**
F.1 Calibration . . . 69
F.1.1 Device setup checklist . . . 70
F.1.2 Disable device checklist . . . 71
F.2 Operation . . . 72

G Development LOG **73**

List of Figures

2.1 Conceptual system architecture of the image-based water-level monitoring prototype. . 5
2.2 Hardware components used in the water-level monitoring system. For more detailed information on hardware components, see section 4.1 6
2.3 Final system assembly and reference object deployment. 7

3.1 Overview of Python file structure with descriptions. 9

5.1 Hardware schematic of the camera-based water-level monitoring system. 12
5.2 Overview of the soldered components, including a 1.2Ω, 5W resistor and MOSFET driver integrated on a 60×90mm PCB, with output leads for IR LED connection. 13
5.3 Assembly and mounting of the system enclosure. 14
5.4 Integration of the camera module into the system enclosure. 15
5.5 Integration of the IR LED module with heatsink, enclosure, and diffuser. 16
5.6 Final assembly and integration of power, processing, and connectivity components. . . 17
5.7 Construction and installation of the modular reference board for waterline detection. . . 18
5.8 Preprocessing workflow for image-based waterline detection. 20
5.9 Illustration of the circular nature of the Hue channel [47] 21
5.10 Schematic overview of the visual waterline detection process. 21
5.11 Algorithm results displaying the most probable waterline location (red dashed line) for each image channel. 22
5.12 Visual outputs for quality control and verification of the waterline detection process. . . . 22
5.13 Illustration of the Kolmogorov–Smirnov (KS) statistic.[7] 23
5.14 KS-test detection process in the HSV channel. 23

6.1 Schematic overview of the deployment setup 24

7.1 Field deployment location (50.80762°N, 5.91350°E). 26
7.2 Field deployment of the water-level monitoring system at the Kleine Geul. 27
7.3 Time series of detection error (cm) for both the mean-difference and KS-test methods across the 42-hour field deployment. 29
7.4 Detection error over time for the mean-difference method. A 24-hour sinusoidal curve (red line) highlights the diurnal bias, with a clear positive shift during daylight hours. The mean bias across the deployment was 1.57cm (orange dashed line). The green dashed line represents the zero-error reference level. Day (white) and night (grey) periods are shaded accordingly. 30
7.5 Detection error over time for the KS-test method. A sinusoidal fit (red line) is shown for comparison, but no clear diurnal pattern is observed. The average bias was 1.02 cm (orange dashed line), with relatively stable error across day and night intervals. Shaded regions distinguish day (white) and night (grey) conditions. 30
7.6 Detection errors over time using the mean-difference method, with precipitation periods shaded in blue. 31
7.7 Detection errors over time using the KS-test method, with rain periods indicated in blue. 31

A.1 Function analysis 48
A.2 Application of the Basic Design Cycle in this project. 49
A.3 Morphological chart 52

B.1 Design considerations reference object. 55

D.1 Overview of the SD-card imaging process. 60

D.2 Swapfile configuration. 61
D.3 Example of .service file 63

E.1 Enable SSH and VNC settings in RPI environment. 64
E.2 Setting up a remote connection using RealVNC. 65
E.3 Set interfacing options. 65
E.4 Configure Wireless LAN connection. 66
E.5 Location of IP-address 66
E.6 Insert Host name/ IP-address 67
E.7 Login interface for PuTTY 67
E.8 Setup file transfer configuration. 68

F.1 Switching RPI boot-up configuration 70

List of Tables

3.1 Inventory of design files and their locations 8

4.1 Bill of materials for the prototype 10

5.1 Logged detection output per image cycle. 22

7.1 Summary of accuracy and precision metrics for both detection methods over 270 field measurements. 32
7.2 Comparison of water-level detection performance during daytime and nighttime conditions for both the mean-difference and KS-test method. 32
7.3 Summary of detection performance under dry and rainy conditions for both the mean-difference and KS-test method. 33
7.4 Average processing characteristics per cycle. 33
7.5 Estimated average power consumption over 1 hour of operation. 33

A.1 List of prioritized system requirements based on the MoSCoW method. 51

C.1 Summary of requirement evaluation with MoSCoW classification and performance status. 59

Nomenclature

Abbreviations

Abbreviation	Definition
CLI	Command Line Interface
CSI	Camera Serial Interface
GND	Ground
GUI	Graphical User Interface
IR	Infrared
KS	Kolmogorov–Smirnov
LED	Light Emitting Diode
MAE	Mean Absolute Error
OS	Operating System
POC	Proof of Concept
RPI	Raspberry Pi
RGB	Red Green Blue (color model)
RMSE	Root Mean Squared Error
ROI	Region of Interest
SFTP	Secure File Transfer Protocol
SSH	Secure Shell
VNC	Virtual Network Computing

Symbols

Symbol	Definition	Unit
N	Number of samples	[–]
μ	Mean	[varies]
σ	Standard deviation	[varies]

1

Hardware in context

Collecting accurate water level data is crucial for efficient river management, enabling precise calculation of river discharge when combined with velocity and cross-sectional data. Real-time water level monitoring systems play an essential role in disaster prevention by facilitating flood early-warning systems, which are critical in protecting lives and property [22, 18]. With the intensification of climate change, the probability of short-term heavy rainfall events has increased significantly, leading to more frequent and severe flooding. Reliable water level monitoring has become increasingly important for proactive disaster management and environmental protection.

1.1. State of the art

Current technologies for water level measurements typically fall into two categories: contact and non-contact sensors. Contact sensors, including pressure and floater sensors, are relatively low-cost and straightforward to operate, but are vulnerable to corrosion and flood-related debris [45].

Floater sensors usually only detect whether the water level is crossing a predefined threshold, whereas pressure sensors are more robust and able to store exact water level data. Floater sensors are therefore mostly used in the industrial sector to monitor tank levels and are budgeted around ~€100 [31]. Pressure sensors are commonly used for water level monitoring and are often placed in combination with a shaft to protect the device. Low-cost versions can be acquired from around ~€300 with an accuracy of ±1 cm [30].

Non-contact sensors such as radar [27] and ultrasonic sensors [5] offer protection from high water velocities and floating debris. Ultrasonic sensors are available in a wide cost range, but low-cost versions start around ~€100 with an accuracy of ±1 cm [23]. However, these sensors are sensitive to environmental factors like wind and vibrations. Radar is significantly more expensive, starting from ~€1000, but provides higher accuracy of ±2 mm [41] and is more stable under varying conditions such as fog and rain. Both types, however, are prone to vandalism due to their visibility on bridges or infrastructures directly above river channels.

Imaging technologies provide an alternative approach for water level monitoring and include three main methods: satellite imagery [19, 35, 36], images of water gauges [21, 20], and images directly capturing water level lines [33, 11, 13]. Advances in deep learning have improved automation in detecting water levels, using object recognition and semantic segmentation [26, 6, 8, 34, 12, 17, 39, 40, 15, 48].

However, these techniques often require the installation of water gauges and illumination systems for nighttime capturing which are challenging to maintain sustainably, especially in remote or resource-limited settings. Additionally, they often depend on a reliable power connection and a high-performance processing unit, increasing both cost and operational demands.

1.2. Research gap and objectives

There is a growing need for affordable, compact, sustainable, and off-grid water level monitoring systems that are suitable for remote regions and capable of operating both during the day and at night. While existing methods using floaters or pressure sensors are relatively low-cost, they are vulnerable to damage under flood conditions. Radar, ultrasonic, and advanced camera-based systems depend on stable power and are prone to vandalism.

1.3. Introducing the proof of concept

This paper presents a proof of concept (POC) for a compact, autonomous, and off-grid water-level monitoring system, tailored for use in remote or resource-limited environments. The system is energy-efficient, cost-effective, and designed to be less susceptible to vandalism and flood damage.

It integrates a low-cost imaging sensor with an onboard processor, housed in a discreet mounting setup to minimize exposure. Water level detection is achieved via visual sensing, leveraging optical differences, such as brightness and texture, between the water and its surroundings. This eliminates the need for contact-based measurement components.

The system was developed based on four core design criteria:

- **Flood & vandalism protection:** Low vulnerability to extreme water levels and vandalism.
- **Availability & continuity:** Water level data must be available within 5 minutes of capture; at least one detection every 10 minutes.
- **Cost-effectiveness:** Total material costs should not exceed €400.
- **All-condition operation:** Reliable detection both day and night, regardless of ambient light.

In this paper, we present the final product resulting from multiple design choices and iterative development. This is structured in the style of the *HardwareX* journal, with an emphasis on reproducibility, aiming to provide a solid foundation for potential future developments of this POC.

The following chapters are structured as follows: Chapter 2 provides a hardware description, listing all components and highlighting key design decisions. Chapter 3 presents a design file summary. Chapter 4 contains a bill of materials. Chapter 5 describes hardware and software build instructions and the main algorithm. Chapter 6 outlines operation procedures, Chapter 7 presents the field setup and results, Chapter 8 contains the discussion, and Chapter 9 concludes the findings. Fianally, chapter 10 outlines potential future work.

To ensure reproducibility and to illustrate the structured nature of the development process, the annex provides detailed documentation of each step. Annex A outlines the theoretical design methodology. Annex B presents the rationale behind key design decisions. Annex C evaluates the final system against the predefined design criteria. Annexes D, E, and F together form a user manual, offering step-by-step instructions for system setup, remote access, operation, and calibration. Finally, a development log is included in Annex G, documenting implementation notes and key design considerations.

2

Hardware description

The system consists of several components that collectively meet the criteria introduced in Chapter 1. The imaging module captures images of the waterline to be detected. A supporting illumination system, using infrared lighting, enables image capture during nighttime conditions. An onboard processing unit analyses the images and determines the water level. Once the water level is identified, the data is stored locally and transmitted to an online platform. The system is powered by a battery pack and housed within an enclosure to protect it from environmental influences. A simple mounting mechanism allows for easy short-term deployment. Finally, a reference object is used to enhance the accuracy of the water level detection. These components are described in more detail below.

A conceptual schematic of the system is shown in Figure 2.1, while Figure 2.2 and 2.3 displays the individual components used. Additionally, Figure 5.1 in Chapter 5 presents the hardware schematic, showing the full electrical layout of the system. A comprehensive overview of the design methodology (Annex A), component selection (Annex B), and criteria evaluation (Annex C) is provided in the Appendix.

2.1. Imaging module

The Raspberry Pi Camera Module V2 NoIR (Figure 2.2a) was selected for its Sony IMX219 sensor, which delivers 8 MP stills at up to 3280 × 2464 and supports 1080p30, 720p60, and 640 × 480 p60/90 video modes [28]. The sensor's 1.12 µm × 1.12 µm pixel pitch and 3.68 × 2.76 mm imaging area (¼″ optical format) offer high sensitivity and fine spatial resolution [10]. As the NoIR variant, this camera module omits the infrared (IR) cut filter, allowing near-infrared wavelengths to reach the sensor. This extends the detectable wavelength range from approximately 400–700 nm (visible spectrum) to about 400–1000 nm. The extended range enables reliable low-light and nighttime imaging, particularly when used with an IR LED illuminator [28].

The camera module is mounted flush in the enclosure lid to maintain a fixed, repeatable viewing angle. A polarization filter (Figure 2.2h) is affixed to the outside of the lid directly over the lens opening. This both seals the camera hole against dust and moisture and reduces specular reflections from the water surface, improving contrast and lowering noise in the captured images. The camera attaches via the MIPI CSI-2 interface and is driven using libcamera through the Picamera2 Python library [28].

2.2. Illumination system

The "ILH-IN01-85SL-SC211-WIR200" infrared LED emitter module (Figure 2.2b) is used to enable nighttime imaging at a wavelength of 850 nm, which falls well within the detectable range of the camera sensor. While both 850 nm and 940 nm modules are commonly used, the 850 nm option delivers higher radiant output per watt and longer effective range, whereas 940 nm LEDs are invisible to the naked eye and better suited for stealth applications. Since image quality has priority, the 850 nm module was selected. It operates at a 3.4 V forward voltage, a 5 V reverse voltage, and a maximum current of 1.5 A, with a 50° viewing angle and a radiant intensity of 1.23 W/sr [14]. The IR-LED control circuit was

implemented using a 1.2 Ω, 5 W resistor in series with an Adafruit MOSFET driver (Figure 2.2e). The MOSFET is controlled from a GPIO pin on the processing unit (Raspberry Pi Zero 2 W), so that the IR LED is powered only during image capture. Both the resistor and MOSFET are soldered onto a custom 60 × 90 mm PCB for compact integration. Building instructions are presented in Chapter 5.1

The LED is mounted in the enclosure lid, 65 mm from the camera lens and aligned using the same orientation. It is bonded to a heatsink (Figure 2.2c) with thermal interface fluid to ensure effective heat dissipation. Additionally, a diffusive plastic filter (Figure 2.2g) is installed over the IR emitter hole in the enclosure lid. This piece both seals the opening against dust and moisture and scatters the 850 nm illumination into a uniform field, reducing hot-spot glare and reflections that can introduce noise into the waterline detection.

2.3. Onboard processing unit

The Raspberry Pi Zero 2 W (Figure 2.2d) was selected for the processing unit because its 1 GHz quad-core ARM Cortex-A53 CPU and 512 MB of LPDDR2 SDRAM are sufficiently powerful to support real-time image analysis [29]. It includes onboard 2.4 GHz 802.11 b/g/n wireless LAN allowing wireless data transfer [29]. The 65 mm × 30 mm board features a CSI-2 camera connector for direct attachment of compact camera modules and runs Raspberry Pi OS with native Python support [29]. At 5 V the board consumes about 100 mA at idle and peaks at 449 mA under full load [3], making it compatible with small battery packs for reliable off-grid operation. While more energy-efficient alternatives such as the AMB82-Mini (RTL8735B) offer lower idle currents and deep-sleep modes [2], the extensive software ecosystem, community support, and compatibility with Python-based workflows made the Raspberry Pi Zero 2 W a practical and robust choice for rapid prototyping and algorithm development.

2.4. Data storage and communication

For basic data storage and operation, the Raspberry Pi Zero 2 W uses a 32 GB microSD card. However, because microSD cards can suffer failures in environments with wide temperature swings, a USB flash drive connected via a micro-USB-to-USB-A adapter cable is added. This drive serves as a redundant backup for both raw images and processed output. In addition to local storage, detected water levels are posted to the OpenRiverCam server [25] using API POST requests with an access token. These posts are scripted and integrated into the water level detection algorithm, enabling the system to upload its measurements automatically during each capture cycle. Internet connectivity is provided via the onboard Wi-Fi module of the Raspberry Pi Zero 2 W. During desk testing, the device connected through a local Wi-Fi network; in the field, a mobile hotspot was used to establish the connection.

2.5. Power supply and energy management

The power system centres on a Voltaic Systems V50 battery pack (13,400 mAh, 48.24 Wh Li-Ion) that supports “Always On” mode and is optimized for solar recharging (Figure 2.2f). Its internal charge controller maintains a maximum power point of approximately 5.2 V and accepts up to 2 A of solar input, potentially allowing direct connection of a small solar panel via USB. At 5 V, the V50 draws about 7 mA in ”Always On” standby and can deliver up to 4 A across two USB-A ports.

The system’s peak draw ($\sim$500 mA at 5 V for the Pi Zero 2 W and camera) yields about 26 h of autonomy on battery alone. Though not explored in this research, pairing the battery pack with a modest solar panel (such as a 6 W, 6 V panel [43] providing $\sim$1 A in peak sun) could enable continuous daylight recharging and sustain true autonomous off-grid operation. The 114 × 78 × 26 mm unit mounts securely inside the enclosure, and its dual USB-A outputs feed the Pi and illumination module.

2.6. Enclosure and mounting assembly

A Schneider Electric Thalassa TBS casing (Figure 2.2i) was used as weather-resistant enclosure, with seals retained everywhere except at the IR-LED and camera-lens openings, where holes were drilled into the lid. To maintain modularity, a double-floor platform made from a PVC cutout lined with velcro tape was installed. On the back of the enclosure, a custom hinge assembly fabricated from steel L-profiles and plywood enables straightforward short term deployments without compromising the en-

closure's integrity.

2.7. Reference object for waterline detection

To increase the effectiveness of the water level detection system, a uniform background is preferred. This can be achieved in various ways, depending on site-specific conditions. For example, white paint could be applied to a stable structure located in the river, such as a bridge pillar or rock formation. For the field testing conducted in this research, a reference object was constructed with a modular design, allowing for easy installation and removal.

To create contrast between water and air, a 4 mm matte-white foam PVC sheet (600 × 500 mm) was mounted onto 15 mm plywood. Two 25 mm-diameter rods secured to the plywood combined with tension straps allow stable depth and height adjustment (Figure 2.3d-2.3e). The PVC's high diffuse reflectance makes waterlines more detectable compared to the noisy natural background.

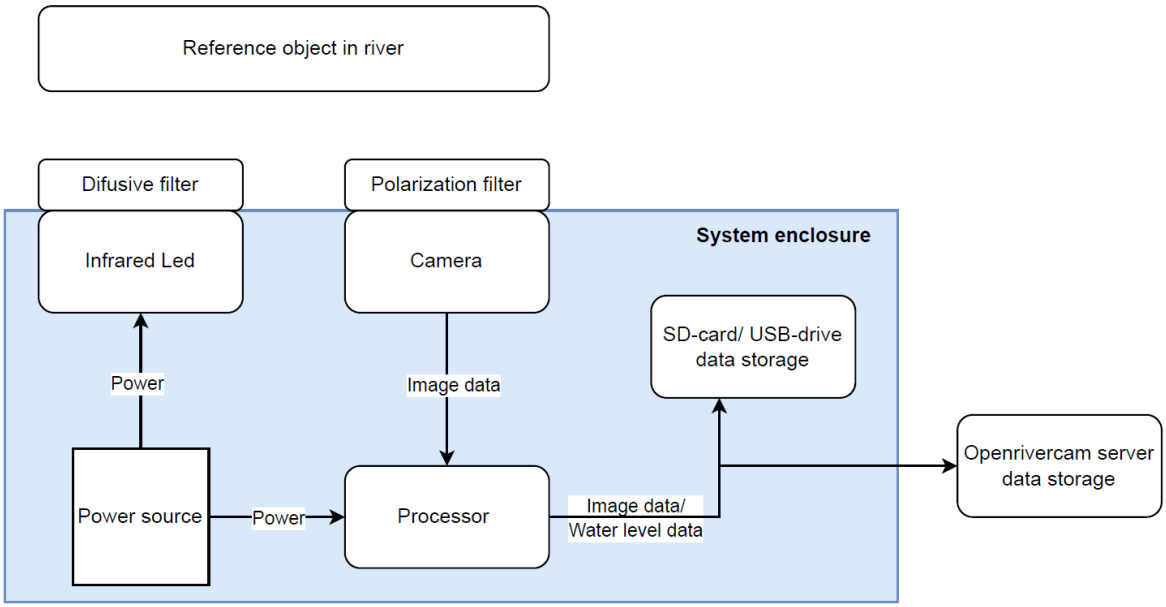
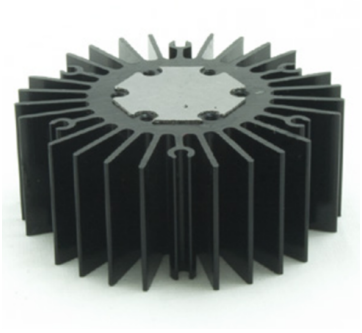


Figure 2.1: Conceptual system architecture of the image-based water-level monitoring prototype.

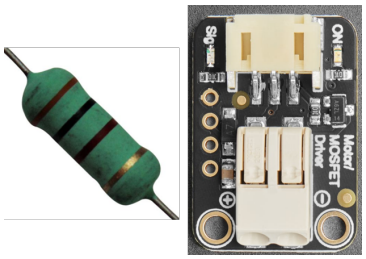
(a) PRI NoIR camera module [28]

(b) IR emitter module [14]



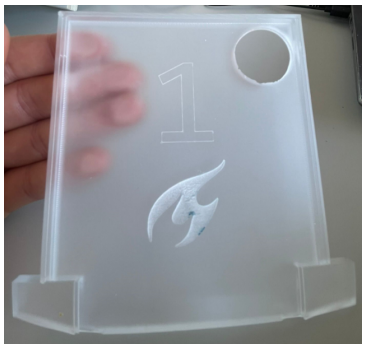
(c) LED heatsink [16]

(d) Raspberry Pi Zero 2 W [29]



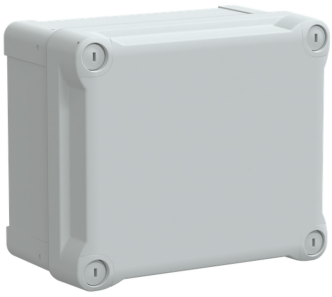
(e) Resistor [24] & MOSFET driver [9]

(f) V50 battery pack [44]



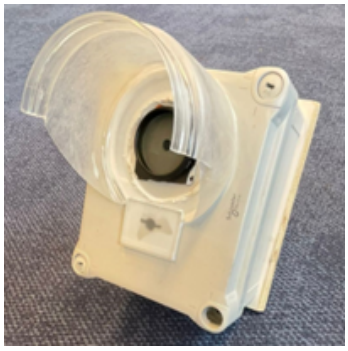
(g) Diffusive filter

(h) Circular polarisation filter [37]

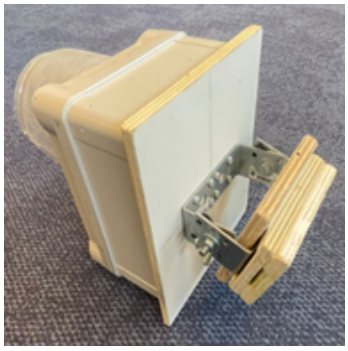


(i) Schneider electric enclosure [32]

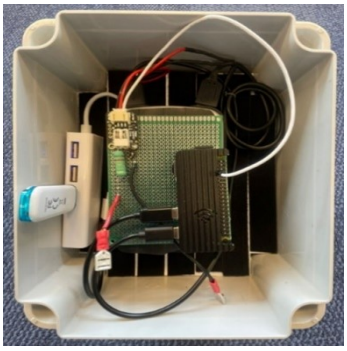
Figure 2.2: Hardware components used in the water-level monitoring system. For more detailed information on hardware components, see section 4.1



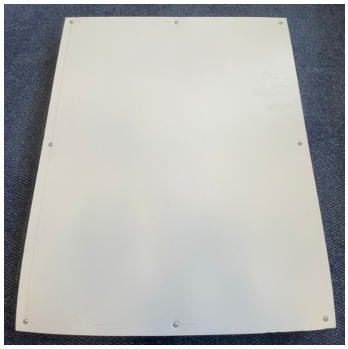
(a) Front view final assembly



(b) Rear view final assembly



(c) Inside view final assembly



(d) Front view reference object

(e) Rear view reference object

(f) Deployment of reference object in river

Figure 2.3: Final system assembly and reference object deployment.

3

Design files summary

All files can be found in the online repository: *https://github.com/AEC-Rens/Water-level-monitoring-system-repository*, as well as in the annex. A hardware schematic file (PDF) is included, showing how the camera module, power supply, and illumination module connect to the Raspberry Pi as shown in Figure 5.1. All components are ready-to-use, and build instructions are provided in Chapter 5.

A system setup manual offers instructions for initial use of the Raspberry Pi environment (annex D). The remote access manual guides the user through accessing the operating system and transferring data between systems (annex E). Instructions on how to calibrate and operate the system during deployment are provided in the operation and calibration manual (annex F).

Finally, a complete image file of the operating system is included, containing all required packages, software installations, calibration procedures, and the Python script, organized within the correct directory structure. Additionally, the experimental data, including captured images and algorithm outputs, are also available in the repository through a publicly accessible Google Drive link.

Table 3.1: Inventory of design files and their locations

Design filename	File type	Description	Location of file
Hardware_schematic	PDF	Schematic showing connection of electrical components	GitHub link
System_setup_manual	PDF	Instructions on how to setup the operating system for first use	GitHub link
Remote_access_manual	PDF	Instructions on how to communicate with the system	GitHub link
Operation_and_calibration_manual	PDF	Instructions on how to perform calibration and operate the system	GitHub link
SD_card_backup_10042025	`.img`	Backup of complete operating system and installed packages/scripts	GitHub link
Data_and_resources	`.md`	Link to experimental data, captured images, and algorithm outputs	GitHub link
Scripts	Folder	All Python scripts used in water-level detection algorithm	GitHub link

The file structure of the water level detection algorithm (which can be found in the GitHub repository) is clarified in Figure 3.1

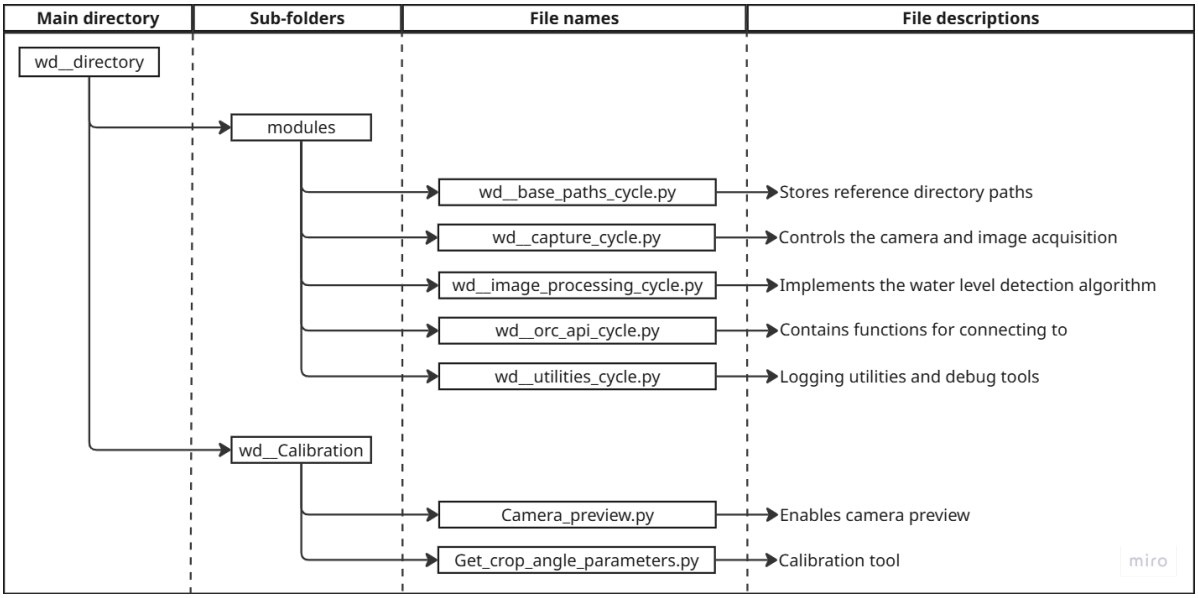


Figure 3.1: Overview of Python file structure with descriptions.

4

Bill of materials summary

The bill of materials provides a detailed list of all required components. From the Raspberry Pi Zero 2 W starter kit and Pi Camera Module V2 NoIR to the ILH-IN01 IR emitter, custom PCB, and Thalassa TBS enclosure. Each annotated with its function, supplier, quantity, total cost, and source. In total, 21 line items sum to €289.20, giving an exact financial overview for budgeting and reproducibility.

Table 4.1: Bill of materials for the prototype

#	Component	Function	Supplier	Cost (€)	Source
A	Raspberry Pi Zero 2 W – starter kit	Central processing and control unit	RaspberryPi	€44.95	raspberrystore.nl
B	Raspberry Pi camera V2 NoIR	Image capture module	RaspberryPi	€17.95	raspberrystore.nl
C	Camera Cable – 200mm	CSI interface cable	RaspberryPi	€1.15	kiwi-electronics.com
D	Camera housing V3/Arducam	Camera protection	Arducam	€5.99	raspberrystore.nl
E	Aluminium case for RPi Zero	RPi protection	Waveshare	€5.92	kiwi-electronics.com
F	MOSFET Driver JST-PH	Power control	Adafruit	€4.59	kiwi-electronics.com
G	JST PH Cable – 200mm	IR LED cable	Adafruit	€1.44	kiwi-electronics.com
H	V50 USB Battery Pack	Portable power	Voltaic Systems	€83.48	kiwi-electronics.com
I	USB Micro B to USB-A cable	Power/data cable	Onlinekabelshop	€2.49	onlinekabelshop.nl
J	1.2Ω Resistor, 5 W	Current limiter	Multicomp Pro	€0.59	farnell.com
K	IR Emitter Module, 850 nm	Night illumination	ILS	€9.78	farnell.com
L	LED Star Heatsink	LED cooling	ILS	€12.48	farnell.com
M	USB-A (m) open wire	Power wiring	Onlinekabelshop	€2.99	onlinekabelshop.nl
N	Prototyping Board – 4x6cm	Circuit platform	Kiwi Electronics	€1.68	kiwi-electronics.com
O	Velcro tape 20mm	Mounting tool	Onlinekabelshop	€5.99	onlinekabelshop.nl
P	Thalassa TBS Casing	Waterproof case	Schneider Electric	€19.70	se.com
Q	Silicone Wire – 2m 26AWG	Low-voltage wire	Kiwi Electronics	€1.20	kiwi-electronics.com
R	USB 2.0 Flash Drive 16GB	Backup device	Philips	€11.50	123accu.nl
S	Mounting hardware (misc.)	System install	–	€20.00	Hardware Store
T	Circular polarized filter	Optical filter	UwCamera	€19.95	uwcamera.nl
U	Foamed PVC board (4mm)	Waterline reference	kunststofplatenshop	€15.38	kunststofplatenshop.nl
			Total	**€289.20**	

5

Build instructions

This chapter details the end-to-end assembly of the water-level monitoring system. In brief, first, the hardware components; power and control PCB, processing unit, camera module, IR illuminator and filters are mounted and interconnected within the weather-proof enclosure (Section 5.1). Next, the software environment is prepared: the operating system is imaged onto the SD card, system settings and packages are configured, the detection scripts are deployed, remote-access is enabled, and the main cycle is registered as a boot-up service (Section 5.2). These instructions ensure that both the physical build and the software setup can be reproduced. Finally, the water level detection algorithm is elaborated (Section 5.3). We provide these steps in more detail below.

5.1. Hardware

The following subsections describe the step-by-step assembly and wiring of all hardware components within the weather-proof enclosure. A hardware schematic, showing all electrical components and connections is displayed in Figure 5.1.

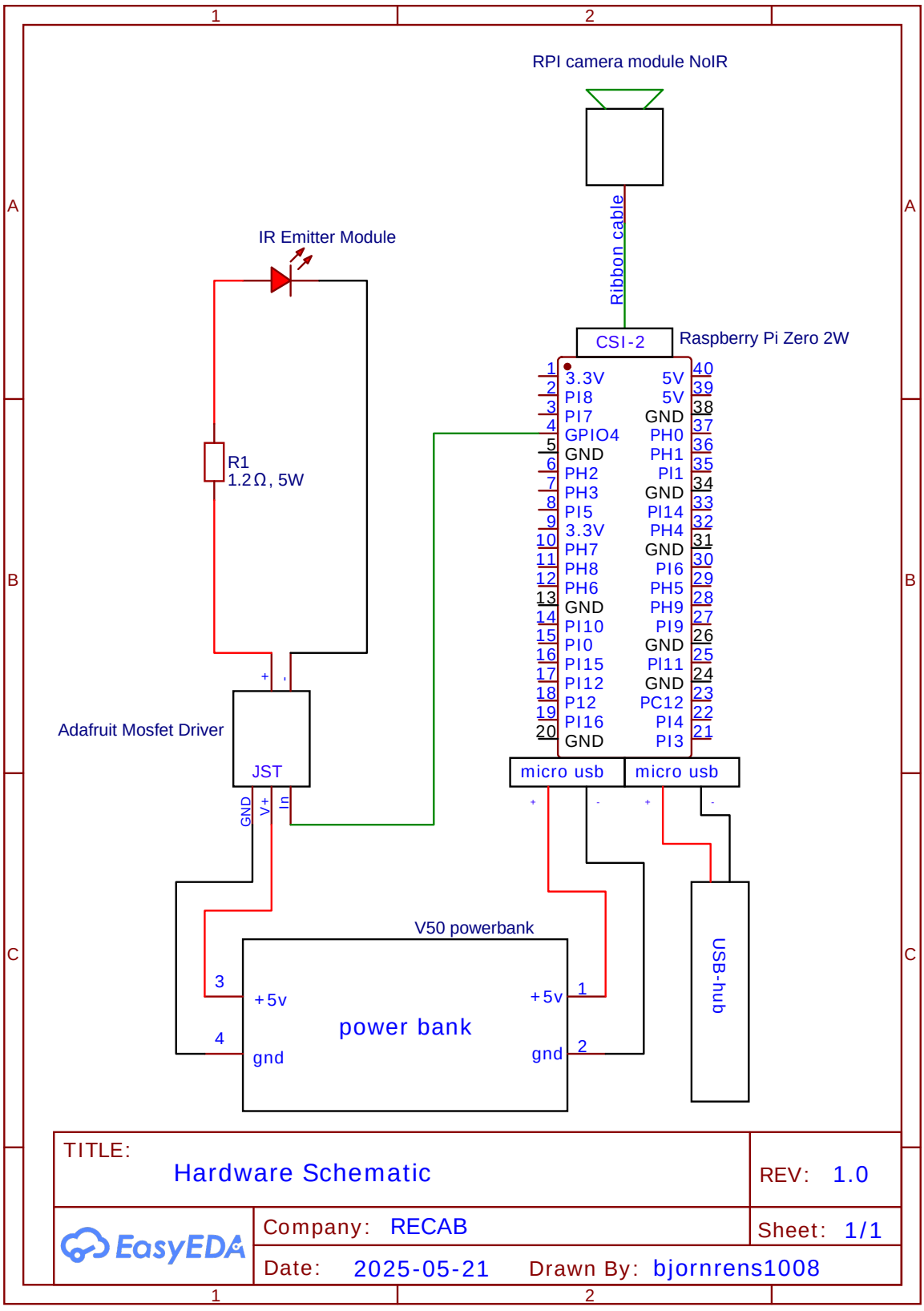


Figure 5.1: Hardware schematic of the camera-based water-level monitoring system.

5.1.1. Preparing power supply and PCB

To enable GPIO-controlled IR illumination and modular power connections, a MOSFET driver and a 1.2 Ω, 5 W resistor were soldered to a 60 × 90 mm prototyping PCB, together with power (+) and ground (GND) leads. Cable lugs were crimped onto the external wire ends using a crimping tool to allow secure, removable connection of the IR LED. The JST connector cable was modified by stripping and soldering its power and ground wires to the corresponding conductors of a USB-A power cable, permitting powering via any 5 V USB source. Heat-shrink tubing was applied over each joint for electrical insulation (Figure 5.2).

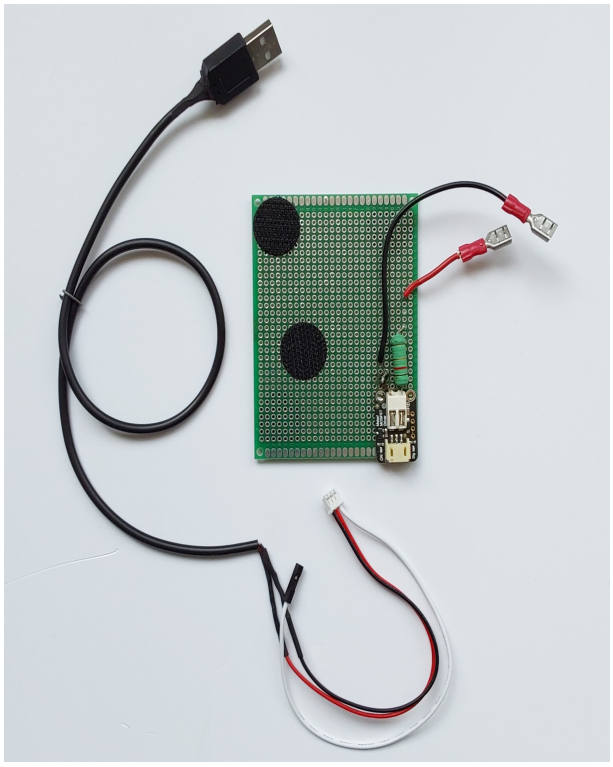


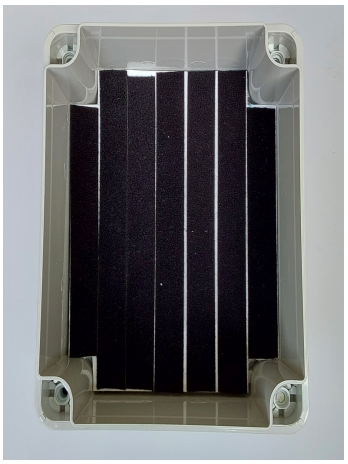
Figure 5.2: Overview of the soldered components, including a 1.2Ω, 5W resistor and MOSFET driver integrated on a 60×90mm PCB, with output leads for IR LED connection.

5.1.2. Mounting mechanism & modular design

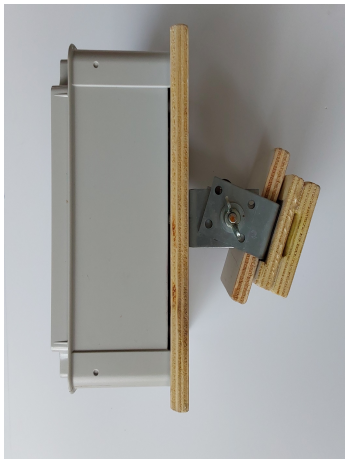
To support rapid assembly and component exchange, a 4 mm PVC platform (cut with a utility knife) was press-fitted into the enclosure and lined with 20 mm Velcro tape (Figure 5.3a-5.3b-5.3c). Steel L-profile brackets were assembled into an adjustable hinge mechanism on the enclosure's back panel; a wingnut and rubber washer lock the hinge's position, enabling stable vertical alignment (Figure 5.3d). The enclosure was then attached to 12 mm plywood via its factory corner-mount screw holes located outside the sealed volume to preserve weather resistance. Opposite the hinge, a wooden block with milled slits served as an anchor for tension straps, allowing the entire assembly to be quickly secured in the field (Figure 5.3e). Plywood and steel were chosen for their low cost and ease of fabrication for short term deployments; for long-term use, other materials should be investigated.

(a) Interior of the Schneider Electric Thalassa TBS enclosure used to house all system components.

(b) Custom-cut 4mm PVC platform designed for press-fit installation into the enclosure base.



(c) Final platform installation with applied Velcro strips to allow modular placement and replacement of components.



(d) Side view of the assembled hinge mechanism using steel L-profiles and 12mm plywood for vertical alignment and structural support.

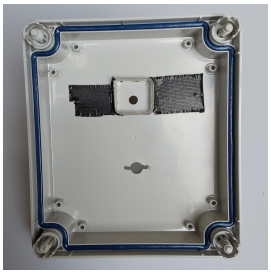
(e) Close-up of the adjustable hinge secured with a wingnut and rubber washer for fine-tuned positioning and quick deployment.

Figure 5.3: Assembly and mounting of the system enclosure.

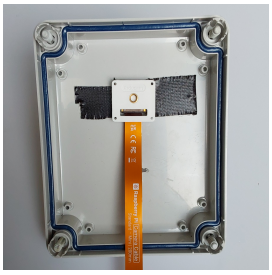
5.1.3. Installing camera module

To integrate the camera and maintain environmental sealing, two apertures (9 mm for the lens, 5 mm for the IR LED) were drilled in the enclosure lid using a step drill (Figure 5.4a). To minimise optical interference in the captured images, the lens and IR emitter should not be positioned too closely together; therefore, the apertures were spaced sufficiently apart during drilling. An Arducam lens housing was shortened by 2 mm at the front so that its transparent cover sits flush with the lid (Figure 5.4f-5.4e); it was then affixed inside the lid with adhesive and black tape to block stray light and LED indicators (Figure 5.4b). A polarizing filter was screwed into its circular adapter and was cut to size and adhered over the camera aperture to seal against dust and moisture while reducing water-surface glare. A transparent rain shield, fabricated from a plastic container and roughened with sandpaper was attached externally to diffuse incident light during daytime operation (Figure 5.4d). Finally, the CSI-2 camera connector ribbon cable is attached to the camera module (Figure 5.4c).

(a) Drilled apertures in the enclosure lid for the camera lens (9mm) and IR LED (5mm), made using a step drill to ensure precise alignment.

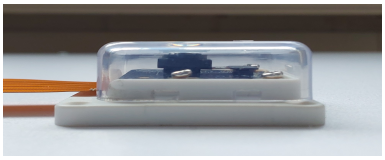


(b) Interior view of the shortened Arducam camera housing mounted flush against the lid and sealed with tape to block stray light.

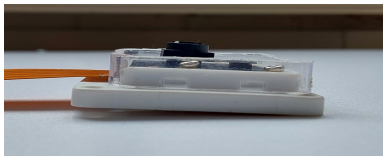


(c) Installation of the camera module inside the lid, with the ribbon cable routed for connection to the main board.

(d) External view showing the rain shield fabricated from a translucent plastic container, affixed to diffuse direct sunlight.



(e) Side views of the modified camera housing: original.

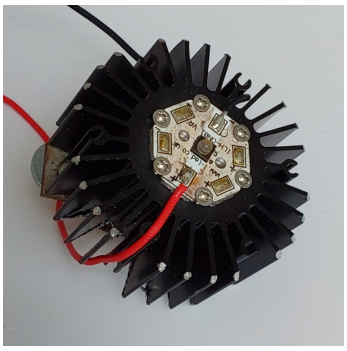


(f) Side views of the modified camera housing, shortened by 2mm to align the lens flush with the outer surface for a watertight fit.

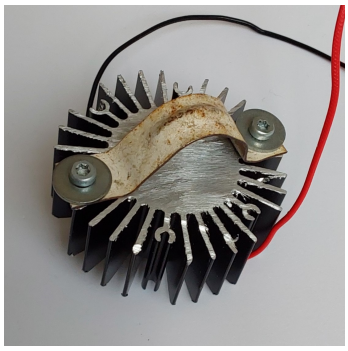
Figure 5.4: Integration of the camera module into the system enclosure.

5.1.4. Installing IR-illumination module

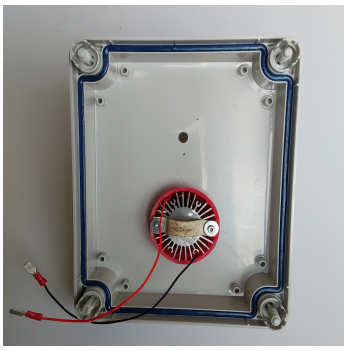
The ILH-IN01-85SL-SC211-WIR200 IR LED emitter was secured to an LED-star heatsink with supplied mounting screws and thermal interface fluid (Figure 5.5a). A saddle bracket was fixed to the heatsink's rear so the assembly can be withdrawn for maintenance without stressing the LED (Figure 5.5b). The heatsink assembly was enclosed in a protective plastic cap and bonded to the interior of the enclosure lid (Figure 5.5c-5.5d-5.5e). The MOSFET/resistor PCB was mounted adjacent to the heatsink and connected via shrink lugs (Figure 5.5f). A diffusive plastic filter was attached over the IR LED to seal the opening and scatter 850 nm illumination 5.4d, reducing hot-spot glare and reflections.



(a) Top view of the ILH-IN01-85SL-SC211-WIR200 infrared LED emitter mounted to a star-type aluminium heatsink using thermal paste and screws to ensure efficient heat transfer.



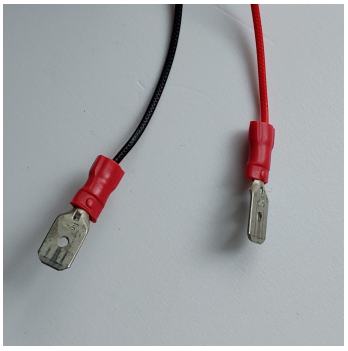
(b) Rear view showing the saddle bracket used to attach the heatsink securely while allowing for maintenance removal without stressing the LED.



(c) Internal view of the enclosure lid with the LED-heatsink assembly installed and wired.

(d) Image of the 3D-printed plastic cap used to enclose the LED module, featuring a central aperture for light transmission.

(e) Fully assembled unit with the diffuser glued in place to the cap, scattering emitted IR light and sealing the enclosure.

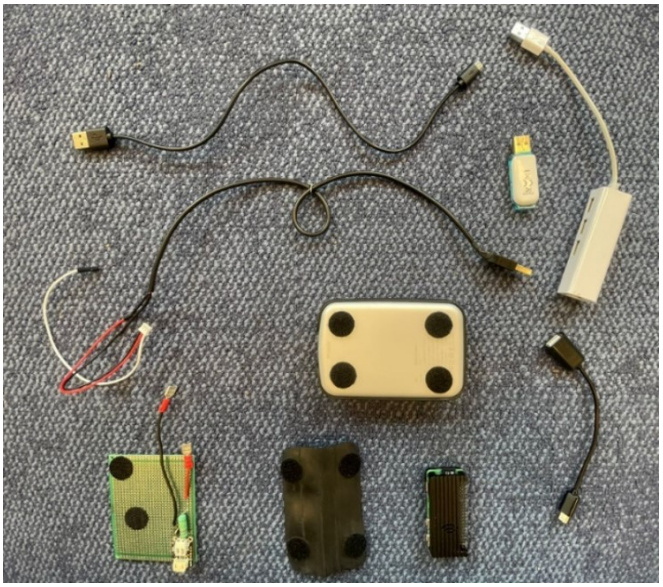


(f) Crimped shrink-lug connectors on the LED leads for modular connection to the control PCB.

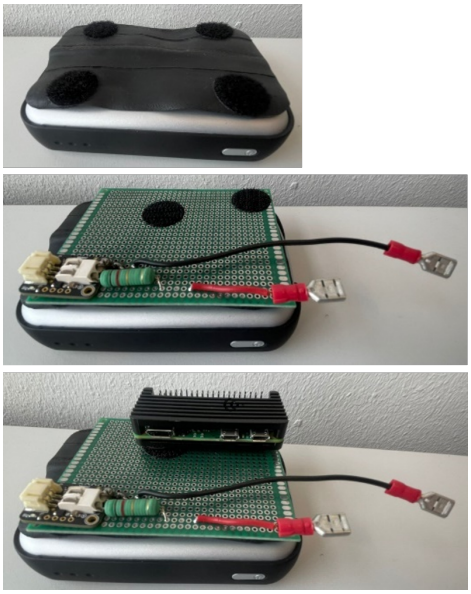
Figure 5.5: Integration of the IR LED module with heatsink, enclosure, and diffuser.

5.1.5. Device assembly

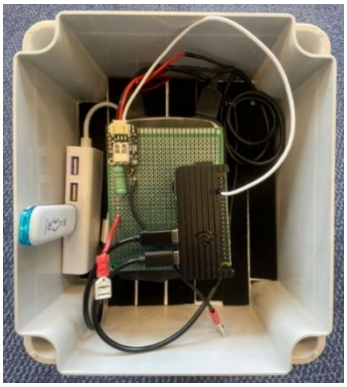
The Voltaic V50 battery pack was secured to the enclosure floor with Velcro tape; a rubber pad between the pack and PCB prevents short circuits (Figure 5.6b). The Raspberry Pi Zero 2 W is enclosed in its aluminium case, which also functions as a passive heatsink. It is secured via Velcro to an available area of the 60 × 90 mm PCB immediately adjacent to the MOSFET driver and the 1.2 Ω resistor that regulate power to the IR LED (Figure 5.6c-5.6d). A USB hub was mounted alongside the battery pack and connected to the Pi with a USB-to-micro-USB adapter cable. This enables the user to connect both a USB flash drive and utilities like a keyboard and mouse to the system in case remote access software fails. Power lines from the battery were routed to both the MOSFET board and the Pi (Figure 5.6c). Finally, the IR-LED cable and the CSI-2 ribbon cable were inserted into their respective ports. Once connections were confirmed, the battery pack was activated, and the lid was re-installed and screwed down to restore weather resistance (Figure 5.6e).



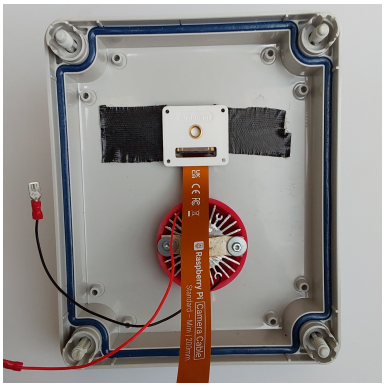
(a) Overview of all hardware components prior to installation, including battery pack, Raspberry Pi, IR control board, USB hub, storage devices, and cables.



(b) Velcro-mounted components on the battery pack include the insulating pad, IR LED control PCB, and Raspberry Pi Zero 2 W with heatsink, enabling a compact and removable integration.



(c) Top-down view of the full internal configuration, showing the Pi, power routing, USB hub, and flash drive.



(d) Inside lid with IR LED, camera module, and corresponding power and ribbon cable connections in place.

(e) Exterior view with reinstalled lid and transparent rain shield covering the camera aperture to protect against glare and weather exposure.

Figure 5.6: Final assembly and integration of power, processing, and connectivity components.

5.1.6. Reference object

A 4 mm matte-white foam-PVC board (600 × 500 mm) was fastened to 15 mm plywood using stainless-steel screws (Figure 5.7a). Saddle brackets were then attached to the plywood's rear, avoiding penetration of the PVC face and holding two 25 mm-diameter rods (Figure 5.7b-5.7c). In combination with adjustable tension straps, this arrangement provides a modular mounting system for field deployment (Figure 5.7d).

(a) Front view of the 4mm matte-white foam-PVC board (600×500mm), offering high contrast for reliable waterline detection.

(b) Rear view showing the 15mm plywood backing, to which the PVC is attached using stainless-steel screws for structural stability.

(c) Side view highlighting the layered construction and saddle brackets that hold two 25mm mounting rods for depth and height adjustability.

(d) Field deployment of the assembled reference board in a natural river channel, stabilized using tension straps and anchored between adjacent wooden branches.

Figure 5.7: Construction and installation of the modular reference board for waterline detection.

5.2. Software

To enable image capture, illumination control, image processing, and the storage and posting of water level detection results, a specific software configuration must be implemented. This section describes the software required to successfully run the water level detection algorithm. First, the acquisition of the operating system for the Raspberry Pi is discussed, followed by a list of the required software packages. Finally, methods for remotely accessing the system are covered. While this chapter introduces the general concepts of these software components, detailed instructions can be found in Annex D: System Setup Manual and Annex E: Remote Access Manual.

5.2.1. System setup summary

To prepare the Raspberry Pi environment, the Raspberry Pi OS (64-bit, Bookworm) is flashed onto a microSD card using the official Raspberry Pi Imager. During this process, the system can be preconfigured with login credentials, Wi-Fi settings, SSH/VNC access, and locale preferences. This results in a ready-to-use software environment, suitable for both direct and remote operation.

Direct operation can be achieved by connecting a keyboard and mouse via the USB hub and attaching a monitor or portable display using a mini-HDMI to HDMI cable. Remote operation is enabled through software tools, as described in Section 5.2.2 (see Annex D.1 for system setup manual).

For stability during image processing tasks, the default swapfile size is increased from 100 MB to 1024 MB via the terminal. This is necessary because the Raspberry Pi Zero 2 W has limited physical RAM (512 MB), which can lead to system crashes or unresponsiveness when handling memory-intensive operations such as image analysis. Increasing the swapfile size provides additional virtual memory, allowing the system to offload some of the processing load to disk and operate more reliably under heavier workloads (see Annex D.2 for instructions).

All necessary software dependencies are installed through a single terminal command. These include system-level camera tools and drivers:

- `libcamera-apps`
- `libcamera-dev`
- `v4l-utils`

Alongside these, the following core Python libraries are required for image processing, data handling, and hardware control:

- `python3-picamera2 0.3.27` (Picamera2 API & libcamera bindings)
- `python3-pillow 11.2.1` (PIL image handling)
- `python3-numpy 2.2.6` (array operations)
- `python3-matplotlib 10.3.1` (plotting backend)
- `python3-scipy 1.15.3` (signal processing & statistics)
- `python3-requests 2.30.0` (HTTP for ORC API integration)
- `python3-psutil 7.0.0` (system resource monitoring)
- `python3-rpi.gpio 0.7.1` (GPIO pin control)

For detailed instructions and troubleshooting, refer to Annex D.3.

To ensure autonomous operation, the main Python script is configured as a `systemd` service, allowing it to start automatically on boot and restart upon crashes. The service file is placed in `/etc/systemd/system/`, and activated via `systemctl`. This setup enables reliable, self-recovering operation in off-grid deployments (See annex D.4 for detailed instructions).

Full instructions, configuration files, and command listings are provided in:
Annex D: System Setup Manual
Online repository: GitHub link

5.2.2. Remote access and data retrieval

To enable remote control and file access, both graphical (VNC) and terminal-based (SSH) remote access methods are supported. Graphical access is particularly useful during debugging or calibration, as it provides an interactive visual interface. Terminal-based access, on the other hand, allows users to connect to the system while the water level detection algorithm is running, enabling monitoring without interrupting its operation. Additionally, file transfers are managed via SFTP using WinSCP.

Remote desktop access is provided through RealVNC. After enabling the SSH and VNC interfaces on the Raspberry Pi, a connection can be established using RealVNC Viewer and the device's IP address. This allows full graphical user interface (GUI) access from a workstation on the same network (see Annex E.1 for detailed instructions).

Headless operation is supported via SSH, using PuTTY. The SSH server is enabled through the raspi-config tool, and Wi-Fi credentials are entered to connect the Pi to a mobile hotspot or local network. After identifying the Pi's IP address, users can log in via PuTTY for full terminal access without a display (see Annex E.2 for detaild instructions).

File transfers are handled using WinSCP, an SFTP client. Once configured with the Pi's IP address and login credentials, WinSCP allows users to upload or download images, logs, and CSV files. The full project directory, including scripts and configuration files, can also be transferred to the Pi in one step (see Annex E.3 for detaild instructions).

Full instructions, configuration files, and command listings are provided in:
Annex E: Remote Access Manual
Online repository: GitHub link

5.3. Water level detection algorithm

With the hardware components described in Section 5.1 and the software environment configured in Section 5.2, this section focuses on the core of the system: the implementation of the water-level detection algorithm. In Chapter 1, the basic working principle of the algorithm was introduced. This chapter builds on that foundation by presenting the full implementation and describing the underlying Python code. It begins with an explanation of the image pre-processing steps, followed by a detailed overview of the two algorithmic approaches used for water-level detection: the mean-difference method and the Kolmogorov–Smirnov (KS) test method.

5.3.1. Image pre-processing

The water-level detection algorithm runs autonomously on the Raspberry Pi Zero 2 W, analysing vertical pixel-intensity patterns in each captured image. Each input image, originally in RGB format with pixel values ranging from 0 to 255, is first normalized to floating-point values between 0 and 1. Depending on the selected detection mode, the algorithm then extracts a specific channel to generate a single detection array. This can be either the normalized RGB values (original mode), the brightness component (Value channel), the colour tone (Hue), or the colour intensity (Saturation) from the HSV colour space. This preprocessing procedure is visualised in Figure 5.8.

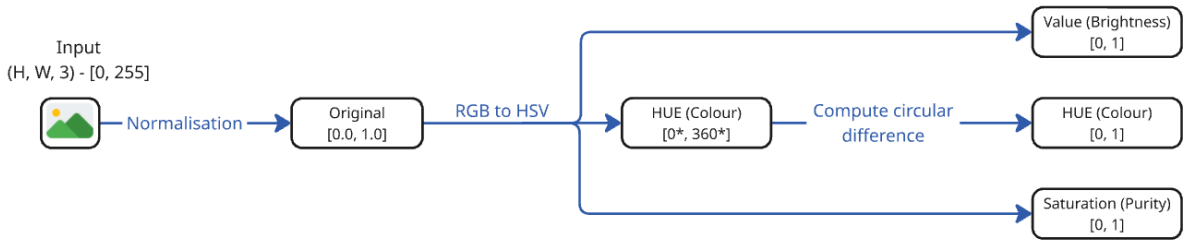


Figure 5.8: Preprocessing workflow for image-based waterline detection.

When using the Hue channel, a special transformation is required to account for its circular scale, which ranges from 0° to 360°. Without this correction, colours that are perceptually similar (such as red (0°)

and purple (360°) as shown in figure 5.9) can appear artificially distant in numerical terms, leading to false high contrasts. To correct for this, the algorithm computes the minimal circular difference, ensuring that hue differences reflect actual colour shifts rather than numeric discontinuities. This improves the reliability of waterline detection in scenes with subtle colour gradients.

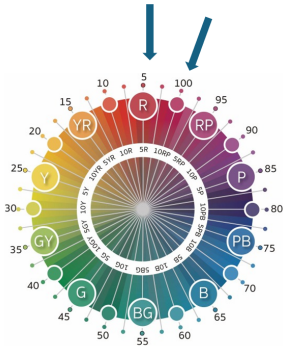
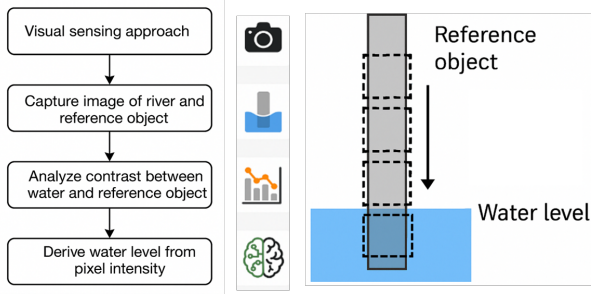


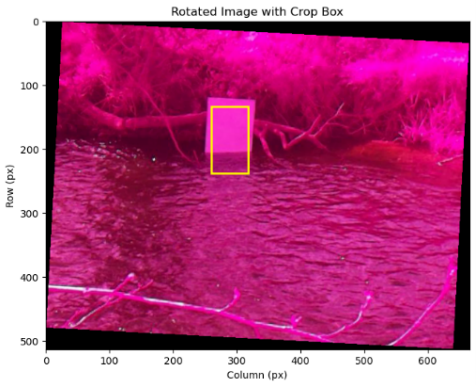
Figure 5.9: Illustration of the circular nature of the Hue channel [47]

5.3.2. Mean-difference method

To identify the water level, an algorithm was developed. To locate the waterline, the algorithm identifies shifts in pixel intensity along the vertical axis of the reference object, where the waterline typically causes a distinct visual transition. To detect this, the detection array is divided into a series of small horizontal ”boxes” as visualized in Figure 5.10a. For each adjacent pair of boxes, the algorithm calculates the absolute difference in their mean intensity. For the hue channel, a circular difference is used to account for the wraparound nature of hue values. These raw differences are then passed through a one-dimensional Gaussian filter to reduce random noise and highlight structured intensity changes. The resulting smoothed signal is normalized into a probability distribution over the vertical axis, emphasizing the most likely positions of the waterline.



(a) Conceptual illustration showing image processing principle; contrast between wet and dry areas allows the waterline to be inferred from image data.



(b) Example image from field testing showing a cropped region (yellow box) containing the reference object, used for waterline detection.

Figure 5.10: Schematic overview of the visual waterline detection process.

The peak of this distribution represents the most probable vertical pixel coordinate of the waterline within that image channel. In cases where multiple peaks exist, the highest (i.e., most prominent) peak in the distribution is selected as the candidate waterline for that channel. To improve robustness under varying lighting and background conditions, this process is repeated across all four image channels (original RGB intensity, value, hue, and saturation) as shown in Figure 5.11. For each channel, the maximum value of the smoothed difference signal (before normalization) is recorded as an indicator of detection strength. The channel with the highest detection strength is selected, and its corresponding peak is used as the final water-level measurement.

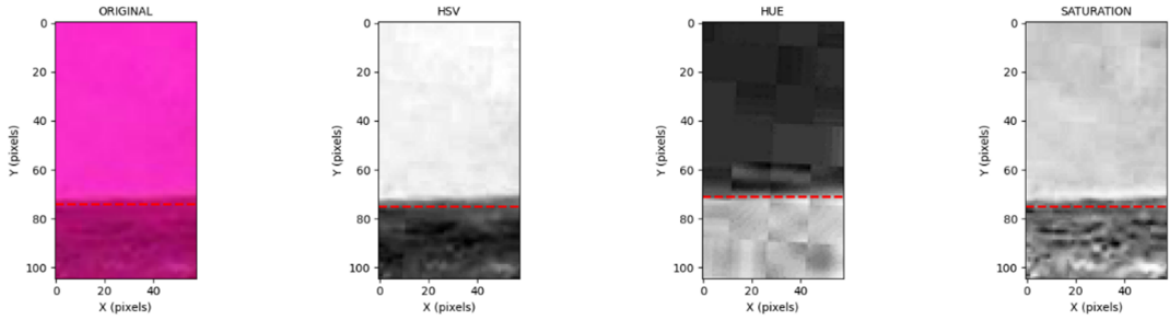


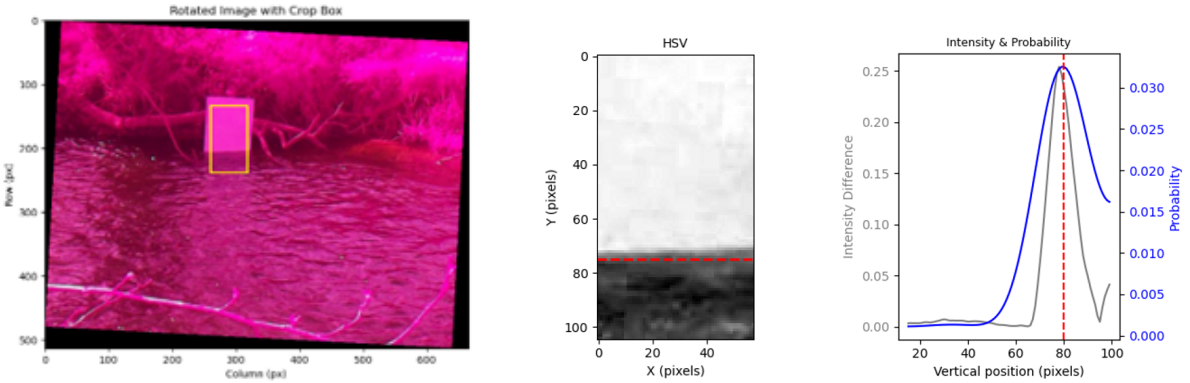
Figure 5.11: Algorithm results displaying the most probable waterline location (red dashed line) for each image channel.

Each cycle's results (including the per-channel waterline positions, the selected detection mode, and its corresponding score) are appended to a CSV file for later analysis as shown in Table 5.1. In addition, visual outputs are saved to support quality control, including the cropped image (Figure 5.12a), the detected waterline overlay (Figure 5.12b), and the intensity difference and probability curves (Figure 5.12c).

Table 5.1 displays the algorithm output. The WL_<channel> columns (column 2-5) reflect the detected y-coordinate representing the waterline in image space. The best_mode column (column 6) reflects the image channel with the highest detection strength. The best_score value (column 7) reflects the maximum intensity difference between adjacent boxes of that channel, prior to normalization. Since all intensities are scaled to the [0–1] range, best_score also falls within this range, where 1 indicates maximum contrast between two comparison boxes, and 0 indicates no difference.

image_name	WL_original	WL_hsv	WL_hue	WL_saturation	best_mode	best_score
capture_140235.jpg	207	208	204	208	hsv	0.25

Table 5.1: Logged detection output per image cycle.



(a) Captured image overlaid with the detection crop box (yellow), isolating the reference object region used in the analysis.

(b) Extracted crop visualized in the HSV brightness channel, with the detected waterline marked in red.

(c) Plot showing the absolute intensity difference curve (grey) and its corresponding normalized probability distribution (blue). The vertical red dashed line indicates the selected waterline position corresponding to the peak detection score.

Figure 5.12: Visual outputs for quality control and verification of the waterline detection process.

5.3.3. KS-test method

As an alternative to the mean-difference approach, a Kolmogorov–Smirnov (KS)-based method has also been implemented. Instead of comparing average pixel intensities between adjacent boxes, this method evaluates the entire distribution of pixel values. Specifically, it calculates the KS statistic, which

quantifies the maximum difference between the cumulative distribution functions (CDFs) of two adjacent regions. An illustration of this concept is shown in Figure 5.13, where the vertical arrow indicates the KS statistic, the largest vertical gap between the two CDFs (one shown in red, the other in blue). This single value reflects how dissimilar the pixel distributions are between the two regions.

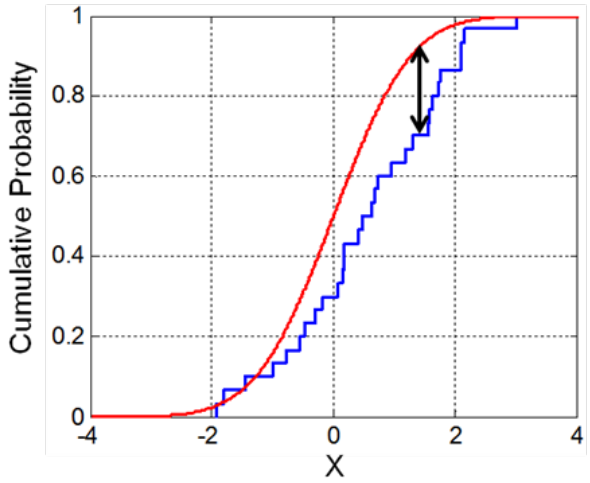
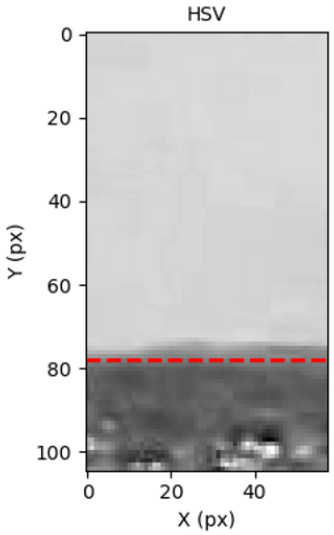


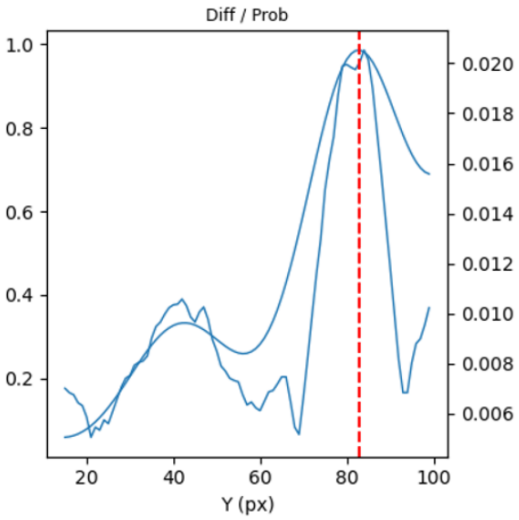
Figure 5.13: Illustration of the Kolmogorov–Smirnov (KS) statistic.[7]

Because the KS test is sensitive to distribution shape rather than just average intensity, it can detect subtle shifts in intensity that the mean-difference method might miss. This is especially useful in noisy or low-contrast environments. For hue values, a sine transform is applied to preserve circular continuity before computing the CDFs.

Once the KS statistic is computed for each vertical step, the resulting values are smoothed using a one-dimensional Gaussian filter, normalized, and passed through the same peak-detection process used in the mean-difference method. The strongest peak across all channels is selected as the final water-level estimate. The output of the KS-test method are shown in Figure 5.14.



(a) Cropped and normalized image showing the vertical structure of the reference object with a dashed red line indicating the detected waterline.



(b) Corresponding KS statistics (blue curve) computed along the vertical axis, and their smoothed probability distribution. The red dashed line marks the vertical position of the highest peak in the normalized signal, which is selected as the most probable waterline location.

Figure 5.14: KS-test detection process in the HSV channel.

6

Operation instructions

This chapter provides instructions for deploying and operating the water-level monitoring system in the field. Section 6.1 covers initial setup and orientation, including verifying service status and installing the reference board. Section 6.2 describes the calibration and verification procedure to align the optical and algorithm parameters (in GUI mode). Section 6.3 explains how to run the device in command line interface mode (CLI) for unattended, periodic measurements and how to retrieve data on demand.

6.1. Setup and orientation

Before sealing the enclosure, establish a graphical access using RealVNC as described in Section 5.2.5. Verify that the water-level detection service is active with the appropriate systemctl commands. Once the service is confirmed to be running correctly, the enclosure may be sealed and prepared for deployment.

Next, install the reference object in the river so that its face is positioned perpendicular to the camera lens. This perpendicular orientation is essential in the current POC because each pixel is directly mapped to real-world dimensions. If the reference board is tilted relative to the camera, parts of the board that are closer to the lens will appear larger (each pixel represents a smaller real-world area) compared to parts that are farther away, where each pixel represents a larger area. This perspective distortion would lead to inaccuracies in water level detection. To allow more flexible camera angles in future versions, georeferencing techniques would be required (see Section 8.3: Limitations).

Once positioned, secure the reference object firmly and ensure the camera's line of sight is unobstructed by overhanging branches or floating debris (Figure 6.1). While mounting methods may vary depending on the site, the adjustable hinge and tension-strap system provides sufficient flexibility to attach the enclosure to either vertical or horizontal supports.

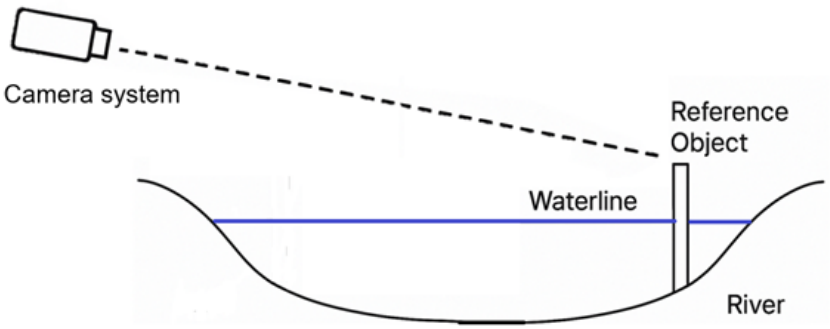


Figure 6.1: Schematic overview of the deployment setup

6.2. Calibration and verification

To ensure accurate water-level detection, the system must be calibrated after installation. This involves aligning the camera with the reference board and extracting key processing parameters. General overview of this process is described below. A more detailed description and checklist can be found in annex F.

First, physical alignment is required to avoid perspective distortion. The camera must be positioned perpendicular to the reference board so that pixel dimensions accurately reflect real-world distances. A live preview is used to fine-tune this orientation before locking the camera mount.

Next, image processing parameters such as rotation angle and cropping boundaries are configured. These parameters ensure the detection algorithm focuses on the correct region of interest and compensates for any slight misalignments in the camera setup. The system's measurement interval is also reviewed and adjusted to suit the intended deployment frequency.

Once these settings are defined, the setup is verified by running a test detection cycle. This confirms that the camera, algorithm, and storage pipeline are functioning as intended. Visual and numerical outputs are inspected to detect any misalignment, image noise, or processing errors.

During desk tests, GUI mode required more energy compared to operating in command line interface (CLI) mode. That is why, after calibration is complete, the system is switched to CLI mode to reduce energy consumption. Remote access is maintained via SSH, allowing users to monitor performance, inspect logs, or restart services without requiring a graphical display. If configured for remote uploading, successful data transmission to the OpenRiverCam server is verified as part of final validation.

Detailed calibration and verification instructions are provided in Annex F: Operation and calibration Manual.

6.3. Operation

Once calibration is complete and the Raspberry Pi is configured to boot in headless CLI mode, the system begins automatic measurement cycles at fixed intervals, as defined in the configuration file. During each cycle, the system illuminates the reference board, captures an image, runs the detection algorithm, optionally uploads the result, and then enters standby until the next cycle, requiring no user interaction.

Manual control remains possible: a cycle can be triggered on demand, or the service can be restarted or paused as needed. Output files, including raw images, annotated results, and CSV logs, are stored locally and can be retrieved via remote access (see annex F.2 for commands). For data access or debugging, the system can temporarily be switched back to GUI mode and accessed via VNC and SFTP tools like WinSCP. After retrieval, CLI mode is restored to maintain low-power operation.

Periodic maintenance involves checking battery levels, ensuring the reference board remains aligned and free of obstructions, and inspecting the enclosure and mounting hardware. With this setup, the system delivers reliable, unattended water-level measurements, while still allowing straightforward manual access when required.

Detailed operation and maintenance procedures are provided in Annex F: Operation and calibration Manual.

7

Validation and characterization

This chapter evaluates the prototype's performance through field validation and detailed characterization. Section 7.1 describes the experimental deployment including location, mounting, environmental conditions, and manual reference measurements, while Section 7.2 outlines the data-acquisition protocol, output formats, and reference-level definition. Section 7.3 presents the detection errors of both the mean-difference and KS-test methods, including outlier analysis, diurnal and precipitation effects, and overall statistics. Finally, Section 7.4 assesses the system's energy consumption per cycle and over extended duty cycles to gauge its suitability for off-grid operation.

7.1. Experimental setup

The deployment took place on the Kleine Geul River near Wittem (Gulpen-Wittem, Limburg, the Netherlands). The site features narrow, loess- and silt-lined banks and a steep gradient that produces fast, highly dynamic flows in response to rainfall [46]. The location of the experimental setup is shown in Figure 7.1, including coordinates.

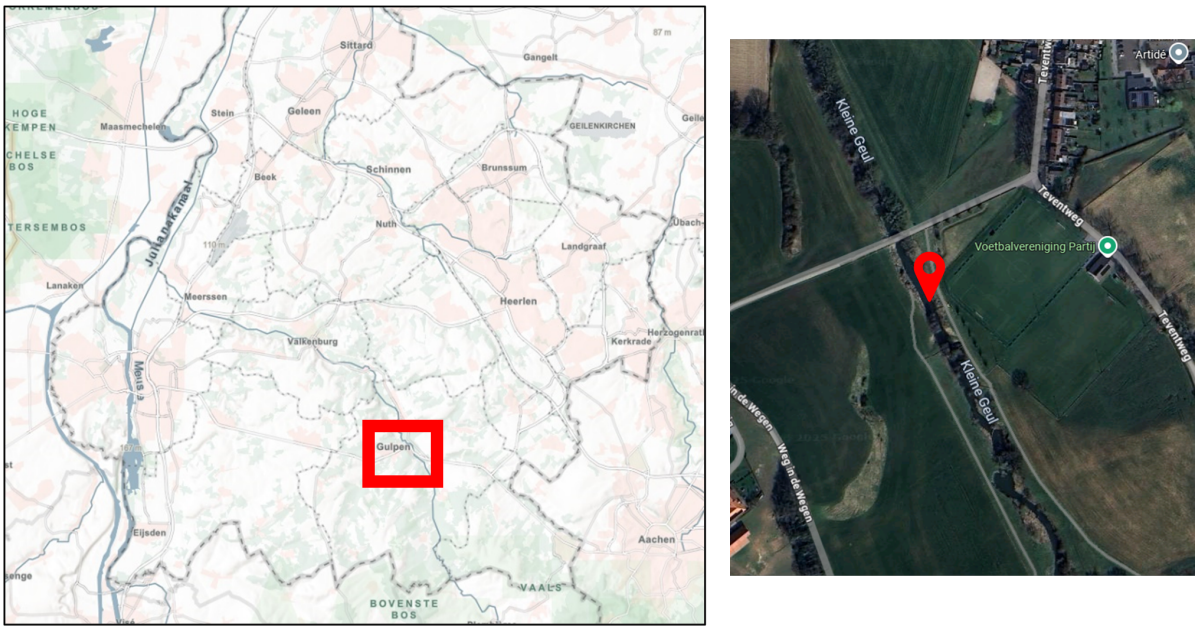


Figure 7.1: Field deployment location (50.80762°N, 5.91350°E).

The system operated continuously for 42 h, with a single manual check and service restart at the 20 h mark to confirm output quality and battery status.

Figure 7.2: Field deployment of the water-level monitoring system at the Kleine Geul.

The weather-resistant enclosure was affixed to a sturdy, elevated support above anticipated flood levels, facing the opposite bank. The reference object was mounted partly submerged, perpendicular to the camera's optical axis and parallel to the river flow to ensure consistent framing. All components were secured with the tension-strap mounting system outlined in Section 5.1.2.

Days 1 and 2 were predominantly sunny with intermittent clouds. Light rain fell during the night between Day 2 and the morning of Day 3, briefly altering illumination and surface reflectivity [42].

7.2. Data acquisition

This section summarizes the timing, data formats, and reference measurements collected during the 42 h field deployment.

During field testing, the device was configured as follows:

- Capture interval: After each detection cycle, the system rested for 540 s (9 min). Given processing times of 20–40 s, this delivered an effective measurement frequency of just under 10 min, meeting the availability requirement.
- Infrared LED usage: The IR emitter remained active during day and night (only when capturing) to confirm system operation visually and to minimize illumination-related variability.
- Network configuration: Real-time posting to the OpenRiverCam API was disabled due to lack of Wi-Fi coverage on-site.
- Operating mode: CLI mode was used to reduce power consumption; USB-backup functionality was disabled, as it was not yet compatible with CLI operation.

A total of 270 measurement cycles were recorded over two continuous deployment segments (approximately 20 h followed by 22 h) for a combined duration of 42 h.

Each cycle produced three primary outputs under `/home/bjorn/wd__directory/results/`:

- Raw images (`raw_images/`): 640 $\times$ 480 px JPEG frames captured by the camera.
- Rotated & cropped views: full-frame JPEGs showing the applied rotation and crop window overlay as shown in Figure 5.12a
- Mode-specific overlays: figures (all in one PNG file) combining the detected waterline overlay on the cropped view with its corresponding intensity-difference or KS-statistic graph as shown in figures 5.11 & 5.12c.
- Summary log (`algorithm_results.csv`): CSV entries listing the filename, waterline pixel position for each mode, selected best mode, and score as shown in Table 5.1.

A resolution of 640 $\times$ 480 pixels was selected for image capture, as the algorithm demonstrated robust performance at this scale. Preliminary tests using higher resolutions (e.g., 1024×768) showed only marginal gains in detection precision while significantly increasing runtime.

An acoustic sensor system located approximately 3 m downstream of the prototype recorded water levels at 15 min intervals throughout the 42 h trial. Because the sensor's readings exhibited very low variability (std = 4 mm), the reference water level was treated as effectively constant for validation purposes . All camera-based detections were then compared against this constant in image-space.

To define this constant in image space, a custom Python script was used to manually annotate the waterline in a set of 42 images (the first image of each hour). The mean of these 42 pixel coordinates defined the image-space validation level. Algorithm outputs were compared to this baseline, and errors were converted to millimetres using an site-specific conversion rate (1px = 6.7mm).

To express detection errors in physical units (millimetres), a pixel-to-length conversion factor was estimated using the known width of the foam-PVC reference board. With a real-world width of 500 mm and an image width of 75 pixels, a scale of approximately 1 px = 6.7 mm was derived. This scale assumes that the reference board remained perpendicular to the camera's optical axis during the deployment.

7.3. Results

This section presents the error characteristics of the two detection methods. Both the mean-difference method and the KS-test method produced comparable results in proximity to the actual water level, with an average mean absolute error (MAE) of 1.8 cm for both approaches. While the KS-test method demonstrates greater robustness under varying conditions (particularly in response to diurnal lighting changes) the mean-difference method achieves slightly higher precision overall.

The remainder of this section elaborates on the field testing results in more detail. First, the raw output data is presented, along with a discussion of outliers for each method. This is followed by an analysis of diurnal and precipitation-related influences on detection performance. Finally, key performance statistics are reported.

7.3.1. Raw data and outliers

Figure 7.3 presents the time series of detection errors (defined as the difference between the detected waterline and the manually annotated reference in cm) for both the mean-difference method and the KS-test method over the 42-hour field deployment. All captured images and corresponding analysis outputs, including overlays, are available in the online repository (GitHub link).

For the mean-difference method, four notable outliers beyond ±10 cm are visible. The first two occurred around 16:00–17:00, coinciding with low-angle sunlight that produced glare on the water surface. Although a circular polarization filter was installed to suppress such reflections, it had been reattached without verifying its proper rotational alignment. As a result, the filter was ineffective, allowing strong reflections to degrade image contrast and impair the algorithm's performance (see repository for captured images).

The latter two outliers for this method appeared around dusk (20:00–21:00). During this transition from daylight to darkness, the camera's extended shutter times captured residual ambient light, introducing blur and reducing contrast. Both of which negatively affected detection accuracy.

For the KS-test method, three significant outliers with errors near –40 cm appear around 09:00 on the second day. Upon reviewing the algorithm output, these errors appear to stem from incorrect scoring of the colour channel outputs. In these cases, the mode selection mechanism incorrectly prioritized a misaligned or low-contrast channel, resulting in a detection failure.

Despite these specific cases, both methods show consistent performance across most of the deployment period, with error values clustered closely around the reference level. The outliers are isolated and traceable to identifiable environmental or algorithmic issues, indicating overall system reliability.

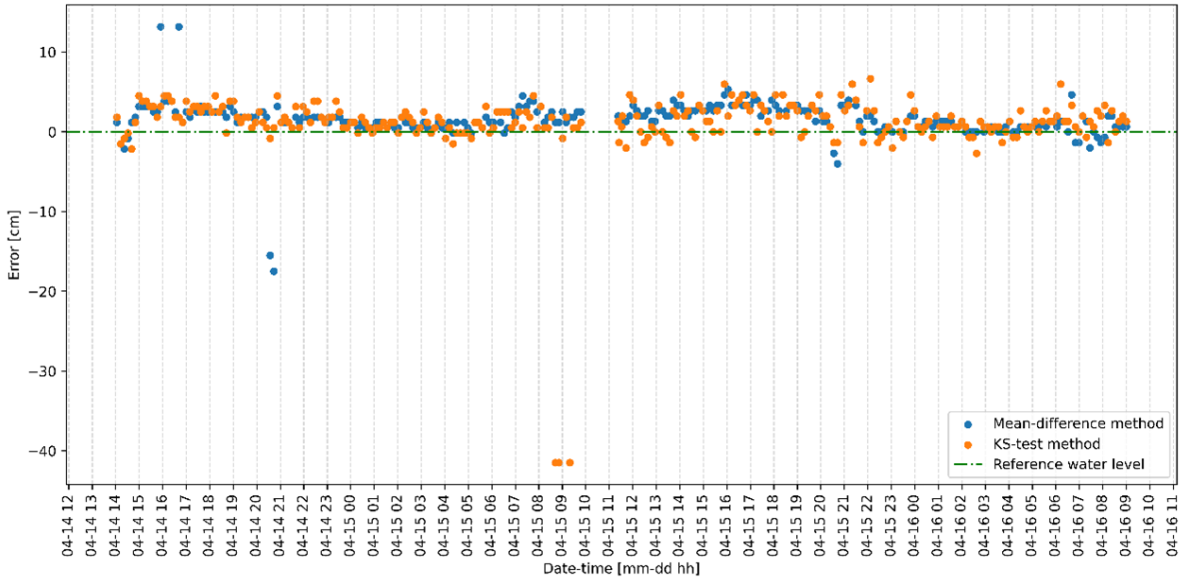


Figure 7.3: Time series of detection error (cm) for both the mean-difference and KS-test methods across the 42-hour field deployment.

7.3.2. Diurnal patterns

Introducing day (white background) and night (grey background) shading in the error plots (Figures 7.4-7.5) reveals a clear diurnal pattern in the mean-difference method. Errors tend to cluster higher above the zero-error line during daylight hours, indicating a systematic positive bias. This is confirmed by the fitted 24-hour sinusoidal trend (red line), which captures the regular daily fluctuation in detection error. The orange dashed line represents the bias, computed as the mean of all error values across the deployment period: 1.57 cm for the mean-difference method (Figure 7.4) and 1.02 cm for the KS-test method (Figure 7.5). These values reflect the average overestimation of water depth relative to the fixed reference level.

To improve visibility of the diurnal trend, the vertical axis range in both plots has been limited. As a result, the extreme outliers discussed in the previous section are not shown here. However, they are still included in the bias calculation and thus contribute to the mean error line.

The KS-test method exhibits more stable error distribution across the 24-hour cycle. Although the plot includes a 24-hour sinusoidal fit for consistency, the trend line does not clearly align with the actual day–night cycle, suggesting that no significant or consistent diurnal pattern is present. This indicates that the KS-test method is less sensitive to illumination changes, even if it produces slightly more scatter overall. The lower average bias and absence of systematic error shift further reinforce its robustness to environmental variability.

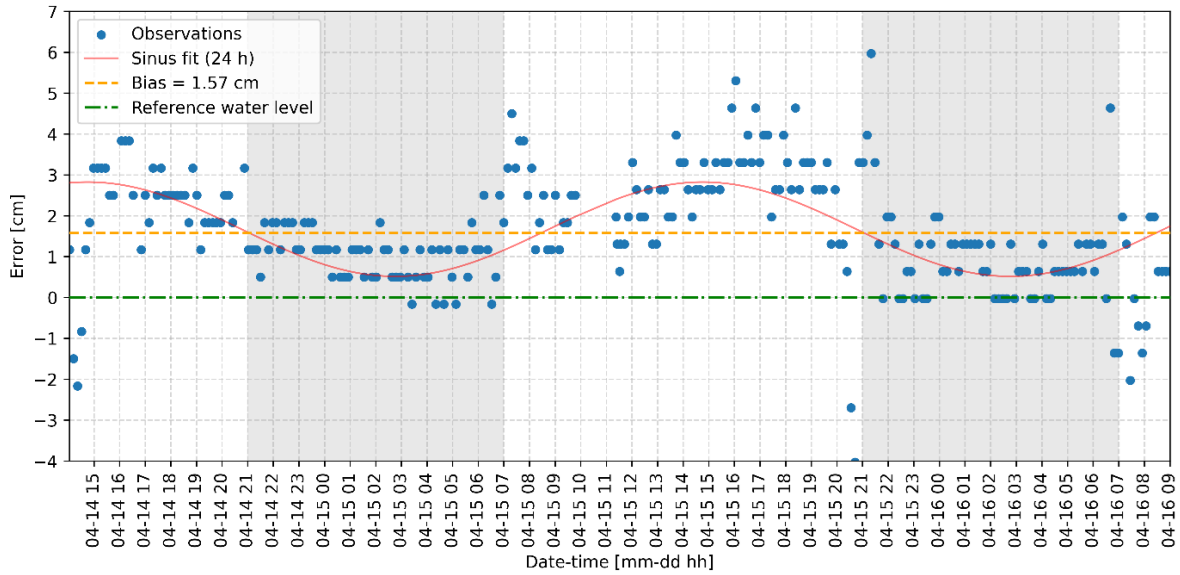


Figure 7.4: Detection error over time for the mean-difference method. A 24-hour sinusoidal curve (red line) highlights the diurnal bias, with a clear positive shift during daylight hours. The mean bias across the deployment was 1.57cm (orange dashed line). The green dashed line represents the zero-error reference level. Day (white) and night (grey) periods are shaded accordingly.

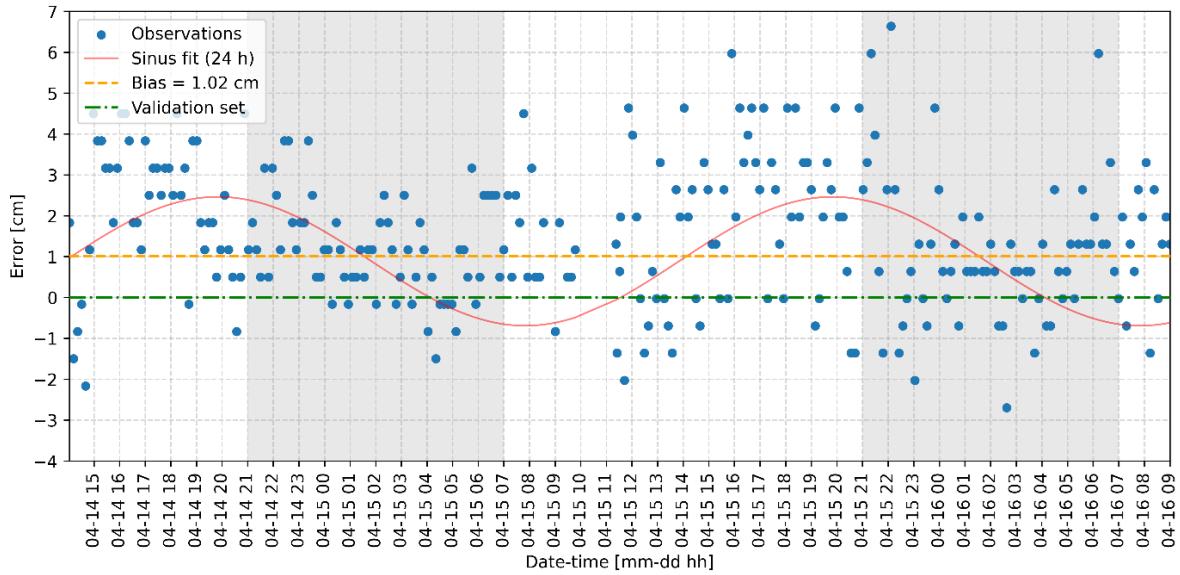


Figure 7.5: Detection error over time for the KS-test method. A sinusoidal fit (red line) is shown for comparison, but no clear diurnal pattern is observed. The average bias was 1.02 cm (orange dashed line), with relatively stable error across day and night intervals. Shaded regions distinguish day (white) and night (grey) conditions.

7.3.3. Influence of precipitation

Figures 7.6 and 7.7 show the detection errors over time for both methods, with precipitation periods highlighted in blue. Visually, errors appear slightly higher during rainfall events. However, this effect cannot be conclusively attributed to precipitation, as all rainfall occurred during daytime hours, when detection errors were already elevated due to diurnal lighting influences (Section 7.3.2).

It is important to note that only 51 out of 270 measurement cycles occurred during rain, limiting the strength of any conclusions. To fully assess the system's robustness in wet conditions, further testing is needed across a wider range of precipitation intensities, durations, and environmental settings.

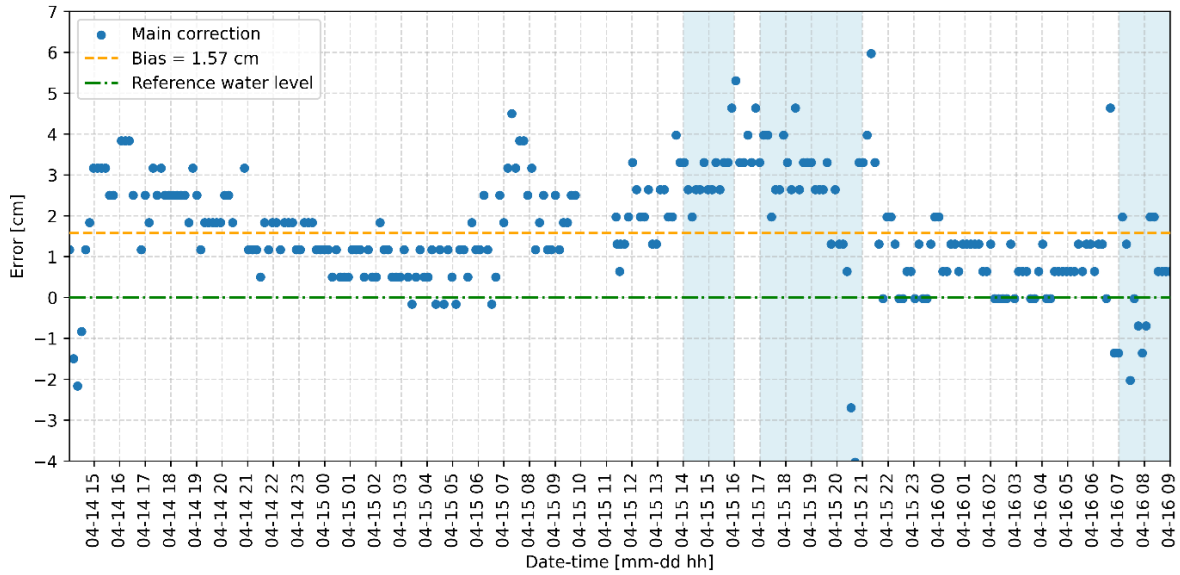


Figure 7.6: Detection errors over time using the mean-difference method, with precipitation periods shaded in blue.

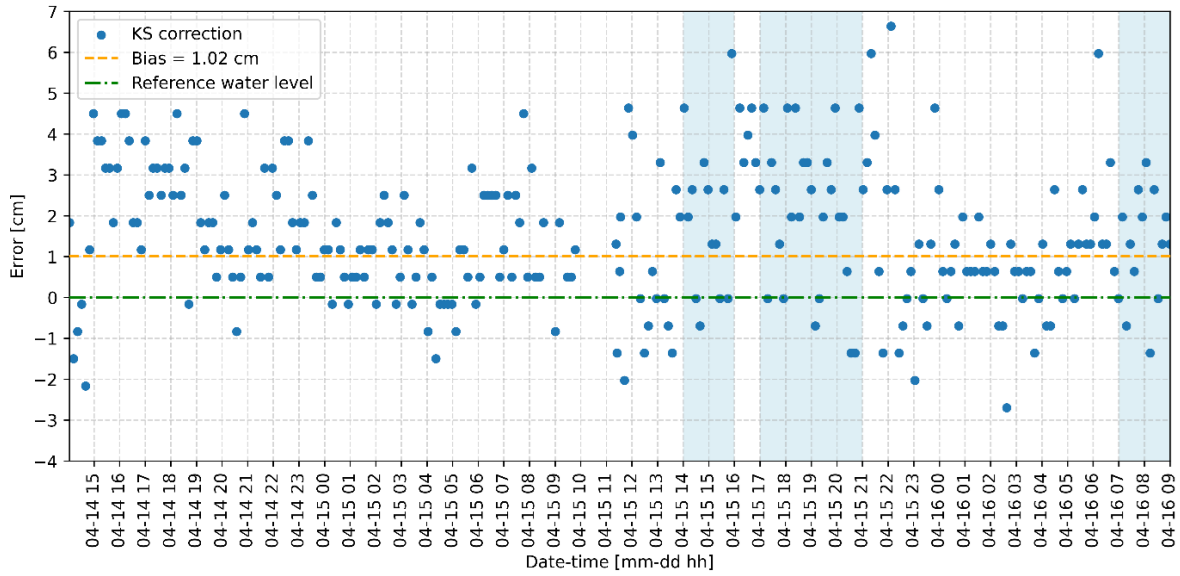


Figure 7.7: Detection errors over time using the KS-test method, with rain periods indicated in blue.

7.3.4. Performance statistics

Table 7.1 summarizes the overall accuracy and precision of both detection methods across all 270 measurement cycles. The mean-difference method shows a slightly higher bias of 1.57 cm compared to 1.02 cm for the KS-test method, but achieves better precision, with a standard deviation of 2.30 cm versus 4.82 cm for the KS-test. This reflects a more consistent performance with fewer extreme deviations.

In terms of absolute error, the mean-difference method records a lower mean absolute error (MAE) of 1.97 cm, and a lower root mean square error (RMSE) of 2.79 cm, compared to 2.23 cm and 4.92 cm respectively for the KS-test method. Notably, while both methods reach similar median accuracy (1.30 cm), the KS-test exhibits greater variability, including a minimum error of –41.50 cm, whereas the mean-difference method's minimum error is –17.50 cm.

The proportion of measurement cycles falling within specific error thresholds also highlights these differences. While the KS-test performs slightly better at the ±1 cm level (33.7% vs. 25.2%), the mean-difference method outperforms at higher thresholds: 95.6% of its errors fall within ±4 cm, compared to 91.5% for the KS-test.

To provide a common benchmark for comparison with conventional water-level sensors, accuracy is also expressed as the half-width of the 95% error interval. That is, the maximum deviation that includes 95% of all errors. Both methods yield the same 95% accuracy bound of ±4.64 cm, despite their different error distributions.

Table 7.1: Summary of accuracy and precision metrics for both detection methods over 270 field measurements.

Statistic	Formula	Mean-Difference	KS-Test
Sample Size	$N = \text{len}(y)$	270	270
Bias (cm)	$\text{Bias} = \frac{1}{N}\sum y_i$	1.57	1.02
Median (cm)	$\text{Median}(y)$	1.30	1.30
Precision (cm)	$\sigma = \sqrt{\frac{1}{N-1}\sum(y_i - \bar{y})^2}$	2.30	4.82
MAE (cm)	$\text{MAE} = \frac{1}{N}\sum\lvert y_i\rvert$	1.97	2.23
RMSE (cm)	$\text{RMSE} = \sqrt{\frac{1}{N}\sum y_i^2}$	2.79	4.92
Min Error (cm)	$\min(y_i)$	-17.50	-41.50
Max Error (cm)	$\max(y_i)$	13.17	6.64
95% Accuracy (cm)	$\max(\lvert P_{2.5}\rvert,\ \lvert P_{97.5}\rvert)$	4.64	4.64
% $\leq$ 1 cm	$\left(\frac{\text{count}(\lvert y_i\rvert \leq 1)}{N}\right) \times 100\%$	25.19%	33.70%
% $\leq$ 2 cm	$\left(\frac{\text{count}(\lvert y_i\rvert \leq 2)}{N}\right) \times 100\%$	64.81%	65.56%
% $\leq$ 3 cm	$\left(\frac{\text{count}(\lvert y_i\rvert \leq 3)}{N}\right) \times 100\%$	80.37%	78.52%
% $\leq$ 4 cm	$\left(\frac{\text{count}(\lvert y_i\rvert \leq 4)}{N}\right) \times 100\%$	95.56%	91.48%
% $\leq$ 5 cm	$\left(\frac{\text{count}(\lvert y_i\rvert \leq 5)}{N}\right) \times 100\%$	97.78%	97.41%

Table 7.2 compares detection performance between daytime and nighttime conditions. For the mean-difference method, a clear improvement is observed at night: bias drops from 2.01 cm to 1.00 cm, and precision improves significantly from 2.89 cm to 0.86 cm. The MAE and RMSE follow the same trend, indicating that this method is particularly sensitive to daytime illumination effects, such as increased water transparency and surface reflections.

In contrast, the KS-test method shows nearly identical bias values for day and night (both ≈1.02 cm), supporting its robustness to changing lighting conditions. However, its error spread is far greater during the day, with precision dropping from 6.26 cm to 1.49 cm at night. This suggests that while the KS method is less prone to systematic over- or underestimation across the day–night cycle, it is still susceptible to larger fluctuations in certain conditions, particularly under direct sunlight.

Table 7.2: Comparison of water-level detection performance during daytime and nighttime conditions for both the mean-difference and KS-test method.

Method	Period	N	Bias (cm)	Median (cm)	Precision (cm)	MAE (cm)	RMSE (cm)
Mean-Difference	Day	154	2.01	2.50	2.89	2.68	3.51
Mean-Difference	Night	116	1.00	1.17	0.86	1.02	1.32
KS-Test	Day	154	1.02	1.97	6.26	2.90	6.32
KS-Test	Night	116	1.02	0.64	1.49	1.34	1.80

Table 7.3 compares performance between dry and precipitation periods. For both methods, there is a modest increase in bias and MAE during rain events. In the mean-difference method, bias increases from 1.48 cm to 1.98 cm, while MAE rises from 1.86 cm to 2.44 cm. For the KS-test method, the

increase is more pronounced: bias rises from 0.82 cm to 1.87 cm, while MAE increases from 2.26 cm to 2.12 cm (note the reverse trend in MAE here).

However, these shifts remain within the range of one standard deviation from overall precision and should therefore be interpreted with caution. Additionally, only 51 of the 270 cycles occurred during rainfall, limiting statistical confidence. More field data under diverse and heavier precipitation would be required to reliably quantify performance impacts.

Table 7.3: Summary of detection performance under dry and rainy conditions for both the mean-difference and KS-test method.

Method	Condition	N	Bias (cm)	Median (cm)	Precision (cm)	MAE (cm)	RMSE (cm)
Mean-Difference	No Precip	219	1.48	1.30	2.40	1.86	2.81
Mean-Difference	Precip	51	1.98	2.64	1.82	2.44	2.68
KS-Test	No Precip	219	0.82	1.17	5.27	2.26	5.32
KS-Test	Precip	51	1.87	1.97	1.80	2.12	2.58

7.4. Power assessment

Energy consumption per processing cycle was first characterized under headless operating conditions. A Keweisi USB power monitor *(Otronic.nl)* was connected between the power bank and the Raspberry Pi Zero 2 W to log supply current and voltage, while runtime and CPU active time were recorded using the psutil library.

Over 100 single-cycle runs, the mean-difference method showed an average current draw of approximately 0.79 A at 4.95 V over 20.81 seconds of runtime, resulting in an average power consumption of $\sim$3.91 W and an energy usage of approximately 0.023 Wh per cycle. The KS-test variant drew around 0.84 A at the same voltage for 21.23 seconds of runtime, consuming $\sim$4.14 W on average and 0.024 Wh per cycle (Table 7.4). These results indicate that although the KS-test method consumes about 4–5% more energy and slightly more compute time, both pipelines remain efficient enough for continuous low-power operation.

A second set of tests simulated the expected field duty cycle: one measurement every nine minutes (six cycles per hour). Over one hour, including idle periods, the mean-difference method averaged of 0.552 A at 4.95 V, resulting in a total hourly consumption of approximately 2.73 Wh. The KS-test method consumed slightly more at 0.585 A (2.90 W), as shown in Table 7.5. These values suggest that both algorithms maintain stable and predictable power demand under representative deployment conditions.

Table 7.4: Average processing characteristics per cycle.

Mode	Voltage	Avg. Current	Avg. Duration per Cycle	Avg. Power	Avg. Energy per Cycle
Mean-difference	4.95 V	$\sim$0.79 A	20.81 s	$\sim$3.91 W	$\sim$0.023 Wh
KS-test	4.95 V	$\sim$0.84 A	21.23 s	$\sim$4.14 W	$\sim$0.024 Wh

Table 7.5: Estimated average power consumption over 1 hour of operation.

Mode	Voltage	Avg. Current	Avg. Power
Mean-difference	4.95 V	$\sim$0.552 A	$\sim$2.73 W
KS-test	4.95 V	$\sim$0.585 A	$\sim$2.90 W

The infrared LED was evaluated separately. During its 5-second activation per cycle ($\sim$30 seconds per hour), it drew a peak current of 1.24 A at 4.95 V, corresponding to $\sim$6.13 W during operation. Over one hour, this yields an additional 0.051 Wh, a negligible increment relative to the baseline power draw. Thus, intermittent use of the LED introduces only minor overhead in the total energy budget.

Taken together, these desk-based assessments and simulated field duty cycles demonstrate that the current hardware and software setup is well-suited for short-term autonomous deployment. To meet the six-month energy-independence goal, however, integration of a solar charging system will be essential.

Preliminary estimations indicate that duty-cycling the Raspberry Pi Zero 2 W between measurement cycles could reduce average power consumption from 2.73 W to 0.138 W. A more detailed evaluation of this approach, including limitations and practical considerations, is provided in the discussion (Chapter 8).

8

Discussion

This chapter reflects critically on the performance and design of the developed camera-based water level monitoring system by interpreting field results, comparing them with expectations and existing literature, and identifying key limitations. Section 8.1 explores the observed detection behaviour in the field, highlighting the influences of wind, sunlight, image compression, and spatial resolution on measurement accuracy. Section 8.2 positions the system in relation to conventional hydrological sensors, including radar, ultrasonic, and pressure-based devices, and examines its robustness under diurnal lighting variations and rainfall events. Finally, Section 8.3 discusses design and deployment constraints, including hardware durability, calibration complexity, power autonomy, and data handling, offering insights into areas for improvement and recommendations for future iterations of the system.

8.1. Interpretation of field data

Wind-driven ripples and advective waves on the reference board (observed on the order of ±3 cm) likely drive the observed accuracy shifts, since this noise matches the system's precision limits and therefore defines a practical lower bound on measurement resolution. To mitigate these disturbances, one approach could be burst-image averaging, in which a rapid sequence of frames is captured, and the detected waterline positions are averaged to smooth out peaks caused by individual waves. Alternatively, a rolling-average filter applied to consecutive detections may offer a similar smoothing effect. This however would increase the system response time as it takes averages of multiple detections.

The mean-difference method exhibits a daytime bias of approximately 2.1cm compared with a nighttime bias of around 1.0cm. In contrast, the KS-test method shows smaller diurnal variation, with biases of 1.86cm by day and 1.07cm by night, demonstrating greater resilience to changing illumination. This systematic overestimation during daytime is likely caused by sunlight penetrating the water surface and increasing the apparent depth in the captured image. As a result, a lighter intensity band becomes visible just below the actual waterline on the reference board, which may cause the algorithm to detect the transition too deep. One mitigation technique is to use an infrared-bandpass filter (*uwcamera.nl*) that restricts incoming light to a narrow IR wavelength range. Although such a filter was evaluated during desk tests, it was excluded from field trials to minimize setup complexity; nevertheless, these preliminary results indicate that an IR-bandpass filter could reduce daytime overestimation.

In addition to these optical effects, JPEG compression may also contribute to the misidentification of the waterline. Captured frames are stored in the JPEG format, which introduces compression artifacts by smoothing fine image details and altering local gradients. Since the algorithm detects the waterline based on intensity or colour differences between adjacent pixel regions, even subtle distortion introduced by JPEG compression can shift the perceived transition zone. This is particularly problematic in scenes where the transition is already gradual, such as a sunlit, semi-transparent water surface. These compression artifacts may lead to consistent underestimation of intensity gradients, biasing detection toward lower water levels.

An important consideration in the interpretation of the detection results is the spatial resolution imposed by the camera setup and reference object geometry. In the current field deployment, the pixel-to-length conversion was based on the projected image height of a 500mm wide foam-PVC reference board, positioned at a fixed distance from the camera lens. This setup yielded an estimated vertical resolution of approximately 6.7mm per pixel, defining the smallest detectable step in the waterline position. While this resolution proved sufficient for short-term deployment, it is tied to the physical distance between the camera and the reference surface. Any variation in the effective range (for example, due to mounting constraints at the field site) would alter the pixel scale and affect measurement precision. In general, objects located farther from the lens occupy fewer pixels in the frame, leading to coarser spatial resolution and reduced accuracy. To improve precision, future designs could incorporate either a higher-resolution imaging sensor or a closer mounting position. However, such adjustments may be limited by site-specific installation constraints and trade-offs in field robustness.

Another constraint encountered during testing relates to the limited dynamic range of the current setup. The vertical crop box, as shown in Figure 5.10b, was aligned to the height of the reference board ($\sim$60 cm), meaning that detection was restricted to this narrow vertical window. In operational settings, water levels can vary by several meters, introducing a much broader range of environmental conditions, lighting artifacts, and noise. As this dynamic range increases, so does the likelihood of encountering non-uniform backgrounds and temporary surface disturbances. These factors may raise the system's error floor. Scaling the approach to real-world hydrological dynamics will require rethinking both the spatial extent of the reference structure and the algorithm's capacity to remain robust across a wider vertical detection range.

8.2. Comparison to expectations & literature

8.2.1. Comparison to traditional sensors

The developed camera-based water level monitoring system was designed as a low-cost, low-power alternative to traditional sensor technologies. Compared to radar sensors, which offer sub-centimetre accuracy (typically ±2mm) but come at a cost of $\sim$€1000 [41] and require prominent installation above the water surface, the camera setup sacrifices some precision for significant gains in affordability and resilience. Radar devices, due to their exposed positioning on bridges or elevated structures, are particularly vulnerable to vandalism and direct flood impact [22].

Ultrasonic sensors present a more affordable non-contact solution ($\sim$€100), generally offering accuracies around ±1 cm [23]. However, they are sensitive to surface turbulence [5], temperature variations [6], and wind interference [22], which can degrade performance under natural conditions. More importantly, both ultrasonic and radar sensors require mounting above the water surface—typically on bridges or other overhead structures—where they are exposed to risks such as vandalism and damage during high flood events.

Pressure sensors, commonly installed in submerged housings, offer good accuracy (±1cm) at moderate cost (€300) [30], but are susceptible to damage form extreme events [27], corrosion and flood-related debris [45]. Floater sensors are less expensive ($\sim$€100) [31], but provide only coarse-level or threshold-based detection and are typically unsuitable for continuous environmental monitoring.

In contrast, the system developed in this study achieved 95% accuracy within ±4.64cm, with mean absolute errors of 1.97–2.23 cm depending on the method. While this level of precision does not match that of radar or pressure-based solutions, the trade-off is justified by several advantages: the system is low-cost ($\sim$290€), easily mountable outside flood zones and less visible (less susceptible to vandalism). These characteristics make it especially suitable for decentralized or vandalism-prone settings, where cost and robustness outweigh millimetre-scale accuracy.

8.2.2. Expected diurnal influence versus observed performance

Given the reliance on optical intensity differences, it was anticipated that changing ambient light conditions would influence system accuracy. A previous study [4] observed a decline in accuracy at night, reporting an increase in mean error from 1.8cm during the day to 2.8cm at night, attributed to poor contrast under passive lighting.

In contrast, the present study found that accuracy improved at night. With the use of active infrared illumination, the system maintained stable lighting conditions after sunset. The mean-difference method showed a diurnal bias shift from 2.01cm during the day to 1.00cm at night, while the KS-test method bias values remained 1.02 during both day and night. These findings contradict earlier reports and suggest that active IR lighting can significantly improve nighttime performance, making it more robust than under fluctuating natural daylight conditions. This indicates a design advantage of the current system in handling illumination variability.

8.2.3. Expected influence of precipitation versus observed performance

It was expected that precipitation would reduce detection accuracy, primarily due to occlusion of the reference board by water droplets, splash effects, and increased surface disturbances. This expectation is supported by previous work, which found that heavy rainfall degraded detection accuracy from 1.8cm to 2.6cm during the day, and from 2.8cm to 3.4cm at night [4].

In the present study, performance under precipitation was evaluated for 51 rainy cycles out of a total of 270. Across both the mean-difference and KS-test methods, no consistent increase in bias or MAE was observed. However, visual inspection revealed greater variability in the error signal during rainfall events. Because all rain events occurred during daylight hours (when systematic overestimation was already present due to sunlight penetration) it is not possible to isolate the independent effect of precipitation. Additionally, the limited sample size under rainy conditions constrains the statistical reliability of these findings.

Interestingly, light to moderate rain may actually improve detection under some conditions. Rainfall can increase the diffusivity of the water surface, enhancing the contrast between submerged and dry areas of the reference board, provided the lens remains free from droplets. This may increase the detectability of the waterline. In contrast, during intense rainfall, visibility can deteriorate rapidly, especially when the camera is mounted at a distance. Heavy rain may cause haze, reduce scene contrast, and obscure the reference object. These observations suggest that the effect of precipitation on detection performance is non-linear: while light rain may offer some beneficial contrast effects, heavier rainfall is more likely to degrade image quality.

8.3. Limitations

Prototype modularity versus long-term deployment

The prototype was built for rapid iteration and modular testing rather than extended field service. Components were attached with velcro, adhesive tape, improvised hinges, and plastics and plywood were used without consideration for UV resistance or mechanical fatigue. Although this approach facilitated development and short-term experimentation, it would not withstand the weathering, vibration, and material aging expected in long term deployment.

CLI mode USB-backup incompatibility

During GUI mode testing, a USB flash drive mounted via the Pi's USB port provided reliable data backup. After switching to CLI operation however, mount-point permissions prevented the drive from being accessed automatically. Efforts to resolve the issue were deprioritized in favour of core functionality, leaving the system without a fully autonomous redundancy mechanism when run in its headless mode.

Polarization filter alignment

A circular polarizer was reattached in the field after camera lens calibration to reduce glare, but it was not realigned or calibrated for the new mounting orientation. As a result, specular reflections persisted under low-angle sunlight, causing bright spots that degraded detection accuracy. Future deployments should include a polarization-filter calibration procedure.

Dependence on a reference object

The current algorithm relies on a high-contrast reference board to distinguish the waterline. This dependence increases the risk of vandalism, displacement by high flows, and site-specific installation challenges where a board cannot be mounted. A mitigation strategy could be applying waterproof paint to stable structures like bridge piers that can act as a reference frame. Alternatively, research into AI-based, reference-free detection methods is needed to eliminate this constraint [12][17][40].

Power-system limitations
With the existing 13,400 mAh battery and current software strategy, the system can sustain approximately 26 hours of continuous, autonomous operation. Although a solar-panel-compatible power bank was selected to enable off-grid charging, no panel was deployed during testing. A nano-power timer board *(kiwi-electronics.com)* was prototyped to cut power between measurements and reduce average draw, but it proved unable to reliably re-enable the supply at the end of each rest period. Minor hardware and firmware modifications will be required before dependable power-cycling can be achieved.

The potential benefit of such a system is significant: powering down the Raspberry Pi Zero 2 W during idle periods could reduce hourly energy consumption from 2.73 Wh (current continuous operation) to 0.138 Wh, based on the energy used by six active processing cycles per hour (6 × 0.023 Wh). This yields a theoretical maximum power saving of:

$$\frac{2.73 - 0.138}{2.73} \times 100\% \approx 94.95\%$$

This estimate assumes ideal conditions with instantaneous startup and shutdown, and does not account for boot-up energy peaks or delays. Further empirical testing is needed to assess these factors and determine the practical efficiency gains achievable with a power-cycling strategy.

While solar charging and improved power management remain essential for long-term autonomous deployment, they also introduce new challenges, including risks related to hardware visibility, vandalism, and circuit reliability in variable environmental conditions.

Calibration limitations
Calibration must be repeated whenever the enclosure or camera mount shifts even slightly. At present, realigning the view and obtaining new rotation and crop parameters requires establishing a local connection via a mobile hotspot, which in turn demands physical access to the field site. Two strategies could remove this requirement, though each would result in additional power or computational costs.

First, a secure remote-access interface could be provided by running a lightweight server on the Raspberry Pi. Whenever the detection results begin to drift or on a scheduled basis, an operator could log in remotely, launch the calibration scripts, and update the parameters without visiting the site. However, maintaining an active network service and handling incoming connections would increase both processor usage and standby power draw.

Second, an automatic real-world coordinate feedback loop could be implemented. By calibrating the camera intrinsics and extrinsic, the system could use known scene landmarks to detect any change in orientation. Image processing would estimate the camera's new pose relative to the reference object, compute the corresponding rotation angle and crop box in pixels, and then update the configuration automatically. This approach would require periodic execution of pose-estimation algorithms and matrix transformations, adding to the computational load and further drawing on the battery during each recalibration cycle.

Georeferencing limitations
The developed system reports only pixel changes that reflect the relative position of the water level. To yield absolute z-coordinates, camera calibration would be required. During development, some effort was spent attempting to integrate this capability, but because these techniques are well established, the decision was made not to prioritize them. However, to achieve the system's full potential (supporting flood-early-warning applications) this conversion is necessary and should be explored in future work.

Related to this, the effective centimetre-level accuracy degrades as the distance between the camera and the reference object increases, since fewer pixels span the same physical height. This spatial resolution limitation could be mitigated in part by using a higher-resolution camera module. Additionally, the viewing angle at which the waterline is observed may influence the accuracy, particularly if the line of sight introduces geometric distortion or foreshortening. These factors should be considered in future designs, especially when flexibility in installation height or orientation is required in the field.

Device storage, image posting and connectivity
The prototype currently retains every raw image, rotated and cropped view, and analysis figure per cycle on local flash storage. This was a strategy that suited development but would rapidly exhaust storage space in long-term deployment. Because reliable internet was unavailable at the field site, raw-image uploads were not exercised. Future off-site monitoring will demand a local router or module integrated into the enclosure to push both water-level readings and raw frames to a server. To balance storage and connectivity, only the numerical measurements (including all colour-channel variants) and their corresponding raw images (for model training) should be kept locally, with all intermediate files discarded. A rolling-buffer mechanism on the Pi can enforce a fixed capacity by deleting the oldest files as new data arrive, ensuring backup during temporary network outages while preventing storage overflow.

Automatic outlier rejection
An automated outlier-rejection mechanism could substantially improve detection robustness by filtering questionable waterline estimates. For example, simple boundary-condition checks (such as disallowing changes in water level of more than a few decimetres within a single cycle) would automatically flag and discard implausible readings. More sophisticated rules could e.g. incorporate recent trend information to reject physical impossible detections. Implementing such real-time quality control on the Raspberry Pi would help ensure that only credible measurements are stored and transmitted.

Dynamic measurement frequency
An adaptive sampling strategy could further optimise both data relevance and power consumption by varying the measurement frequency according to observed water-level dynamics. During periods with little change in water level, the system could lengthen the rest interval (thereby conserving battery power) while upon detecting a rapid rise or fall in stage it could temporarily increase the capture rate to provide finer-resolution monitoring. This “event-driven” scheduling could be implemented by comparing successive readings against a threshold or by integrating external rainfall forecasts. Such an approach would align energy expenditure with actual hydrological risk, extending deployment duration without sacrificing critical temporal resolution during flood events.

9

Conclusion

This chapter distils the primary outcomes of the POC, reflecting on its measured accuracy, operational robustness, and cost advantages. It then considers practical lessons learned for real-world deployments and sets the stage for longer-term enhancements and wider adoption.

9.1. Key findings

The prototype water-level monitoring system demonstrated that low-cost, image-based sensing can yield centimetre-level accuracy under real-world conditions. Using a Raspberry Pi Zero 2 W and infrared-assisted imaging, both the mean-difference and KS-test algorithms achieved reliable performance across a multi-day field deployment.

The mean-difference method reached a mean absolute error (MAE) of 1.97cm and a root-mean-square error (RMSE) of 2.79cm, while the KS-test method recorded a slightly higher MAE of 2.23cm and RMSE of 4.92cm. Despite this, the KS-test algorithm demonstrated lower systematic bias (1.02cm vs. 1.57cm) and greater robustness to varying illumination, particularly during daytime. Notably, 95% of errors for both methods fell within ±4.64cm, confirming that the majority of detections stayed within practical error bounds for hydrological monitoring.

Nighttime performance, supported by consistent IR illumination, proved more accurate and stable, with lower median errors and reduced variance. Daytime overestimation was attributed to increased water transparency under direct sunlight, and possibly amplified by JPEG compression artifacts.

In total, 270 detections were analysed. For the mean-difference method, 64.81% of measurements fell within ±2cm and 95.56% within ±4cm. The KS-test method showed similar performance: 65.56% within ±2cm and 91.48% within ±4cm.

The system enclosure withstood light rainfall and moderate field conditions. Power testing indicated that a 13,400mAh battery supported $\sim$26 hours of continuous operation. Potential integration of solar charging and a nano-timer board for hardware-level power management positions the system well for long-term autonomous deployments.

At an estimated total component cost of $\sim$€290, this camera-based solution offers a promising alternative to traditional radar, ultrasonic, pressure, or floater sensors. While it trades some precision (notably under adverse lighting) for lower cost and improved resilience to vandalism and flood damage. The modular, open-source design makes it particularly well suited for budget-constrained or hard-to-access monitoring locations.

9.2. Practical implications

This study demonstrates that compact, camera-based systems offer a viable, low-cost alternative to conventional radar and ultrasonic water-level gauges, particularly in resource-constrained or remote environments. At an estimated cost of ∼€290, the system achieves centimetre-scale accuracy while remaining robust, modular, and energy-efficient. Its discrete mounting position (well above flood-prone areas and out of easy reach) reduces exposure to vandalism and damage, common concerns for traditional sensors installed on bridges or in open water.

Such systems can meaningfully extend the reach of flood early warning and hydrological monitoring networks, especially in areas where conventional infrastructure is impractical or cost prohibitive. Their adaptability also makes them suitable for temporary installations during high-risk periods.

Energy management is a key enabler of long-term autonomous deployment. With a solar-compatible battery pack and plans for a nano-power timer, multi-week or even multi month operation without intervention appears feasible. Optimizing measurement frequency based on hydrological conditions (lengthening sampling intervals during stable flow and shortening them during rapid changes) can further conserve energy while preserving critical data resolution.

To enhance resilience in unattended deployments, the integration of on-board (or possibly centralised) outlier rejection and a circular data buffer is recommended. This would allow the system to discard implausible detections (e.g., sudden unrealistic water level shifts) and manage limited flash storage by automatically overwriting the oldest data. These additions, along with dynamic sampling and solar integration, would move the system closer to a fully autonomous, low-maintenance sensing solution suitable for real-world deployment.

10

Future work

While this proof-of-concept has demonstrated the viability of a low-cost, camera-based water-level gauge, several key areas remain to be addressed before the system can support long-term, autonomous deployments effectively. In particular, the following efforts should be pursued:

1. **Field-scale validation and durability**
    - Conduct multi-month deployments across contrasting sites and seasons to assess environmental robustness.
    - Monitor weathering, UV-degradation and mechanical fatigue of enclosure and mounting hardware.
2. **Camera calibration for absolute elevations**
    - Implement camera intrinsic/extrinsic calibration and use known reference points to convert pixel heights into real-world z-coordinates.
    - Automate pose-estimation routines to update rotation and crop parameters without manual intervention.
3. **Reference-free and ML-based detection**
    - Build a labeled image library (batch of raw frames under varied illumination, turbidity, vegetation cover) for training deep-learning models to detect waterlines without a physical board [11].
    - Evaluate state-of-the-art segmentation or edge detection networks for waterline detection.
4. **Advanced power management & solar integration**
    - Finalize integration of the nano-power-timer to reliably power-cycle the Pi between measurements.
    - Pair the optimized timer with a small solar panel, then field-test energy self-sufficiency over long term deployment.
5. **Storage, connectivity & data handling**
    - Develop an on-device rolling-buffer scheme to retain only raw images and numeric measurements, deleting intermediate files when capacity is reached.
    - Integrate an internet modem to upload both water-level data and selected raw frames in near-real time.
6. **Dynamic sampling & automatic outlier rejection**
    - Implement adaptive sampling intervals (e.g. lower frequency during low risk periods, higher when rapid changes detected).
    - Embed statistical or rule-based filters (e.g. maximum plausible water-level change per interval) to automatically discard questionable detections.

References

[1] Agile Business Consortium. *MoSCoW Prioritisation*. Accessed: 2025-06-10. 2025. URL: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html.

[2] AmebaIoT. *PowerMode – Deep Sleep Mode*. Accessed: 2025-06-02. 2021. URL: https://www.amebaiot.com/en/amebapro2-arduino-deepsleep/.

[3] Jean-Luc Aufranc. *A deep dive into Raspberry Pi Zero 2 W's power consumption*. Accessed: 2025-06-02. 2021. URL: https://www.cnx-software.com/2021/12/09/raspberry-pi-zero-2-w-power-consumption/.

[4] Joaquim Amândio Azevedo and João André Brás. "Measurement of Water Level in Urban Streams under Bad Weather Conditions". In: *Sensors* (2021). ISSN: 1424-8220. DOI: 10.3390/s21217157. URL: https://www.mdpi.com/1424-8220/21/21/7157.

[5] Inhyeok Bae and Un Ji. "Outlier detection and smoothing process for water level data measured by ultrasonic sensor in stream flows". In: *Water (Switzerland)* 11.5 (2019). ISSN: 20734441. DOI: 10.3390/w11050951.

[6] Ganggang Bai et al. "An intelligent water level monitoring method based on SSD algorithm". In: *Measurement: Journal of the International Measurement Confederation* 185 (2021). ISSN: 02632241. DOI: 10.1016/j.measurement.2021.110047.

[7] Bscan. *KS Example*. https://commons.wikimedia.org/wiki/File:KS_Example.png. CC0 license, via Wikimedia Commons. Accessed: 2025-06-02. 2013.

[8] Chen Chen et al. "An Integrated Method for River Water Level Recognition from Surveillance Images Using Convolution Neural Networks". In: *Remote Sensing* 14.23 (2022). ISSN: 20724292. DOI: 10.3390/rs14236023.

[9] Liz Clark. *Adafruit MOSFET Driver*. https://cdn-learn.adafruit.com/downloads/pdf/adafruit-mosfet-driver.pdf. Accessed: 2025-06-02. This guide details the Adafruit STEMMA MOSFET Driver, featuring an AO3406 N-Channel MOSFET rated for 30Vds and 3.6A peak, suitable for driving motors, solenoids, and LEDs. 2024.

[10] Elektor. *Raspberry Pi NoIR Camera Module v2*. https://www.elektor.com/products/raspberry-pi-noir-camera-module-v2. Accessed: 2025-06-02. 2025.

[11] Anette Eltner et al. "Automatic Image-Based Water Stage Measurement for Long-Term Observations in Ungauged Catchments". In: *Water Resources Research* 54.12 (2018). ISSN: 19447973. DOI: 10.1029/2018WR023913.

[12] Anette Eltner et al. "Using Deep Learning for Automatic Water Stage Measurements". In: *Water Resources Research* 57.3 (2021). ISSN: 19447973. DOI: 10.1029/2020WR027608.

[13] Simon Etter et al. "Quality and timing of crowd-based water level class observations". In: *Hydrological Processes* 34.22 (2020). ISSN: 10991085. DOI: 10.1002/hyp.13864.

[14] Farnell. *IR Emitter Module Datasheet*. https://www.farnell.com/datasheets/3164632.pdf. Accessed: 2025-06-02. 2023.

[15] Francisco E. Fernandes, Luis Gustavo Nonato, and Jó Ueyama. "A river flooding detection system based on deep learning and computer vision". In: *Multimedia Tools and Applications* 81.28 (2022). ISSN: 15737721. DOI: 10.1007/s11042-022-12813-3.

[16] Intelligent LED Solutions. *ILS Aluminium Alloy Heatsinks for PowerStars and PowerClusters*. https://www.farnell.com/datasheets/3161532.pdf. Accessed: 2025-06-02. Includes models such as ILA-HSINK-STAR-50X20MM, ILA-HSINK-STAR-50X40MM, ILA-HSINK-STAR-50X60MM, ILA-HSINK-STAR-50X80MM, ILA-HSINK-70X70X55MM, and ILA-HSINK-78X46X25MM. Supplied with fixing screws and pre-applied Thermal Interface Material (TIM). 2019.

[17] Navid H. Jafari et al. “Real-time water level monitoring using live cameras and computer vision techniques”. In: *Computers and Geosciences* 147 (2021). ISSN: 00983004. DOI: 10.1016/j.cageo.2020.104642.

[18] Frans Klijn et al. “Adaptive flood risk management planning based on a comprehensive flood risk conceptualisation”. In: *Mitigation and Adaptation Strategies for Global Change* 20.6 (2015). ISSN: 15731596. DOI: 10.1007/s11027-015-9638-z.

[19] C. J. Koblinsky et al. “Measurement of river level variations with satellite altimetry”. In: *Water Resources Research* 29.6 (1993). ISSN: 19447973. DOI: 10.1029/93WR00542.

[20] Lung Chih Kuo and Cheng Chi Tai. “Automatic water-level measurement system for confined-space applications”. In: *Review of Scientific Instruments* 92.8 (2021). ISSN: 10897623. DOI: 10.1063/5.0046804.

[21] Yan Ting Lin, Yi Chun Lin, and Jen Yu Han. “Automatic water-level detection using single-camera images with varied poses”. In: *Measurement: Journal of the International Measurement Confederation* 127 (2018). ISSN: 02632241. DOI: 10.1016/j.measurement.2018.05.100.

[22] Wen Cheng Liu and Wei Che Huang. “Evaluation of deep learning computer vision for water level measurements in rivers”. In: *Heliyon* 10.4 (2024). ISSN: 24058440. DOI: 10.1016/j.heliyon.2024.e25989.

[23] MaxBotix Inc. *MB7052 XL-MaxSonar-WRMAT Ultrasonic Sensor*. Accessed: 2025-06-01. 2024. URL: https://maxbotix.com/products/mb7052.

[24] Multicomp Pro. *Wire Wound Fixed Resistors Datasheet*. https://www.farnell.com/datasheets/3916833.pdf. Accessed: 2025-06-02. The datasheet covers various models including MCKNP series resistors with power ratings from 0.5W to 10W, resistance values ranging from 0.1Ω to 50kΩ, and tolerances of ±1% and ±5%. Features include flame-resistant coating, non-inductive options, and compliance with E24 and E96 series standards. 2023.

[25] OpenRiverCam. *OpenRiverCam Platform*. Accessed: 2025-06-02. 2025. URL: https://openrivercam.com/.

[26] Jinqiu Pan et al. “Deep learning-based unmanned surveillance systems for observing water levels”. In: *IEEE Access* 6 (2018). ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2883702.

[27] Jonathan D. Paul, Wouter Buytaert, and Neeraj Sah. “A Technical Evaluation of Lidar-Based Measurement of River Water Levels”. In: *Water Resources Research* 56.4 (2020). ISSN: 19447973. DOI: 10.1029/2019WR026810.

[28] Raspberry Pi Foundation. *Raspberry Pi NoIR Camera Module v2*. https://www.raspberrypi.com/products/pi-noir-camera-v2/. Part number: RPI NOIR CAMERA BOARD. 8 MP Sony IMX219 image sensor. Supports 1080p30, 720p60, and 640x480p90 video. 2023.

[29] Raspberry Pi Ltd. *Raspberry Pi Zero 2 W Product Brief*. https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf. Accessed: 2025-06-02. 2024.

[30] RS Components. *Capacitive Liquid Level Sensor for Non-metallic Containers, RS Stock No. 2726967*. Accessed: 2025-06-01. 2024. URL: https://nl.rs-online.com/web/p/level-sensors/2726967.

[31] RS Online. *Float Level Sensor, Horizontal, Plastic Stem*. Accessed: 2025-06-01. 2024. URL: https://nl.rs-online.com/web/p/level-sensors/2726967.

[32] Schneider Electric. *Thalassa TBS ABS Enclosure NSYTBS191610H*. https://www.se.com/nl/nl/product/NSYTBS191610H/tbs-tbp-doos-abs-+-deksel-193x164x105mm-ip66-ik07-h40-ral7035/. Accessed: 2025-06-02. ABS enclosure with dimensions 193x164x105 mm, IP66, IK07, RAL7035, suitable for outdoor use. 2025.

[33] Gerhard Schoener. “Time-Lapse Photography: Low-Cost, Low-Tech Alternative for Monitoring Flow Depth”. In: *Journal of Hydrologic Engineering* 23.2 (2018). ISSN: 1084-0699. DOI: 10.1061/(asce)he.1943-5584.0001616.

[34] Lorenzo Steccanella et al. “Deep learning waterline detection for low-cost autonomous boats”. In: *Advances in Intelligent Systems and Computing*. Vol. 867. 2019. DOI: 10.1007/978-3-030-01370-7{_}48.

[35] Wenchao Sun, Hiroshi Ishidaira, and Satish Bastola. “Calibration of hydrological models in ungauged basins based on satellite radar altimetry observations of river water level”. In: *Hydrological Processes* 26.23 (2012). ISSN: 08856087. DOI: 10.1002/hyp.8429.

[36] M. J. Tourian et al. “Spatiotemporal densification of river water level time series by multimission satellite altimetry”. In: *Water Resources Research* 52.2 (2016). ISSN: 19447973. DOI: 10.1002/2015WR017654.

[37] UwCamera.nl. *37mm MRC CPL Filter (16 Lagen) / Waterafstotend Multi-Coated CPL Lensfilter*. https://www.uwcamera.nl/37mm-MRC-CPL-Filter-(16-lagen)-/-Waterafstotend-Multi-Coated-CPL-Lensfilter. Accessed: 2025-06-02. Features a 16-layer multi-resistant coating that reduces reflections and enhances color saturation. The hydrophobic coating repels water, oil, and dust, making it suitable for outdoor photography. The filter is rotatable to adjust polarization effects and is compatible with lenses having a 37mm thread. 2025.

[38] Annemiek van Boeijen et al. *Delft Design Guide: Design strategies and methods*. English. BIS Publishers, 2013.

[39] Remy Vandaele, Sarah L. Dance, and Varun Ojha. “Deep learning for automated river-level monitoring through river-camera images: An approach based on water segmentation and transfer learning”. In: *Hydrology and Earth System Sciences* 25.8 (2021). ISSN: 16077938. DOI: 10.5194/hess-25-4435-2021.

[40] Ryan L. Vanden Boomen, Zeyun Yu, and Qian Liao. “Application of Deep Learning for Image-Based Stream Gaging”. In: *Water Resources Research* 57.11 (2021). ISSN: 19447973. DOI: 10.1029/2021WR029980.

[41] VEGA Grieshaber KG. *VEGAPULS C 23 Radar Level Sensor*. Accessed: 2025-06-01. 2024. URL: https://www.vega.com/en-us/products/product-catalog/level/radar/vegapuls-c-23.

[42] Visual Crossing Corporation. *Weather Query Builder*. https://www.visualcrossing.com/weather-query-builder/. Accessed: 2025-06-02. 2025.

[43] Voltaic Systems. *6 Watt 6 Volt Solar Panel (P106)*. https://voltaicsystems.com/6-watt-panel/. Accessed: 2025-06-02. Features: IP67 waterproof rating, UV-resistant urethane coating, 19% efficient monocrystalline cells, 6.0W peak power at 6.5V, 930mA peak current, dimensions: 17.5 x 22.1 x 0.5 cm, weight: 255g. 2025.

[44] Voltaic Systems. *V50 USB Battery Pack*. https://voltaicsystems.com/v50/. Accessed: 2025-06-02. 13,400mAh capacity, 48Wh Li-Ion battery with Always On mode, dual USB-A outputs, USB-C PD input/output, optimized for solar charging. 2025.

[45] Sheng Wei Wang et al. “A continuous water-level sensor based on load cell and floating pipe”. In: *Proceedings of 4th IEEE International Conference on Applied System Innovation 2018, ICASI 2018*. 2018. DOI: 10.1109/ICASI.2018.8394554.

[46] W. van de Westeringh. *Soils and their Geology in the Geul Valley*. Tech. rep. 80-8. Accessed: 2025-06-02. Landbouwhogeschool Wageningen, 1980. URL: https://edepot.wur.nl/467976.

[47] Fan Wu, Shih-Wen Hsiao, and Peng Lu. “An AIGC-empowered methodology to product color matching design”. In: *Displays* 81 (2024), p. 102623. ISSN: 0141-9382. DOI: https://doi.org/10.1016/j.displa.2023.102623. URL: https://www.sciencedirect.com/science/article/pii/S0141938223002573.

[48] Di Zhang and Junyan Tong. “Robust water level measurement method based on computer vision”. In: *Journal of Hydrology* 620 (2023). ISSN: 00221694. DOI: 10.1016/j.jhydrol.2023.129456.

A

Design Methodology

This section introduces the design methodology and theoretical tools that structured the development of the prototype. Tools include the Basic Design Cycle, function analysis, morphological charting, and SCAMPER, as described in the Delft Design Guide [38].

A.1. Design Perspective

The approach for the development process has been shaped by the methods outlined in the Delft Design Guide [38], with particular emphasis on the principle of Design for the Majority. This perspective advocates for developing products that are accessible, relevant, and sustainable for people at the Base of the Pyramid (BoP), the majority of the world's population who often live in economically disadvantaged conditions.

Most existing water monitoring technologies are designed with users at the Top of the Pyramid (ToP) in mind, what relates to wealthier regions with robust infrastructure, technical support, and high financial capacity. These technologies are often too costly, maintenance-intensive, or complex for deployment in low-resource settings. As a result, they fail to meet the needs of local water management professionals in developing areas.

This research specifically adopts a BoP design mindset to ensure that the proposed water level monitoring system is affordable, accessible, available, reliable, sustainable, and culturally acceptable. Key principles influencing the design include:

- **Affordability**: The product must be cost-effective, with minimal reliance on patented technology.
- **Accessibility**: The system should be deployable in remote or under-resourced environments.
- **Availability**: Materials and skills required to build or maintain the system should be locally accessible.
- **Reliability**: The system must operate autonomously with minimal maintenance, supporting community-led usage and repair.
- **Sustainability**: Designs should have low environmental impact, for example through self-sufficient power sources.
- **Acceptability**: Local values, working practices, and preferences must be acknowledged and integrated.

The design perspective serves as a tool to focus the development process on end-user needs. In this research, the primary end users are water management officers operating in low-resource regions, where reliable, low-cost monitoring systems are lacking. However, since the outcome of this project is a proof-of-concept through a minimal viable product (MVP), it will serve as a starting point for future iterations and refinements. Therefore, the project supervisors are also considered interim users during this development phase. Their technical and contextual expertise plays a pivotal role in aligning the

development process with real-world applicability and guiding design decisions when direct user input from the target region is not available.

In addition to the design perspective, a function analysis was performed to map the system's purpose within its context. This analysis structured the translation of user needs into concrete system requirements and supported the development of the requirements list outlined in Section A.4.

A.2. Function Analysis

To complement the design perspective and ensure that all essential functions of the system are logically connected to its components, a Function Analysis was conducted. This method supports the development of both new and existing products by identifying all system functions and linking them to physical or conceptual elements, referred to as ”organs” [38].

This approach encourages a problem-solving mindset by abstracting the system's objectives into functional blocks, helping to focus on ”what the system should do” rather than prematurely locking in design choices.

Figure A.1 illustrates the high-level functional decomposition for the real-time water level monitoring system developed in this project.

The function tree begins with the overarching purpose of the system and breaks it down into five key subsystems:

- Sensor system, responsible for acquiring data from the environment
- Processing system, which performs image analysis and stores data
- Communication system, enabling remote access to measurements
- Power system, designed for off-grid, autonomous operation
- Casing / General design, addressing physical robustness, cost-effectiveness, and maintenance simplicity

Each of these subsystems is further decomposed into specific functional requirements, such as operating during day- and nighttime, data transmission via Wi-Fi and low-frequency maintenance. This analysis helped ensure that each design decision was tied to a clearly identified system function, and it provided a reference framework for validating the prototype during field testing.

The function analysis served not only to clarify design objectives but also supported the creation of the system requirements (Section A.4) and informed the morphological chart and component selection process described later in this chapter.

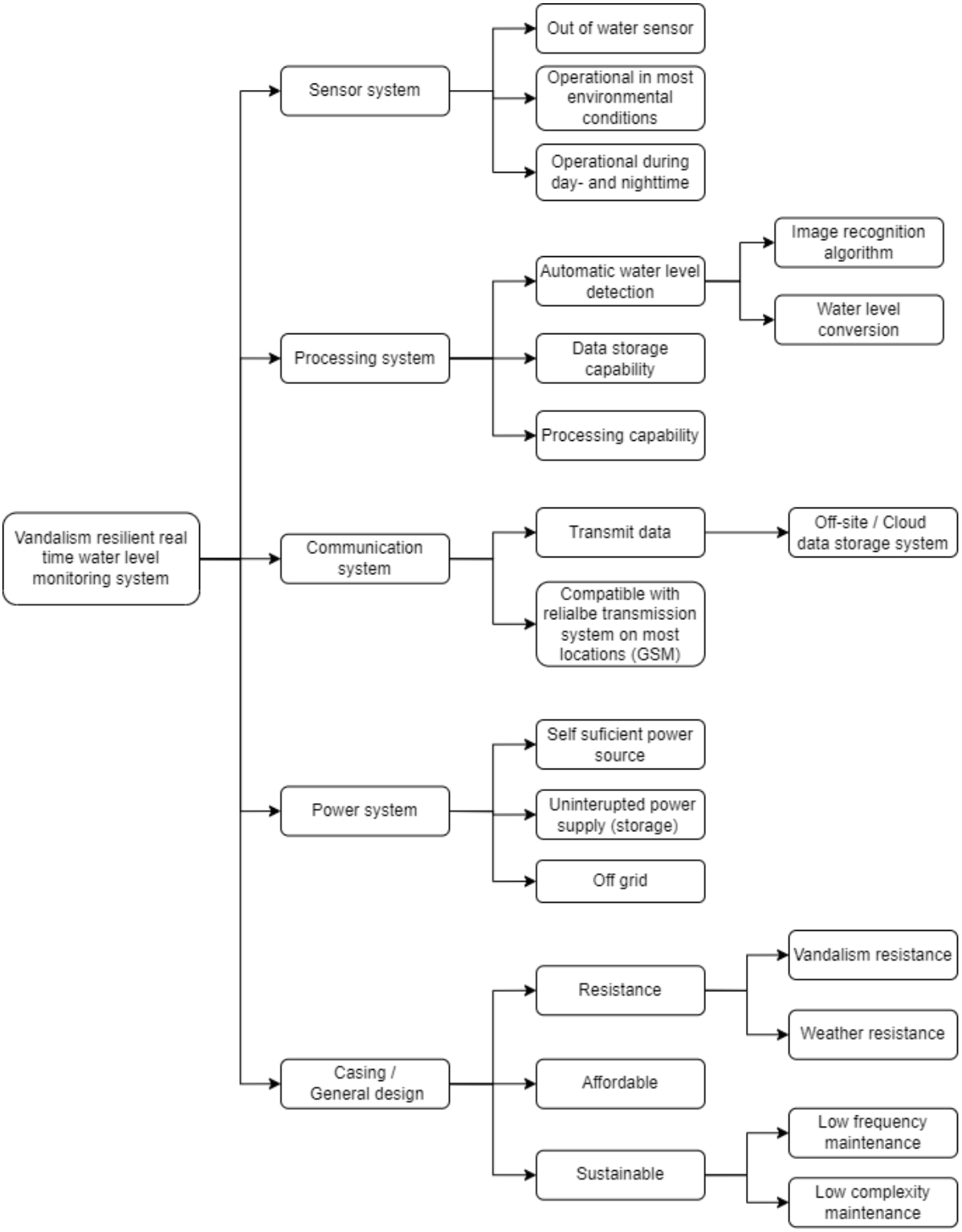


Figure A.1: Function analysis

A.3. Basic Design Cycle

The development process of this research project followed the principles of the Basic Design Cycle, as described in the Delft Design Guide (Boeijen and Daalhuizen, 2013). This model outlines the core reasoning steps in design and promotes an iterative approach, helping designers refine both their understanding of the problem and the development of viable solutions over time.

The cycle consists of the following five key phases:

- **Analysis** – The problem context is studied in depth to clarify user needs, constraints, and key design requirements. This stage provided the foundation for the Function Analysis (Section A.2) and the List of Requirements (Section A.4).
- **Synthesis** – Based on the insights from the analysis, potential solutions and configurations are generated. These are structured using methods such as the Morphological Chart (Section A.5).
- **Simulation** – Early concepts are simulated through low-complexity prototyping, desk testing and reasoning-based evaluations. These simulations help identify weak points and guide refinement before costly iterations.
- **Evaluation** – Prototypes are evaluated based on their performance against predefined criteria (e.g., water level accuracy, processing speed, power efficiency).
- **Decision** – The outcome of the evaluation stage determines whether the solution is accepted, adjusted, or returned to an earlier step in the cycle for further refinement.

These steps are not followed strictly in a linear order but are revisited as needed. Iteration is a core aspect of this model, supporting a deeper alignment between user needs and technical feasibility over time.

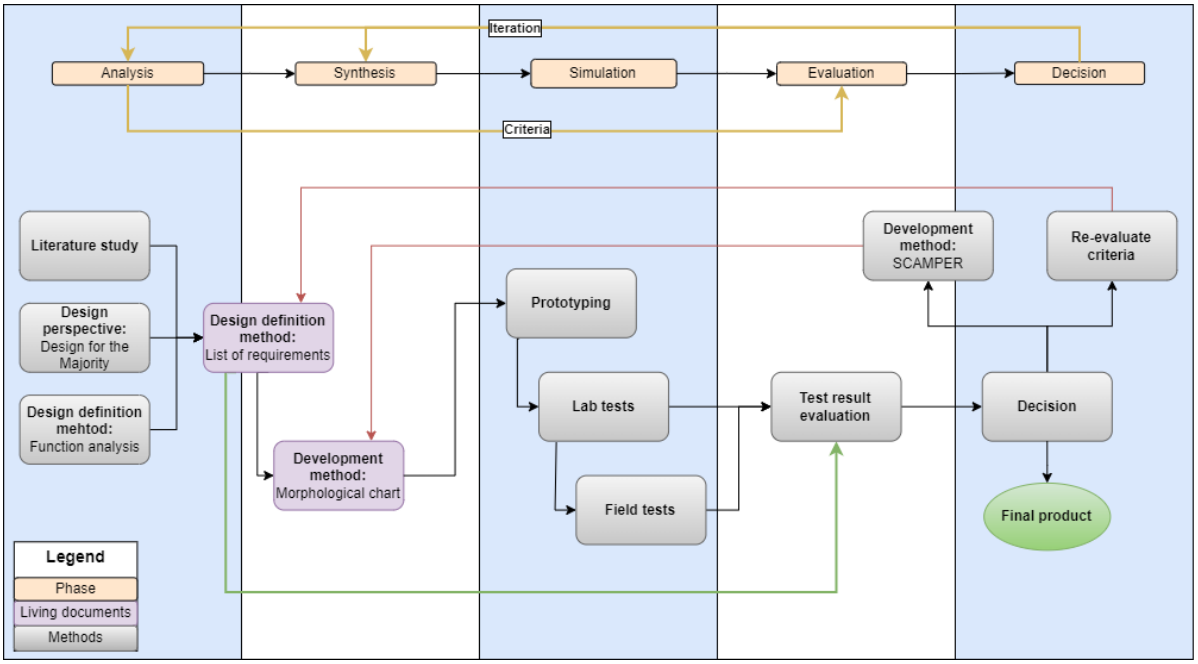


Figure A.2: Application of the Basic Design Cycle in this project.

As shown, the cycle is structured around core design phases (orange), supported by living documents such as the List of Requirements and the Morphological chart (purple), and driven by key design methods (grey). The iterative nature of the process is highlighted by arrows looping between testing, evaluation, and refinement, ensuring that the final prototype reflects both the system requirements, and the user-focused design perspective introduced earlier.

A.4. List of Requirements

To translate user needs and environmental constraints into concrete, testable conditions, a structured List of Requirements (LoR) was created. This method, outlined in the Delft Design Guide [38], plays a central role in anchoring the design process and preventing inconsistent decisions.

The LoR serves as a living document, which acts as both a boundary-setting tool during concept development and a reference point for prototype validation. In the context of this research, the list was directly informed by:

- The design perspective (Section A.1)
- The outcomes of the function analysis (Figure A.1)
- Feedback and scoping sessions with the project team

The list distinguishes between demands (must be met) and wishes (desirable but non-essential) using the MoSCoW prioritization framework:

- **Must-have**: critical for functionality
- **Should-have**: strongly desirable
- **Could-have**: optional improvements
- **Won't-have (for now)**: future scope

The full list of requirements is detailed in this section, and many are explicitly revisited in the performance evaluation and results analysis (Annex C).

The following design requirements were established in cooperation with the project team and guided the development of the prototype water-level monitoring system. These criteria reflect both the functional needs and operational constraints of deploying a robust, low-cost, off-grid system in flood-prone, remote regions such as Kenya. Where applicable, requirements will be validated through field testing, while others are scoped for future development or longer-term operation.

To structure and prioritize the system design requirements, the MoSCoW method was employed. This method categorizes requirements into four groups — Must-have, Should-have, Could-have, and Won't-have (for now) — based on their criticality to the successful functioning of the prototype:

- **Must-have (M)** requirements are essential for the system to function as a valid proof-of-concept. These requirements define the core capabilities without which the device would fail to meet its primary objectives.
- **Should-have (S)** requirements are important but not vital. While not essential for basic operation, they add significant value and improve performance or robustness.
- **Could-have (C)** requirements are desirable but non-essential features. These elements are not strictly required for the prototype, but their development could enhance the system's usability, resilience, or long-term sustainability.
- **Won't-have (W)** requirements are acknowledged but excluded from the current scope due to time, technical, or budgetary constraints. Many of these are identified as potentials for future work, to be explored in follow-up studies or as part of system refinement beyond this proof-of-concept phase.

This prioritization approach allows the project team to focus on delivering a functional and testable prototype within the available resources and timeframe, while also laying the groundwork for future development directions.

Source: Agile Business Consortium [1]

ID	Type	MoSCoW	Description
1	Flood protection	M	The device must remain fully operational and stay dry during floods with a return period of 1/100 years or smaller. The environmental conditions for these events should be verified through a review of site-specific flood risk assessment during the testing phase.
2	Measurement availability	M	Measured water level data must be processed and made available to users within 5 minutes of data collection.
3	Continuous measurement	M	The device must be capable of performing measurements and processing at a minimum frequency of once every 5 minutes under extreme operational conditions, ensuring accurate data collection during critical events.
4	Cost effectiveness	M	The total material cost of the device must not exceed €400, including all components, as confirmed through cost analysis.
5	All-condition measurement	M	The device must measure water levels accurately during both day and night, regardless of ambient light conditions, as verified in a range of lighting scenarios during the testing phase.
6	Water level detection accuracy	S	The device must detect water levels with an accuracy of ±2 cm, as validated during field experiments during the testing phase.
7	Secure installation	S	The device must be installed securely, ensuring the sensor does not shift from its designated frame position for a minimum of 6 months, even under external disturbances.
8	Weather resistance	S	The device must withstand all extreme weather conditions (including heat, wind, and rain) prevalent in Kenya, with a return period of <1/100 years, as demonstrated through environmental stress tests (environmental stress tests not within scope of this project?).
9	Vandalism resistance	S	The device must incorporate design elements that deter and resist vandalism, with specific vandal-proofing measures validated during testing, and final criteria determined during the testing phase.
10	Energy independence	C	The device must operate independently (using power storage or a combination of power generation and storage) for at least 6 months without requiring external intervention.
11	Data storage backup	C	The device must be able to store measured data for up to 48 hours during connection interruptions, ensuring no loss of critical measurements.
12	Data accuracy filtering	W	The system must filter or flag statistical outliers before transmitting data, with criteria for outliers to be defined and finalized during the testing phase.
13	Maintenance-free operation	W	The device must function without any maintenance for a continuous period of at least 6 months, as validated during operational testing (operational testing not within scope of this project?).

Table A.1: List of prioritized system requirements based on the MoSCoW method.

A.5. Morphological Chart

To explore a wide range of conceptual solutions, a Morphological Chart was used. This method, described in the Delft Design Guide (Boeijen and Daalhuizen, 2013), is especially useful in early-stage design when multiple functional requirements need to be addressed simultaneously.

The Morphological Chart breaks down the system's main function (to deliver a vandalism-resilient, real-time water level monitoring system)[1] into multiple subfunctions, such as flood protection, measurement accuracy, power supply, and vandalism resistance. For each subfunction, a list of alternative technical solutions was generated and visualized in a matrix format.

Steps followed in creating the chart:

1. Define the main function and subfunctions. These functions are derived from the Function Analysis (Section A.2).
2. List solution options. Ideas were generated from literature, team brainstorming, and BoP-oriented design principles.
3. Evaluate feasibility. Solutions were assessed based on relevance to the List of Requirements (Section A.4), resource constraints, and alignment with low-power, low-cost principles.
4. Select a preferred combination. A configuration of components was selected to form the initial prototype, with additional paths marked for future iteration.

Cells highlighted in green represent the options included in the current prototype. Those unhighlighted represent unimplemented ideas. This visualization not only illustrates the decision-making process but also demonstrates the modularity of the design. Multiple components can be substituted or improved independently without altering the full system architecture.

The chart also served as a living document during testing and was revisited during the SCAMPER-based evaluation phase (Section A.6).

MoSCoW	Criterion	Option 1	Option 2	Option 3	Option 4	Option 5
M	1. Flood Protection	Away from river	Above river (bridge/stilts)	Floating in river	Submersible with sealed casing	
M	2. Water Level Detection	High resolution camera	Zoom lens	Close to river/gauge		
M	3. Measurement Availability	High processing power	Low computational algorithm	Small datatype upload format		
M	4. Continuous Measurement	Event-triggered measurement	Duty-cycling	Dynamic threshold (historical)	Machine learning anomaly detection	
C	5. Data Accuracy Filtering	Integration of filtering workflow	Post-transmission correction			
S	6. Energy Independence	Power storage capacity	Power generation (solar/wind/hydro)	Storage + Generation	Low power demand	
W	7. Maintenance-Free Operation	High lifetime materials	Adaptive algorithm	Operational via online connection	Automatic lens cleaning	Debris shielding for sensor
S	8. Secure Installation	Secured to manmade structure	Secured on tree	Secured out of reach		
M	9. Weather Resistance	Waterproof	Integrated fan (lens condensation)	Impact-resistant casing (wind/storm)	Compact wind-resistant design	Heat-resistant electronics
W	10. Data Storage Backup	Large SD card (processing+storage)	Additional SD card	Local-to-cloud sync	On-device ring buffer storage	
S	11. Cost Effectiveness	Optimized modular design				
S	12. All-condition Measurement	NoIR lens + IR Light	IR camera	Flashlight	Thermal night vision	
M	13. Vandalism Resistance	Camouflage	Hidden design	Discouraging (beehive/police)	Installed in monitored areas	Compact

Included in current prototype | **Not included** in current prototype

Figure A.3: Morphological chart

[1] Decide on including vandalism resistance in project or not.

A.6. SCAMPER Method

To guide idea generation and concept refinement throughout the design process, the SCAMPER method was employed. This creative thinking technique is based on seven structured heuristics, each prompting a specific question to stimulate innovation or identify opportunities for improvement. It was particularly useful during the evaluation and testing phases, where results revealed areas for improvement or simplification.

The SCAMPER framework includes the following heuristics [38]:

- **Substitute** – What materials, components, or technologies could be replaced with simpler, cheaper, or more sustainable alternatives?
- **Combine** – Are there functions or parts that could be integrated to reduce complexity or improve efficiency?
- **Adapt** – How could elements from other systems or contexts be adapted to suit this design?
- **Modify** – What modifications (e.g., in shape, scale, performance) could improve usability or functionality?
- **Put to Another Use** – Could existing components or algorithms serve secondary purposes (e.g., environmental monitoring)?
- **Eliminate** – Which features or parts can be simplified, reduced, or removed to lower cost or maintenance?
- **Reverse** – Could processes be re-ordered or reversed to improve reliability or reduce resource use?

The SCAMPER method was applied iteratively after each testing round to reflect critically on the prototype's performance against the List of Requirements (Section A.4) and adjust system configurations where appropriate. For example, under the Substitute and Reverse heuristics, the original 5V power system was replaced with a 12V configuration to enable the use of a higher-powered 12V LED for improved nighttime visibility. This was evaluated alongside a 5V LED alternative. However, after testing, the system was reverted to the original 5V setup. This decision was based on the need to better align with the system's low-power design goals and overall energy efficiency requirements, as outlined in the criteria for off-grid operation.

Together, the design methodologies discussed in this section formed the foundation for a structured, iterative, and user-centered development process. The discussed methods ensured that both the conceptual framework and practical constraints were considered. The main report describes how these theoretical foundations were translated into a physical prototype through hardware development and testing.

The actual SCAMPER results can be found in the development log (Annex G).

B

Design choices

B.1. Operating system Raspberry Pi

The design of the water level detection system was influenced by insights gained from the literature review. Existing systems typically either relied on high-computational deep learning approaches, or simpler wildlife cameras that stored raw images without real-time processing. To fill this gap, this research deliberately focused on creating a lightweight, autonomous algorithm that could run efficiently on a low-power device such as the Raspberry Pi Zero 2 W.

Alternative hardware options, such as the Realtek AMB82 Mini, were initially explored. However, due to the researcher's greater familiarity with Python environments and the need to develop a working Prove Of Concept (POC) within a limited timeframe, the decision was made to build the system around the Raspberry Pi platform.

Throughout the development process, algorithmic choices were guided by the need to minimize memory usage, processing demand, and energy consumption. Deep learning models were avoided, as their computational intensity would have conflicted with the low-power and low-cost objectives.

B.2. Reference object

The reference object underwent several design iterations during development. The initial prototype consisted of a small foam panel affixed to a shovel, which proved too unstable for consistent imaging and served only to validate the basic camera functionality. In the second iteration, a hollow metal rail painted white and marked at 10 cm intervals was used; a mounting handle allowed attachment to riverbanks or concrete structures (as tested at TU Delft gutter). Although this rod performed well for early algorithm versions, its narrow profile yielded only a few pixels in width when imaged at greater distances, reducing detection reliability. To address this, the final reference object was constructed from a large matte PVC board, providing sufficient pixel area for each intensity box and thereby smoothing the computed difference signals and reducing noise. Other candidate materials evaluated during this process are listed below.

	Material	IR Reflection (850 nm)	Waterline Visibility	Tape / Paint Adhesion	Water Resistance	Weight / Stability	Cost	Remarks
	Stainless steel pipe	Good (diffuse)	Good (if matte/brushed)	Good	Excellent	High (stable)	High	Durable, industrial-grade
	Aluminium pipe	Very good (specular)	May be worse (glossy)	Moderate – primer recommended	Good	Low to moderate	Medium	Lightweight, but risk of IR glare
	White PVC pipe/sheet	Good (diffuse)	Good (clear contrast)	Very good	Good	Low	Low	Budget option, easy to work with
	Retroreflective tape (3M/Avery)	Very good (IR-specific)	Reflection under water unclear	Self-adhesive, primer may help	Good above water	n/a	Medium	Use only above water for IR detection
	Aluminium foil / metal tape	Very strong (glare)	Poor (glossy)	Poor	Moderate	n/a	Low	Not recommended, except for temporary use
	Matte black tape	Poor (low reflection)	Excellent contrast with tape	Good	Depends on type	n/a	Low	Ideal for markings
	Concrete + white paint	Good (depends on paint)	Good (sharp transition via absorption)	Good with primer on rough concrete	Very good (immersible)	n/a	Low–medium	Ideal for fixed setups; use matte white or epoxy paint

Figure B.1: Design considerations reference object.

B.3. Algorithm development

Over the course of five prototype iterations, the water-level detection algorithm evolved from a simple global-threshold approach to a robust, multi-mode statistical pipeline, with each stage informed by systematic feedback loops between hardware and software.

In the first prototype, Otsu's method was employed to segment an improvised foam staff gauge from its background. Although this yielded a clear binarization in some frames, it proved unstable under changing lighting and reflections; large bright regions such as sky and wet concrete were often misclassified, and underwater portions of the gauge confused the thresholding.

In the second prototype, the refined Otsu workflow added component-size filtering and tight cropping around the gauge. Yet noise persisted, runtime remained high, and manual cropping imposed a significant setup burden. These results motivated moving away from whole-object segmentation toward methods that directly target the waterline's characteristic intensity jump.

In Prototype 3, a "box-differencing" algorithm was implemented: two fixed-height boxes slide down the vertically aligned gauge, computing the difference in mean pixel intensity at each position. After smoothing the resulting profile and suppressing edge peaks, the highest local maximum reliably indicated the waterline. Although this approach markedly outperformed Otsu's method, exploration of a continuous image-subtraction (CIS) variant revealed that rapid-burst capture was beyond the Pi's capacity and that CIS suffered from noise due to vegetation and lighting shifts. Manual cropping remained an obstacle, however, underscoring the need for automated region-of-interest definitions.

Prototype 4 broadened the pipeline to four detection modes (raw intensity, HSV value, hue, and saturation) and introduced a Kolmogorov–Smirnov (KS) statistic as an alternative to mean-difference. For each mode, differences were smoothed, normalized into a probability distribution, and peak-scored, with the highest-scoring mode selected per image. While the KS method consumed more energy, it proved more resilient to glare.

By Prototype 5, the best detection routines were refactored into modular Python packages (wd__capture, wd__image_processing, etc.) and deployed as a systemd service to run headless from boot. Field testing over 270 cycles confirmed that both the mean-difference and KS-test methods met "few-centimeter" accuracy targets on a resource-constrained Pi Zero 2 W, with the service reliably cycling and logging results.

B.4. Hardware development

Over the course of five iterative prototypes, the hardware platform evolved from a bare-bones test setup into a fully integrated, field-ready enclosure. At each stage, lessons learned in bench and field trials drove targeted improvements in reference-object stability, illumination, power management, and environmental resilience.

To optimize processing, the camera output was downsampled. Rather than capturing the full 3280 × 2464 pixel frame, the Raspberry Pi Camera was configured to output a 640 × 480 pixel image. Technically, this is achieved by dividing the full sensor area into nine equal sections and selecting only the central compartment. Importantly, this preserves the resolution of the field of view while substantially reducing image file size, memory usage, and processing requirements.

The initial goal was to verify basic camera-based staging and data logging on the Raspberry Pi Zero 2 W with minimal investment of time and resources. A generic 20.000 mAh USB power bank provided 5 V/2.4 A to both the Pi and a TP-Link M7200 portable LTE router, enabling remote image capture even where Wi-Fi was unavailable. The Pi and camera module V2 were housed in an off-the-shelf Pi Zero plastic case. This assembly was simply taped onto a fence next to the gutter at Tu Delft. Although sufficient to test the Otsu-threshold algorithm in Prototype V1, the open case offered no protection from rain.

Addressing the glare issues, Prototype V2 introduced a circular polarizer was attached in front of the lens aperture, and the Pi case remained unchanged. Power wiring remained the same. This iteration proved the value of a stable, high-contrast gauge, though reflections under variable illumination still induced noise.

To enable reliable night-time captures, Prototype V4 incorporated a MOSFET-driven IR LED emitter (ILH-IN01-85SL-SC211-WIR200) mounted on an LED-star heatsink with thermal interface fluid. A small PCB bearing the MOSFET driver and a 1.2 Ω/5 W current-limiting resistor was wired between the powerbank's second USB output and the LED; cable lugs and heat-shrink provided safe installation and modularity. A custom 850 nm band-pass filter and high-quality circular polarizer were added via a photography filter holder attached the lid. The Pi and MOSFET board were attached to a 60 × 90 mm prototyping PCB, replacing the off-the-shelf case. This arrangement permitted visual confirmation of IR operation, reduced day/night variability, and maintained modularity for rapid assembly.

In the final iteration, all components; Raspberry Pi Zero 2 W (in an aluminium heatsink case), IR-LED driver PCB, powerbank, USB hub, and camera; were consolidated onto a single prototype. The reference object became a 600 × 500 mm matte-white foam-PVC board mounted on 15 mm plywood with stainless-steel screws and saddle brackets, providing a broad, non-buoyant target. A 4 mm PVC foam board was cut to fit inside a weather-resistant box and lined with Velcro for rapid component placement. Adjustable steel L-bracket hinges and a wing-nut lock allowed precise tilt alignment against a mounting structure. Tension straps and a rear wooden block anchor provided secure fastening. These modular fixtures enabled consistent camera orientation.

By Prototype V5, the hardware package balanced modularity, weather resistance, and low-power off-grid operation: IR illumination drew only 10 mAh/h in regular bursts; the 13 400 mAh powerbank enabled 26 h runtime; and the camera, filters, and PCB remained firmly secured within a single sealed enclosure. Each design decision, whether selecting a MOSFET for LED switching or adopting Velcro-lined platforms for rapid reconfiguration, was driven by field robustness and low-cost simplicity.

All design decisions can be found in a chronological fashion in the development log in Annex G.

C

Criteria evaluation

This section evaluates how the developed water level monitoring system performed against the project's design requirements. A critical assessment is made per criterion, supported by the field test data. Where relevant, additional insights for future improvement are identified.

C.1. Measurement Performance

Measurement Availability (Requirement 2 + 3 — Must-have)
The system was designed to ensure that measured water level data is processed and made available within 5 minutes after collection. Field observations indicate that the system processed each capture well within the available timeframe (~20 s). During the test, the Raspberry Pi handled image capture, processing, and local storage without noticeable delay.

Continuous measurement was stable across the 42-hour deployment. No system failures or crashes were recorded, and the algorithm operated autonomously as intended. While the test duration was limited to two days, the initial signs are promising. However, longer deployment periods are needed to confidently confirm robustness for continuous operation under more variable and challenging conditions.

Water Level Detection Accuracy (Requirement 6 — Should-have)
The water level detection system achieved a Mean Absolute Error (MAE) of 1.97 cm and a Root Mean Squared Error (RMSE) of 2.79 cm using the mean-difference method, compared to manual references. The KS-test method performed slightly better, with an MAE of 1.97 cm and an RMSE of 2.49 cm. A clear bias was observed in both methods: 1.57 cm for mean-difference and 1.02 cm for KS-test. This bias is likely caused by surface reflectance and water transparency effects, particularly under daylight conditions.

95% of the detected water levels fell within 4.64 cm of the reference when using the KS-test method, and 4.17 cm using the mean-difference method. While the system does not yet meet the 2 cm accuracy design goal (only 64–65% of detections fall within that margin), these results indicate a solid baseline. Post-processing corrections could likely reduce the bias and improve accuracy further, bringing the system within specification and supporting its strong potential for low-cost monitoring applications.

All-Condition Measurement (Requirement 5 — Must-have)
The device successfully captured water level measurements during both daytime and nighttime. Interestingly, the results show slightly better precision during night periods. This likely results from the fact that under infrared-only illumination, the waterline becomes visually more distinct due to reduced ambient reflections and longer shutter times, leading to sharper transitions in the images. Daytime conditions, especially around dusk or in the presence of glare, posed more challenges. Despite this, the device remained operational and continued to detect waterlines throughout all lighting conditions, fulfilling the requirement.

Flood Protection (Requirement 1 — Must-have)
One key design advantage is that the system can be installed at a significant distance from the river, reducing the risk of flood damage. During field testing, the device was mounted approximately 8.63 meters from the riverbank, a safe distance under normal and moderate flood conditions. While extreme flood scenarios were not encountered, the elevated and flexible installation potential meets the flood resilience requirement conceptually. It would be valuable in future research to investigate how system performance varies with different distances to the reference object, as greater distances might influence detection robustness.

Secure Installation (Requirement 7 — Should-have)
During the two-day test, the device remained firmly in place, and no movement or misalignment was observed. Nevertheless, the mounting system was designed for short-term field testing, not extended deployments. Further work should focus on developing a more versatile and durable mounting solution, ideally in close collaboration with end users to ensure practicality and ease of use in diverse field conditions.

Weather Resistance (Requirement 8 — Should-have)
The device remained operational during light rainfall events experienced during the field test. No water ingress or malfunction was observed. However, the prototype has not yet been exposed to extreme weather conditions such as heavy storms, strong winds, or high temperatures. Full validation of weather resistance thus remains an open item for future testing.

Vandalism Resistance (Requirement 9 — Should-have)
No incidents of vandalism occurred during field deployment. However, this is likely due to the protected nature of the test site. Vandalism resistance was approached mainly through design choices: keeping the device as small as possible, maintaining a low material value to minimize incentive, and enabling elevated or hidden installation. While these strategies are promising, the actual effectiveness of the design against vandalism was not formally tested and remains to be evaluated in more exposed environments.

C.2. Operational Aspects

Cost Effectiveness (Requirement 4 — Must-have)
The total material cost of the prototype was approximately €290, comfortably below the €400 ceiling defined in the requirements list. The choice of affordable components, open-source design, and low-power architecture ensured cost efficiency.

Energy Independence (Requirement 10 — Could-have)
The device operated autonomously for approximately 42 hours on a single battery cycle. However, the goal of achieving at least six months of energy independence was not yet met. Solar panel integration in combination with a power management system is foreseen as a promising next step to achieve this target, but it was beyond the scope of the current research phase.

Data Storage Backup (Requirement 11 — Could-have)
The system successfully backed up data to a USB storage device during GUI operation. However, USB backup functionality was not yet implemented under CLI operation, which is the preferred field setting. This is recognized as a small technical gap, and future work could easily address this with minor script adaptations.

Data Accuracy Filtering (Requirement 12 — Won't-have) The current prototype does not implement active filtering of statistical outliers. Outlier handling and automatic quality control remain identified areas for future development, to enhance system reliability before transmitting data to external servers.

Maintenance-Free Operation (Requirement 13 — Won't-have)
While the prototype operated autonomously for ~42 hours without maintenance, the original design goal of six months maintenance-free operation was not tested. Achieving true long-term maintenance-free functionality would require improved robustness, weatherproofing, and energy supply upgrades, forming important directions for future work.

C.2.1. Summary of Performance

This section summarizes the degree to which the developed water level monitoring system met the predefined design requirements. Each requirement is assessed based on field test results and development observations. For clarity, performance is indicated using green (met), orange (partially met), and red (unmet).

ID	Requirement	MoSCoW	Assessment	Status
1	Flood protection	Must-have	Device can be installed at safe distance from river; no flood exposure during test but conceptually validated	Met
2	Measurement availability	Must-have	Measurement cycle within target interval	Met
3	Continuous measurement	Must-have	Stable operation over 42h test without crashes	Met
4	Cost effectiveness	Must-have	Total cost (€290) well below €400 limit	Met
5	All-condition measurement	Must-have	Functioned both during day and night; better precision at night	Met
6	Water level detection accuracy	Should-have	95% accuracy ±4cm (outside ±2 cm)	Unmet
7	Secure installation	Should-have	Stable for short-term field test; mounting system needs development for long-term use	Partly met
8	Weather resistance	Should-have	Light rain tolerated; no extreme weather tested	Partly met
9	Vandalism resistance	Should-have	No vandalism occurred; design minimizes risk but effectiveness untested	Partly met
10	Energy independence	Could-have	42h on battery; 6-month target not achieved, solar future option and power management system to be implemented	Unmet
11	Data storage backup	Could-have	Worked in GUI mode; not yet in CLI operation	Partly met
12	Data accuracy filtering	Won't-have	Outlier filtering not yet implemented	Unmet
13	Maintenance-free operation	Won't-have	Short test duration; maintenance-free goal untested	Unmet

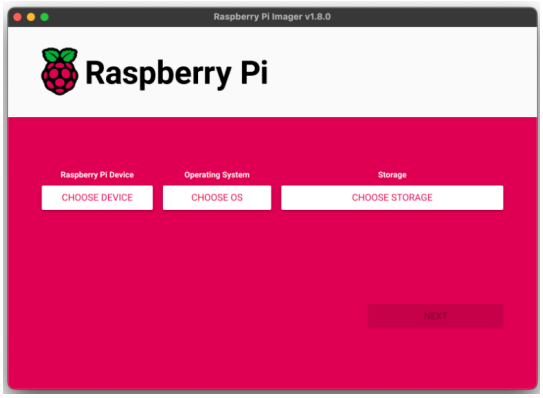
Table C.1: Summary of requirement evaluation with MoSCoW classification and performance status.

D

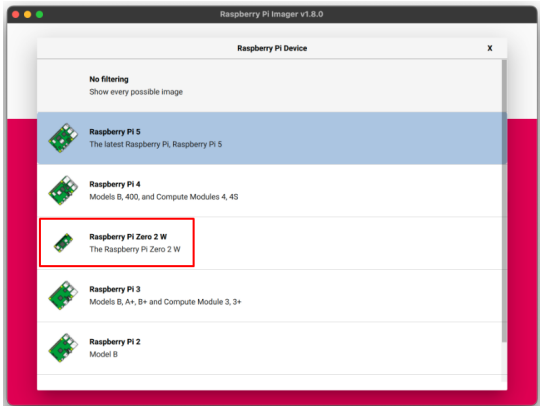
System Setup Manual

D.1. Operating System

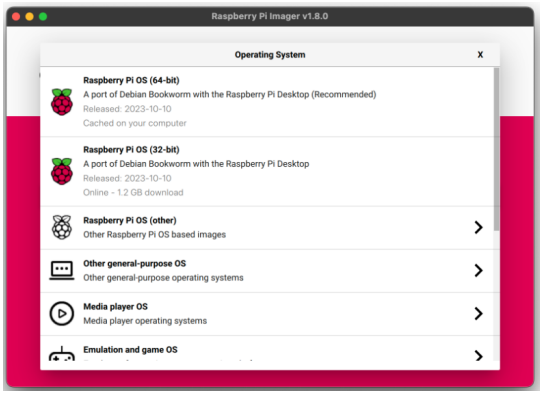
The Raspberry Pi OS is flashed onto the microSD card using the official *Raspberry Pi Imager*, available via raspberrypi.com. After launching the Imager, select the Raspberry Pi Zero 2 W (Figure D.1b), followed by the 64-bit Debian Bookworm with desktop OS image (Figure D.1c). Insert the microSD card using a card reader (Figure D.1d). Before writing, optional settings such as username, Wi-Fi, hostname, locale, keyboard layout, and SSH/VNC access can be preconfigured. Once applied, the OS is written to the card, producing a ready-to-use environment.



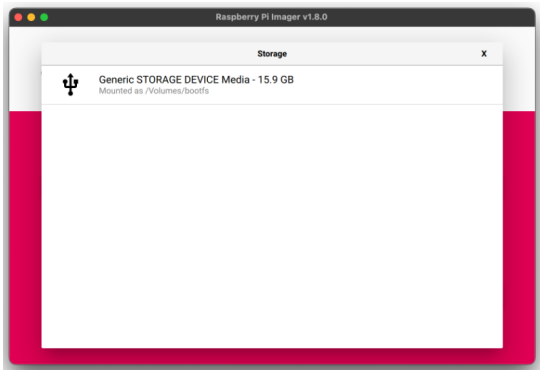
(a) Select target device.



(b) Select processor module.



(c) Select operating system.



(d) Select target storage.

Figure D.1: Overview of the SD-card imaging process.

D.2. First-Boot Configuration: Swapfile and Updates

To allow initial GUI-based configuration, the Pi Zero 2 W can be connected to an external monitor (HDMI → mini-HDMI) and controlled via USB hub (micro-USB → USB-A) with keyboard and mouse. After booting into the desktop, the default 100 MB swapfile is increased to 1024 MB to improve stability under image-processing load; care should be taken to avoid excessive SD-card wear (by increasing the swapfile size too much) (source). This can be achieved using the terminal as follows:

1. Disable swap temporarily:

   ```
   sudo dphys-swapfile swapoff
   ```

2. Edit the swap configuration file:

   ```
   sudo nano /etc/dphys-swapfile
   ```

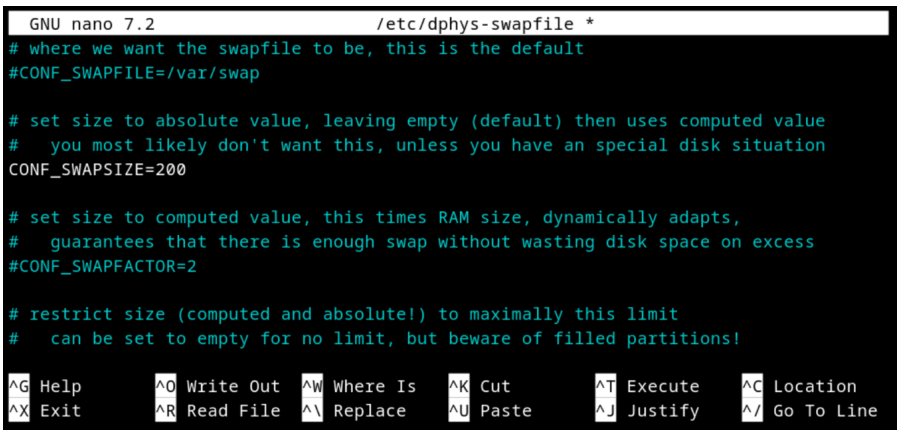


Figure D.2: Swapfile configuration.

3. Increase swap size: Change the line `CONF_SWAPSIZE=100` to `CONF_SWAPSIZE=1024`. Save changes with `CTRL + O`, `CTRL + X`, and confirm with `Y + ENTER`.
4. Apply the new swap size:

   ```
   sudo dphys-swapfile setup
   ```

5. Re-enable swap:

   ```
   sudo dphys-swapfile swapon
   ```

6. Reboot the system:

   ```
   sudo reboot
   ```

After reboot, the system is stable for all GUI and processing tasks. Regular package updates are then performed:

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

D.3. Package Installation

The water-level detection software depends on both system-level camera drivers and a set of Python libraries. Before operation, ensure the following packages are installed on Raspberry Pi OS:

System & camera features

- libcamera-apps
- libcamera-dev
- v4l-utils

Core Python libraries

- python3-picamera2 (Picamera2 API & libcamera bindings)
- python3-pillow (PIL image handling)
- python3-numpy (array operations)
- python3-matplotlib (plotting backend)
- python3-scipy (signal processing & statistics)
- python3-requests (HTTP for OCR API)
- python3-psutil (system metrics: CPU, memory)
- python3-rpi.gpio (GPIO control)

This can be achieved by running the following line in the terminal command prompt:

```
sudo apt update
sudo apt install -y libcamera-apps libcamera-dev v4l-utils \
python3-picamera2 python3-pillow python3-numpy \
python3-matplotlib python3-scipy python3-requests \
python3-psutil python3-rpi.gpio
```

D.4. Auto-Start Service

To enable the water-level detection algorithm to launch automatically after any reboot, create a `systemd` service unit. This ensures the system is self-recovering: if it loses power or crashes, it will come back online and resume measurements without manual intervention. In combination with a power-management board, energy use can be minimized by only powering the Pi when a measurement cycle is due (see Section 8).

Create and edit the service file with:

```
sudo nano /etc/systemd/system/<filename>.service
```

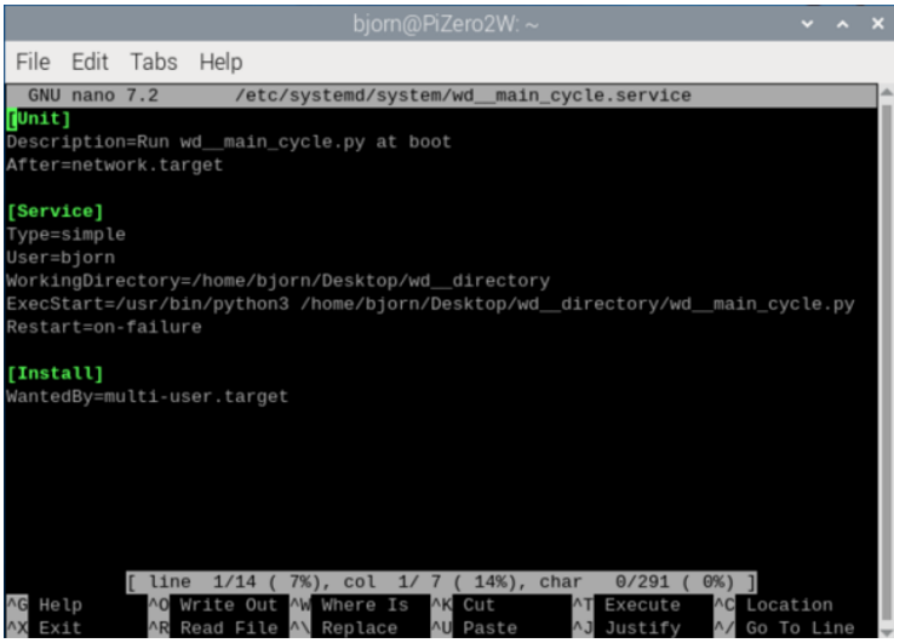


Figure D.3: Example of .service file

This file will be available on `<GitHub link>`.

The service file begins with a [Unit] section that names the task and tells systemd to wait until the network is online before starting. In the [Service] section, it specifies that systemd should run the Python interpreter on your wd__main_cycle.py script as the pi user, from the project's working directory, and automatically restart it if it ever crashes. Finally, the [Install] section hooks the service into the normal multi-user boot sequence so that enabling it causes the script to launch on every reboot without any further intervention.

After placing the service unit file into /etc/systemd/system/, the systemd manager must reload its configuration so that the new unit becomes available. The service can then be configured to start automatically on boot or prevented from doing so, and it supports manual start and stop operations as well as status inspection. This ensures that the water-level detection algorithm recovers automatically after power interruptions and can be managed interactively for troubleshooting or maintenance.

The relevant commands are:

```
sudo systemctl daemon-reload
sudo systemctl enable <filename>.service
sudo systemctl disable <filename>.service
sudo systemctl start <filename>.service
sudo systemctl stop <filename>.service
sudo systemctl status <filename>.service
```

E

Remote access manual

To facilitate remote access and file transfer, a VNC software and SFTP client are configured.

E.1. Remote access GUI mode; RealVNC

1. Enable SSH and VNC interfaces via **Menu** → **Preferences** → **Raspberry Pi Configuration** → **Interfaces**, selecting **Enabled** for both **SSH** and **VNC**.

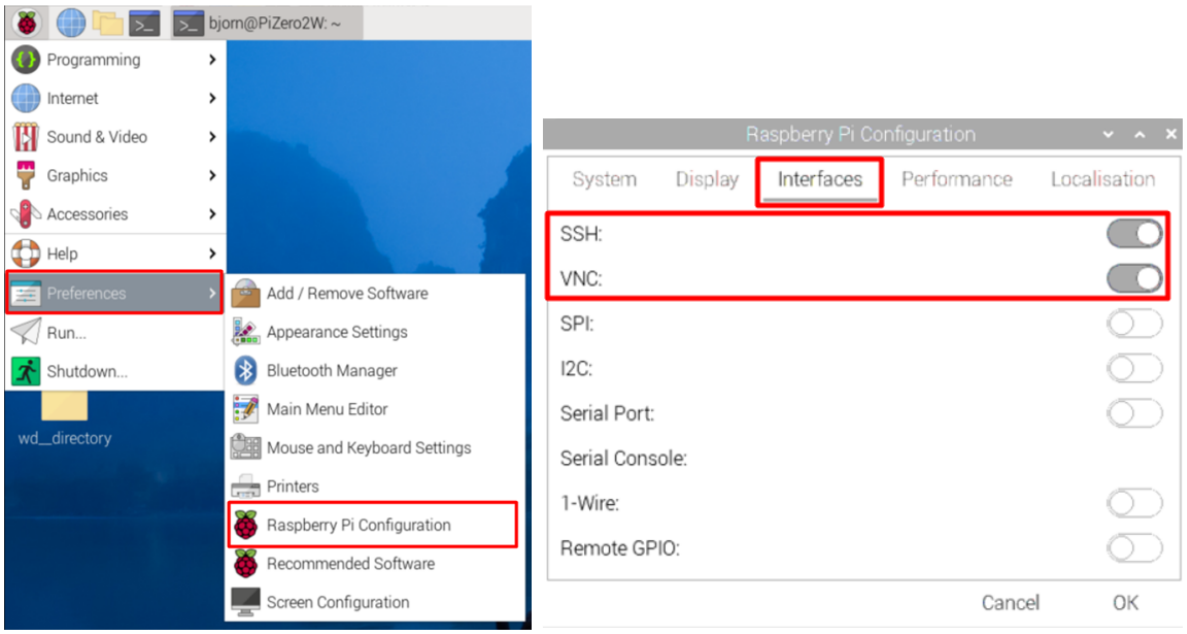


Figure E.1: Enable SSH and VNC settings in RPI environment.

2. From a workstation, download and install RealVNC Viewer (`https://www.realvnc.com/download/`).
3. In VNC Viewer, create a new connection using the Pi's IP address (visible by hovering over the network icon on the RPi desktop) as the VNC Server address.
4. Authenticate with the Pi user credentials configured during OS imaging; upon success, the Pi desktop is displayed for remote control.

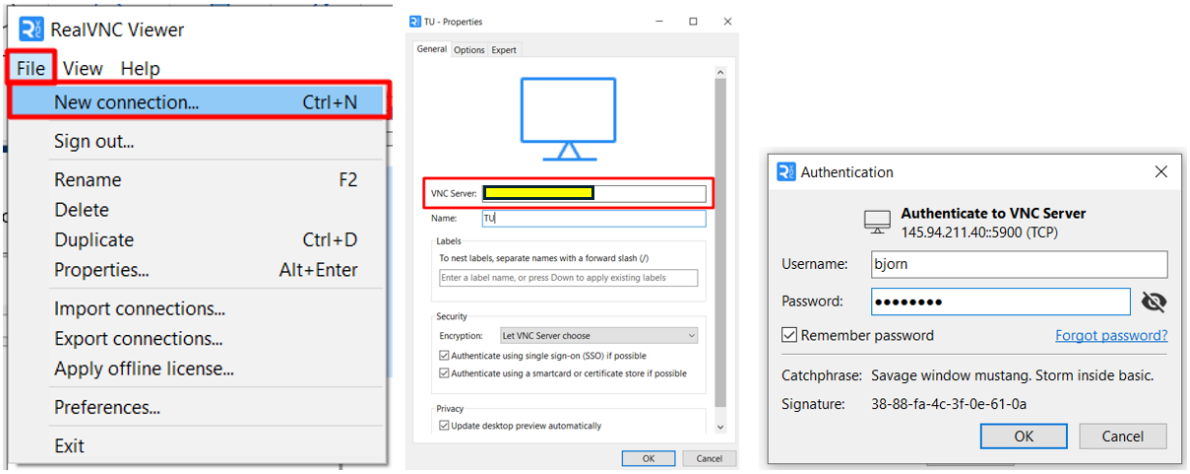


Figure E.2: Setting up a remote connection using RealVNC.

Now the system can be controlled form a pc assuming both systems are using the same internet connection.

E.2. Remote access CLI mode; SSH & PuTTY

In deployments where power savings dictate operating without a monitor or GUI, the Raspberry Pi must be accessed remotely via SSH over Wi-Fi or a mobile hotspot. The following steps describe enabling the SSH server, configuring wireless network credentials, and establishing an SSH connection from a Windows workstation using PuTTY(source PuTTY).

Initial SSH enablement (with monitor & keyboard)

1. Connect the Pi to a monitor and USB keyboard.
2. Open a terminal and run:

   ```
   sudo raspi-config
   ```

3. Navigate to **Interfacing Options** → **SSH**, select **Enable**, then **OK**.

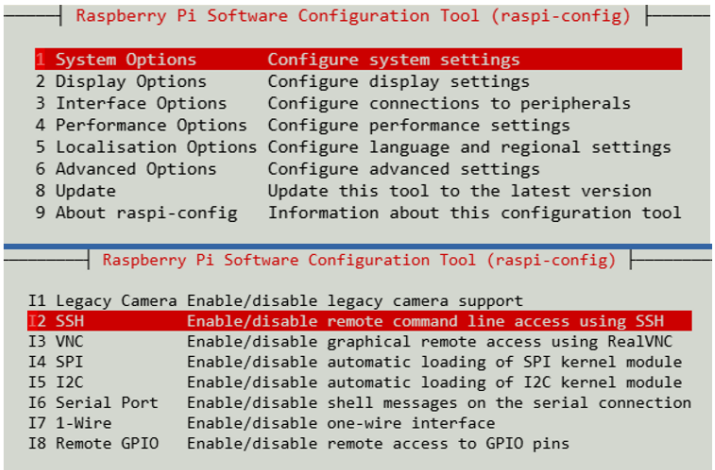


Figure E.3: Set interfacing options.

Configure Wi-Fi credentials:

4. In the same `raspi-config` tool, go to **System Options → Wireless LAN**.

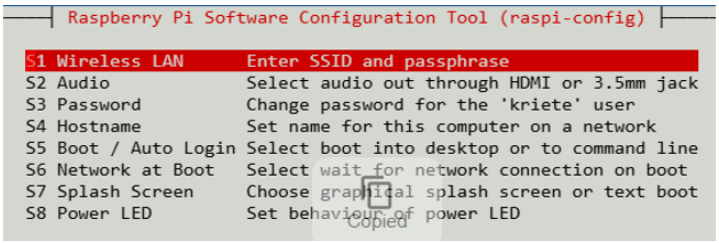


Figure E.4: Configure Wireless LAN connection.

5. Enter your hotspot's SSID and passphrase when prompted.
6. Exit and allow the Pi to reboot (if prompted).

Determine the Pi's IP address on the hotspot:

7. Open a terminal and run:

   ```
   ip a
   ```

8. Note the address listed under the Wi-Fi interface.

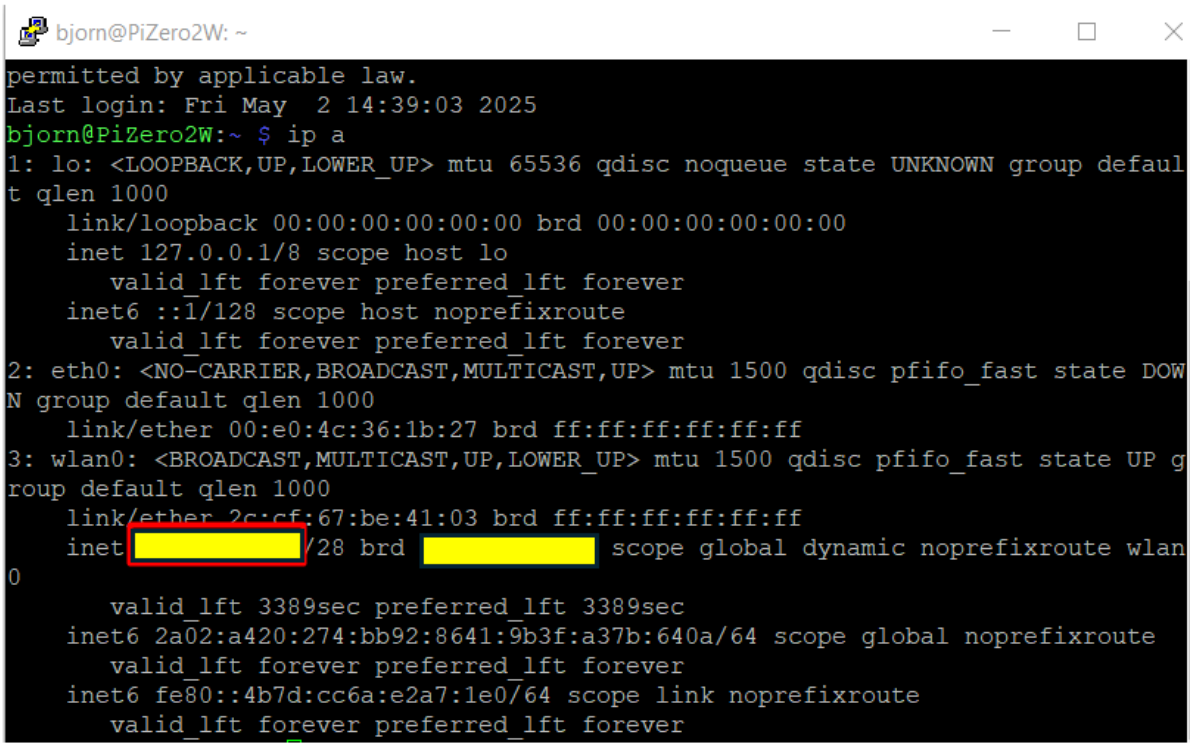


Figure E.5: Location of IP-address

Establish the SSH connection:

9. Download and install PuTTY from `https://www.putty.org/`
10. Launch PuTTY.
11. In **Host Name (or IP address)** enter the Pi's IP from step 8.
12. Ensure **Port** is set to 22 and **Connection type** is SSH.

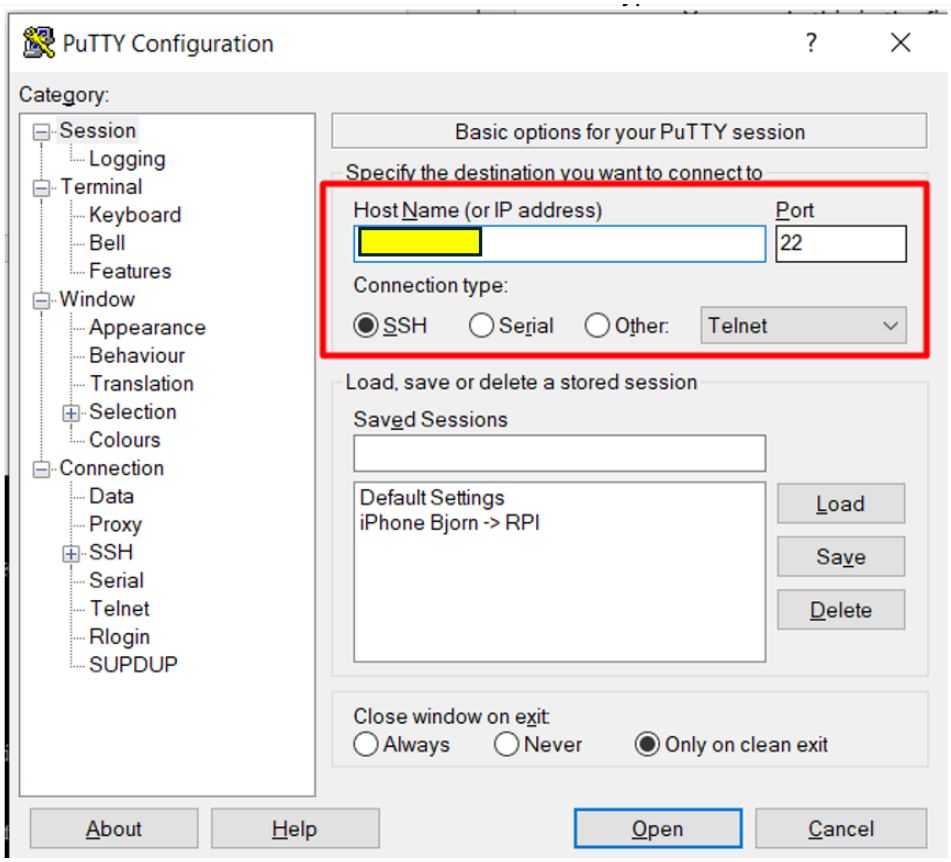


Figure E.6: Insert Host name/ IP-address

13. Click **Open**, then log in with your Raspberry Pi username and password.

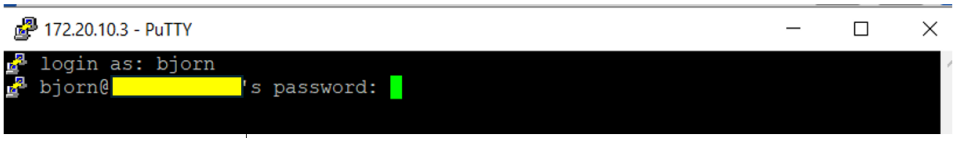


Figure E.7: Login interface for PuTTY

Once connected, you can manage the Pi entirely via the terminal in CLI mode.

E.3. SFTP file transfer via WinSCP

To enable reliable file transfer, the SFTP client WinSCP is configured.

1. Install WinSCP on the workstation (`https://winscp.net/`).
2. Launch WinSCP and configure a New Site with:
   - File protocol: **SFTP**
   - Host name: `Pi IP address`
   - Port number: `<standard>`
   - User name: `<username>` (as set in the OS imager)
   - Password: `<password>` (as set in the OS imager)
3. Save the site and connect; navigate to the project output directory (e.g. `/home/pi/wd_directory/output/`) to upload or download images, logs, and CSV files.

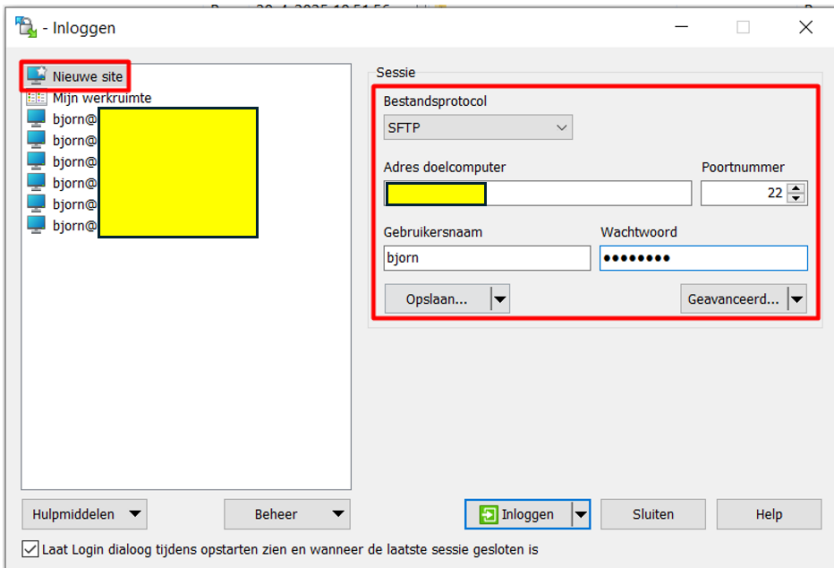


Figure E.8: Setup file transfer configuration.

Once connected, WinSCP can also be used to upload the entire project directory (cloned from the GitHub repository) to the Pi. Simply clone the wd__directory locally on your workstation (e.g. via git clone (GitHub link)), then in WinSCP's local pane navigate to that folder and drag it into /home/pi/ on the remote pane. This transfers all scripts, modules, and configuration files in one step, ready for execution.

F

Operation and calibration manual

F.1. Calibration

Accurate water-level detection requires that the optical axis and algorithm parameters be tuned to the installed reference object. With the enclosure sealed and the reference board fixed, calibration proceeds in two stages: physical alignment and parameter extraction.

First, a live preview is used to finalize camera orientation. In GUI mode (using RealVNC), `Camera_preview.py` (in `wd__Calibration/`) is launched so that the system orientation can be adjusted to the reference object. Once the mount is locked in place, the preview window is closed.

Next, rotation and crop parameters are determined by running `get_crop_angle_parameters.py`, which guides the user through three interactive clicks:

1. Rotation: two clicks on the waterline (left then right) compute the angle required to horizontalize the image.
2. Cropping: four clicks define the rectangular region of interest around the board.
3. Preview: a final overlaid image confirms that the waterline is level and the crop box fully encloses the board.

Once the rotation and crop values are confirmed, they are entered into `wd__config_cycle.py` under `"processing_params"` and `"crop_params"`. Additionally, the interval between measurement cycles governed by the `cycle_rest_seconds` parameter in `wd__config_cycle.py`, should be reviewed and adjusted to suit the deployment requirements (9 minutes in this research). This ensures that the system waits the correct amount of time between successive captures without needing manual intervention.

Verification is then performed by executing a single detection cycle, ideally from within the Thonny IDE (standard Python terminal on RPI OS) so that the initial processing output can be observed. All results (captured images, overlay plots, and CSV logs) are saved to `wd__directory/wd__results/`, and Thonny's console displays informational and error messages to facilitate debugging.

After calibration is confirmed, the Pi must be switched to a CLI mode (text-console) to save power. This is accomplished by running `sudo raspi-config` in a terminal, navigating to:

```
Sudo raspi-config
System Options
Boot
Console Text Console
```

Next, select "Finish" and agree to reboot by selecting "Yes".

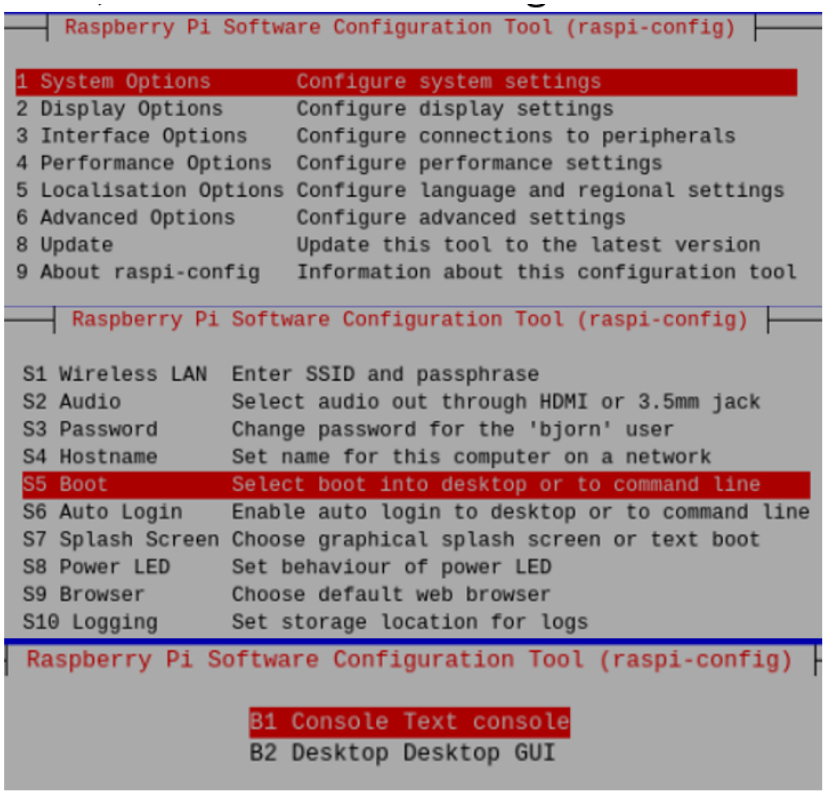


Figure F.1: Switching RPI boot-up configuration

Once the system restarts, SSH access via PuTTY (Section 5.2.5) restores terminal control. The status and activity of the detection service may then be checked with the appropriate `systemctl` commands, and if network posting is enabled, successful uploads can be verified on the OpenRiverCam server.

F.1.1. Device setup checklist

1. Clear al data storages / charge all batteries
2. Setup mobile hotspot
   - `<host name>`, `<host password>`
   - Connect laptop with mobile hotspot
   - Turn on Raspberry Pi and seal device
3. Mount device facing Reference Object
4. Run: `Camera_preview.py` in `wd__Calibration` using Thonny
   - Adjust device orientation so that OI is in the frame
   - Secure camera orientation
   - Adapt lens focus using lens tool
   - Exit out
5. Run: `get_crop_angle_parameters.py` in `wd__Calibration` using Thonny
   - Rotation: two clicks on the waterline (left then right) compute the angle required to horizontalize the image.
   - Cropping: four clicks define the rectangular region of interest around the board.
   - Preview: a final overlaid image confirms that the waterline is level and the crop box fully encloses the board.

6. Open: `wd__config.py` in `wd__directory`
    - Insert rotation and crop parameters
    - Insert cycle wait time under the `cycle_rest_seconds` parameter
    - Save & Exit out
7. (Test)Run: `wd__main_cycle.py` in `wd__directory`
    - Check for errors
    - Check behaviour LED
    - Check cropbox file in `wd__results`
    - Check 4modes file in `wd__results`
    - Check raw images
    - Check writing to usb
    - Check sleep parameter
8. Switch to no GUI mode
    - Open terminal

```
Sudo raspi-config
1 system options
S5 boot
B1 Console Text console
Finish
```

    - Reboot now: no
9. Turn on reboot service file
    - `sudo systemctl enable wd__main_cycle.service`
    - `sudo systemctl status wd__main_cycle.service`
10. Reboot
11. Check IR led if program is running correctly
12. (optionally) Utilize a SSH connection via PuTTY to validate operation

F.1.2. Disable device checklist

1. Take down device
    - Move to safe location (dry and not above water)
2. Make SHH connection via PuTTY
3. Check, stop and disable algorithm
    - `sudo systemctl status wd__main_cycle.service`
    - `sudo systemctl stop wd__main_cycle.service`
    - `sudo systemctl disable wd__main_cycle.service`

F.2. Operation

Once calibration is complete and the Pi has been configured to boot into CLI mode, measurement cycles proceed automatically at the interval specified by the `cycle_rest_seconds` parameter in `wd__config_cycle.py`. In normal operation, the system illuminates the reference board, captures an image, runs the detection algorithm, uploads result if enabled, and then sleeps until the next cycle without any further user intervention.

If a cycle needs to be triggered on demand the detection service may be restarted manually. In the terminal, the following command forces an immediate cycle.

```
sudo systemctl restart wd_main_cycle.service
```

To confirm that the service is running correctly and view recent log entries, the operator can issue:

```
sudo systemctl status wd_main_cycle.service
```

Each cycle writes its outputs to the file system: raw photographs appear under `~/output/raw_images/`; annotated PNGs with the detected waterline are saved in `~/results/`; and a line is appended to the CSV log at `~/results/algorithm_results.csv`. These files can be investigated or retrieved remotely over the RealVNC connection by first pausing the detection service and reverting to GUI mode. To do this, stop the service:

```
sudo systemctl stop wd_main_cycle.service
```

then run `sudo raspi-config` to switch the boot target back to the GUI mode and reboot. Once the Pi restarts, connect via RealVNC and use WinSCP or a similar SFTP client to access and copy the output files. Once the transfer is complete, CLI mode is restored by reconfiguring the boot target and restarting the service:

```
sudo systemctl start wd_main_cycle.service
```

Periodic maintenance includes verifying that the battery maintains sufficient charge, inspecting the reference board for any debris or alignment shift, and checking the enclosure's seals and mounting hardware. By following these procedures, the system delivers reliable, unattended water-level measurements while still allowing straightforward manual checks and data retrieval when needed.

G

Development LOG

7 DEVELOPMENT LOG

7.1 PROTOTYPE V1:

7.1.1 Hardware V1

Components

Function	component
Power supply	ISY powerbank 20.000mA, 2.4A, 5V
Processing	Raspberry Pi Zero 2W
Camera	RPI Camera module V2
Internet connection	Tp-Link portable router
Casing	RPI Zero standard casing

Staff gauge v1

Piece of foam temporarily attached to a spade using tiewraps.

Power source

The powersource used in the first prototype is a Lithium-polymeer battery capable of outputting 2x 5 volts with 2.4A current. This power source is taken as a first option as it was available by the researcher and able to stably power both the operating system (RPI) and the portable router. It contains a capacity of 20.000mA which is more than enough to do preliminary testing of a few hours at a time. The battery weighs about 375g, and dimensions are 73.5x24x152mm. The battery costs about 20€.

Operating system

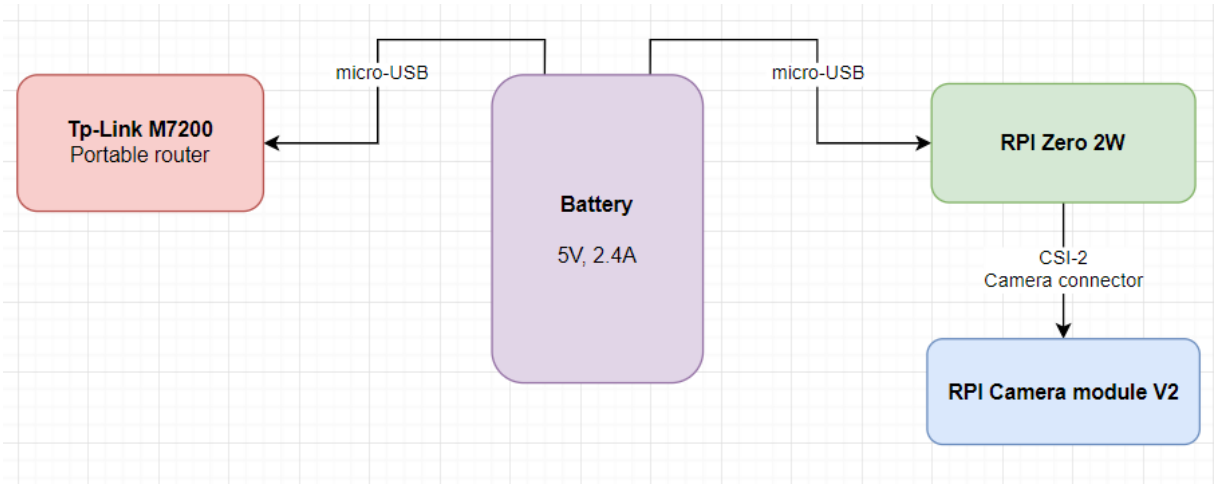
The raspberry pi zero 2W contains a quad-core 64-bit ARM Cortex-A53 processor clocked at 1GHz and 512MB of SDRAM. The device offers 2.4GHz 802.11 b/g/n wireless LAN and Bluetooth 4.2, along with support for Bluetooth Low Energy (BLE). It has a built in wireless LAN and is dimensioned 65mm x 30mm. The board has a microSD card slot, a CSI-2 camera connector, a USB On-The-Go (OTG) port and an unpopulated footprint for a HAT-compatible 40-pin GPIO header. It can be powered via a micro USB socket and output via a mini HDMI port (Raspberry Pi, 2024). The RPI zero 2 W costs about 20€.

Camera module

The camera module V2 is an 8-megapixel module and dimensioned around 25 x 24 x 9 mm. it weighs 3g and is capable of capturing video in 1080p47, 1640 x 1232p41 and 640x480p206. It uses a Sony IMX219 sensor with a resolution of 3280 x 2464 pixels. It comes with adjustable focus and a focal length of 3.04mm. Horizontal Field of View (FoV) is 62.2 degrees and Vertical Field of View (FoV) is 48.8 degrees. Its maximum exposure time is 11.76 seconds, and the product is available in a NoIR version, where the IR filter is removed (raspberrypi, 2025). The camera module V2 costs about 18€.

Portable router

The TPL-link M7200 is a portable router that supports 4G FDD/TDD-LTE and 3G DC-HSPA signals. It contains a battery with a capacity of 2000mAh which translates to roughly 8 hours of Wi-Fi connection. It can be powered via a micro usb-port and allows 1 sim card. Its capable of 150 Mbps download speed and 50 Mbps upload speed. Its dimensioned 94 x 56.7 x 19.8mm (tp-link, 2025) (tp-link, 2025). It costs about 50€.



7.1.2 Algorithm V1

Nr.	File name	link	Description
V1_1	Scripts/Otsu_cropping_th-factor_21112024_V4.ipynb	Algorithm v1	Otsu method + preprocessing
V1_2	Scripts/Timelapse_5mins.py	Timelapse v1	Timelapse image capture

V1_1

Algorithm component V1_1 utilises packages numpy for basic python computations, Pillow and scipy.ndimage for image processing, matplotlib for visualisation, os for managing paths and time to analyse computation demand of the script. The OpenCV package is refrained from because of its size, which is not recommended while using devices running on low processing power like the Raspberry Pi Zero 2 W.

The Otsu thresholding method is a widely used technique in computer vision for image segmentation. Its purpose is to determine an optimal threshold value that separates pixels into two classes: the object of interest and the background. [https://medium.com/@vignesh.g1609/image-segmentation-using-otsu-threshold-selection-method-856ccdacf22]

The algorithm analyses the histogram of pixel intensities in the image and identifies the threshold value that minimizes the variance within each class. After applying this threshold, the image is binarized, which simplifies analysis using basic coding techniques [https://medium.com/@vignesh.g1609/image-segmentation-using-otsu-threshold-selection-method-856ccdacf22].

Medium source based on this paper:
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4310076&tag=1

Description	Formula
Threshold value	k
Probability of pixel in image - Background class 0: pixel i below k - Foreground class 1: pixel i above k	$p_i = n_i/N$
Probability of picking point from class 0	$\omega_0 = \sum_{i=0}^{k} p_i$
Probability of picking point from class 1	$\omega_1 = \sum_{i=k}^{255} p_i = 1 - \omega_0$
Mean levels of each class	$\mu_0 = \sum_{i=1}^{k} ip_i$ $\mu_1 = \sum_{i=k}^{255} ip_i$
Variance of each class	$\sigma_0^2 = \sum_{i=0}^{k} (1-\mu_0)^2 p_i/\omega_0$ $\sigma_1^2 = \sum_{i=0}^{k} (1-\mu_1)^2 p_i/\omega_1$
Within class variability	$\sigma_W^2 = \omega_0\sigma_0^2 + \omega_1\sigma_1^2$
Between class variability	$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$ $= \omega_0\omega_1(\mu_1 - \mu_0)^2$
Total variance	$\sigma_T^2 = \sum_{i=1}^{L} (i - \mu_T)^2 p_i$
Otsu Function	$\sigma_B(k) = \omega_0(k)\sigma_0^2(k) + \omega_1(k)\sigma_1^2(k)$

[https://medium.com/@vignesh.g1609/image-segmentation-using-otsu-threshold-selection-method-856ccdacf22]

Otsu method functions

```python
# Function for thresholding an image
def threshold_image(im, th):
    return np.where(im >= th, 255, 0).astype(np.uint8)

# Function to compute Otsu's criteria
def compute_otsu_criteria(im, th):
    thresholded_im = im >= th
    nb_pixels = im.size
    nb_pixels1 = np.count_nonzero(thresholded_im)
    weight1 = nb_pixels1 / nb_pixels
    weight0 = 1 - weight1
    if weight1 == 0 or weight0 == 0:
        return np.inf
    val_pixels1 = im[thresholded_im]
    val_pixels0 = im[~thresholded_im]
    var0 = np.var(val_pixels0) if len(val_pixels0) > 0 else 0
    var1 = np.var(val_pixels1) if len(val_pixels1) > 0 else 0
    return weight0 * var0 + weight1 * var1

# Function to find the best threshold with an adjustment factor
def find_best_threshold(im, adjustment_factor=2):
    threshold_range = range(0, np.max(im) + 1, 10)
    criterias = [compute_otsu_criteria(im, th) for th in threshold_range]
    best_threshold = threshold_range[np.argmin(criterias)]
    adjusted_threshold = best_threshold * adjustment_factor
    return min(adjusted_threshold, np.max(im))  # Ensure it does not exceed the max pixel value
```

A adjustment factor is introduced to the script to improve its performance. This adjustment factor is mannualy adjusted to improve the output of this specific image. It is not part of the original method and not automatically an improvement for other images taken with different lighting or environmental conditions. The original code is taken form the following source: [https://medium.com/@vignesh.g1609/image-segmentation-using-otsu-threshold-selection-method-856ccdacf22]

Processing otsu method output and selecting longest vertical object

Utilizing scipy.ndimage, the longest vertical string of connected pixels is localised in the binary image. for optimization purposes, the width of this object is limited to match the general width of the staff gauge. implementing this greatly reduced the noise of other long connected vertical polygons in the binarized images caused by glare or other background pixels that where not correctly classified.

The function [find_longest_vertical_string] labels connected components and stores them in objects. Next, it iterates trough those objects and calculates the height of each object. If a component is found, it determines if it does fit the max width criteria. It outputs the longest vertical component that confirms the boundary conditions of the manually added max width, which should be implemented by the user and is variable per experiment or setup.

```python
# Function to find the longest vertical connected component
def find_longest_vertical_string(binary_image, max_width=50):
    """
    Find the longest vertical connected component in the binary image
    and limit the width for visualization purposes.
    """
    # Label the connected components
    labeled_image, num_features = label(binary_image)
    objects = find_objects(labeled_image)

    max_height = 0
    longest_vertical_component = None

    # Iterate through each connected component and find the vertical one with the greatest height
    for obj in objects:
        y_slice, x_slice = obj
        height = y_slice.stop - y_slice.start
        width = x_slice.stop - x_slice.start

        if height > max_height and height > width:
            max_height = height
            longest_vertical_component = (y_slice, x_slice)

    # If a component is found, adjust the width to be at most `max_width`
    if longest_vertical_component:
        y_slice, x_slice = longest_vertical_component
        center_x = (x_slice.start + x_slice.stop) // 2  # Find the center of the width
        half_max_width = max_width // 2

        # Limit the x_slice to the central max_width region
        x_start = max(x_slice.start, center_x - half_max_width)
        x_stop = min(x_slice.stop, center_x + half_max_width)
        x_slice = slice(x_start, x_stop)

        longest_vertical_component = (y_slice, x_slice)

    return longest_vertical_component
```

Running the algorithm

- Load and preprocess the image

Opening the image, converting it to grey scale and storing both variants.

- Cropping the image to the area of interest

Depending on the situation, it is recommended to crop the image as much as possible to reduce noise and processing time/power. This step is highly dependent on location and setup.

- Apply otsu thresholding method
- Find longest vertically connected component which fits requirements
- Plot images

For analysis, the original image, grayscale image, cropped image and binary image with the cropped region displayed as an overlay are plotted. This takes significant computational power for the RPI hence this is purely done for analysis and optimisation of the algorithm. In practice, the original image would be posted and stored, together with all information about the processing parameters and waterline detection output.

- A summary file is written to store important information about the run and parameters.

Image capturing script for camera module

Packages that are utilised running this script are os for path management, time for analysis of runtime and computational efficiency and picamera 2 for operating the camera module.
Using the standard configuration settings on the camera module images are taken with an resolution of 3280 x 2464. The script is setup using a timelapse which makes sure that every 10 minues, an image is taken and stored locally. Images names are generated with timestamps using YMDHMS.

7.1.3 Experiment V1

The runtime for the method on the Raspberry Pi Zero 2 W was **356 seconds** when processing pixels individually. By grouping pixels into 10x10 blocks, the runtime was significantly reduced to **61 seconds**, with similar results.

Observations:

- **Sky as noise**: The sky, often light in colour, was frequently detected as an object of interest, introducing noise into the results.
- **Concrete reflections**: Light-coloured concrete surfaces reflected enough light to also be classified as objects of interest, adding noise.
- **Effectiveness of thresholding**: While the thresholding method performed reasonably well in this experiment, its effectiveness may vary under different conditions, such as changes in lighting, cloudiness, or rain.
- **Staffgauge/water line**: the goal of this method is to determine the height of the water level. This means that a clear distinction between the water and staff gauge needs to be determined. In the images can be seen that the Otsu method has difficulties determining what pixel is part of the underwater staff gauge and what part is part of the above water staff gauge. This is caused by disturbed pixel intensities caused bay reflection near the waterline, but also the fact that the camera can detect the underwater part of the staff gauge.
- **Threshold**: during the first test, the threshold of the Otsu method is manually set so it performs the best. However, the best threshold is greatly dependent on environmental conditions like lighting (position of the sun, cloudiness, seasonality, brightness) and precipitation (snow, hail or rain). Also wind and vegetation like leaves flying trough the shot can affect the thresholding.

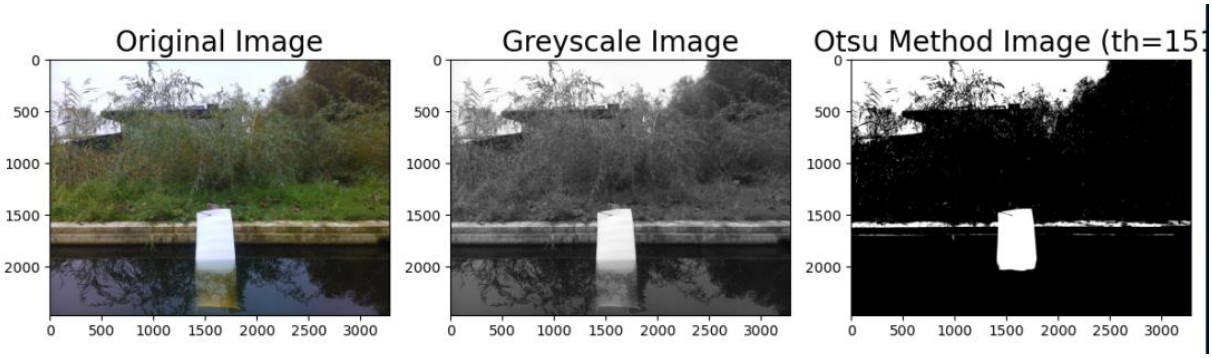


Figure 1: Results of Otsu thresholding method preliminary experiment

The find best threshold function loops trough all possible thresholds between and 255 (range of pixel intensities) and tries to find the best threshold that defines background or object of interest. However this processes can be very computational expensive (while using RPI). Trying every 10th threshold value however sped up the process +- 10x.

10 pixel thresholding **1 pixel thresholding**

```
Time to load and convert image: 0.8919 seconds
Time for thresholding: 27.5857 seconds
Time for plotting: 27.9491 seconds
Vertical height of the water gauge (in pixels): 2080
Time for calculating vertical height: 3.5239 seconds
Total script runtime: 61.0900 seconds
```

```
Time to load and convert image: 0.8207 seconds
Time for thresholding: 296.4523 seconds
Time for plotting: 50.9546 seconds
Vertical height of the water gauge (in pixels): 2080
Time for calculating vertical height: 7.2191 seconds
Total script runtime: 356.1274 seconds
```

7.1.4 Notes on prototype 1 and feedback loop for next prototype

The Otsu thresholding method was not able to differentiate the waterline correctly in a variable environmental conditions. For this method to work, manual adaptation of the thresholding factor is was needed to increase the effectiveness of the thresholding, which is not desirable for an autonomous system. Also reflection of light and clouds on the water surface caused noise in the thresholding method.

For the next design iteration, realising a stable staff gauge and improved detection method have priority. The focus will be on:

- Non floating staff gauge to stabilise the focus area
- Noise reduction in the water level detection algorithm
  - Reflections on water surface
  - Background noise

SCAMPER Heuristics

• Substitute: What materials or processes can be replaced?
  - Staff gauge is not functioning well. Need a more stable gauge that doesn't move in water.
• Combine: What elements can be integrated?
  - A form of noise reduction can be integrated to increase performance (background and reflection noise)
• Adapt: How can the concept be adjusted for improvement?
  - The current otsu thresholding method can be adapted so it is more robust for variable lighting conditions.
• Modify: What aspects (e.g., size or shape) can be altered?
  - The staff gauge has to be altered so it's a more consistent way of detecting the line between water and air.
• Put to another use: Can the concept serve other purposes?
  -
• Eliminate: What features can be simplified or removed?
  -
• Reverse: Can processes be reversed or re-sequenced?
  -

7.2 PROTOTYPE V2

7.2.1 Hardware

Staff gauge v1

Because of an poorly functioning temporary staff gauge v1, a second staff gauge was created. A white hollow piece of aluminium was connected to a handle and marked using water proof marker every 10 cm. the problem with staff gauge v1 was that the foam was too buoyant which caused the staff gauge to float up and not remain in the same location. The new staff gauge was able to be attached to the side of the river bed or experiment setup (gutter) and was much better at creating a stable measure.

7.2.2 Algorithm

A piece of code is introduced that enables making images using a timelapse manner. Using the most updated version of raspberry pi libraries: Dabian bookworm (www.debian.org). utilizing the Picamera2 package, the camera module can be controlled to take images and show previews.

Image capturing

```python
import os
import time
from picamera2 import Picamera2, Preview

def capture_timelapse(output_folder, interval_seconds, duration_seconds):
    """
    Captures timelapse images using the Raspberry Pi camera with timestamps.

    Parameters:
    - output_folder (str): Path to the folder where images will be stored.
    - interval_seconds (int): Time interval between consecutive captures in seconds.
    - duration_seconds (int): Total duration of the timelapse in seconds.
    """
    # Create the output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Initialize the PiCamera2
    picam2 = Picamera2()

    # Configure the camera for still capture
    camera_config = picam2.create_still_configuration()
    picam2.configure(camera_config)

    # Start the camera
    picam2.start()

    # Calculate the number of frames to capture
    num_frames = duration_seconds // interval_seconds

    print(f"Starting timelapse: {num_frames} frames, {interval_seconds}s interval")
    print(f"Images will be saved to: {output_folder}")
```

```python
    try:
        for frame in range(1, num_frames + 1):
            # Generate a filename with a timestamp
            timestamp = time.strftime("%Y%m%d_%H%M%S")
            filename = os.path.join(output_folder, f"Delft_{timestamp}.jpg")

            # Capture the image
            metadata = picam2.capture_file(filename)
            print(f"Captured {filename} with metadata: {metadata}")

            # Wait for the next interval
            if frame < num_frames:
                time.sleep(interval_seconds)

    except KeyboardInterrupt:
        print("Timelapse interrupted by user.")

    finally:
        # Stop the camera
        picam2.stop()
        print("Timelapse complete.")

# Run the script
if __name__ == "__main__":
    # Folder where images will be stored
    desktop_path = os.path.expanduser("~/Desktop")
    output_folder = os.path.join(desktop_path, "Timelapse_Images")

    # Timelapse settings
    interval_seconds = 300  # Capture an image every 10 seconds
    duration_seconds = 1800  # Total duration of 5 minutes

    capture_timelapse(output_folder, interval_seconds, duration_seconds)
```

Image processing

Furthermore, the package Pillow (PIL) and scipy.ndimage are used for image processing along side matplotlib to analyse results visually. These packages have a small demand on the minimalistic processing capabilities of the raspberry pi and therefore are preferred over large and more sophisticated libraries like OpenCV.

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import os
from scipy.ndimage import label, find_objects
import time
```

To increase userfriendlyness, a piece of code is introduced to easily switch between the testing environment (PC) and the endproduct/prototype (raspberry pi).

```
# Define the base path for the system (Raspberry Pi or PC)
def get_base_paths(system="pc"):
    if system == "raspberry_pi":
        return {
            "image_path": "/home/bjorn/Desktop/Timelapse_Images",
            "output_path": "/home/bjorn/Desktop/Test_Script_Output"
        }
    elif system == "pc":
        return {
            "image_path": r"C:\Users\bjorn\Desktop\rpi_mirror\Timelapse_Images",
            "output_path": r"C:\Users\bjorn\Desktop\rpi_mirror\Test_Script_Output"
        }
    else:
        raise ValueError("Invalid system. Choose 'raspberry_pi' or 'pc'.")
```

The same method (otsu thresholding) is used to detect the staff gauge in the captured images. To enhance the waterlevel detection capabilities, an additional piece of code is introduced. This piece of code uses scipy to label groups of pixels that are categorized as object of interest by the othsu method. The algorithm detects the largest vertically connected group of pixels and sees this as the staff gauge. other smaller groups of pixels that are labelled as object of interest by the otsu method are hereby filtered out, with the goals of filtering out noise from reflection or mistakenly marked background. To enhance this function, a maximum width is introduced that matches the dimensions of the staff gauge.

```python
# Function to find the longest vertical connected component
def find_longest_vertical_string(binary_image, max_width=50):
    """
    Find the longest vertical connected component in the binary image
    and limit the width for visualization purposes.
    """
    # Label the connected components
    labeled_image, num_features = label(binary_image)
    objects = find_objects(labeled_image)

    max_height = 0
    longest_vertical_component = None

    # Iterate through each connected component and find the vertical one with the greatest height
    for obj in objects:
        y_slice, x_slice = obj
        height = y_slice.stop - y_slice.start
        width = x_slice.stop - x_slice.start

        if height > max_height and height > width:
            max_height = height
            longest_vertical_component = (y_slice, x_slice)

    # If a component is found, adjust the width to be at most `max_width`
    if longest_vertical_component:
        y_slice, x_slice = longest_vertical_component
        center_x = (x_slice.start + x_slice.stop) // 2  # Find the center of the width
        half_max_width = max_width // 2

        # Limit the x_slice to the central max_width region
        x_start = max(x_slice.start, center_x - half_max_width)
        x_stop = min(x_slice.stop, center_x + half_max_width)
        x_slice = slice(x_start, x_stop)

        longest_vertical_component = (y_slice, x_slice)

    return longest_vertical_component
```

Image processing

The image processing algorithm was executed with extensive debugging information printed during runtime, along with a log file for analysis. The processed images were stored in an output folder for review.

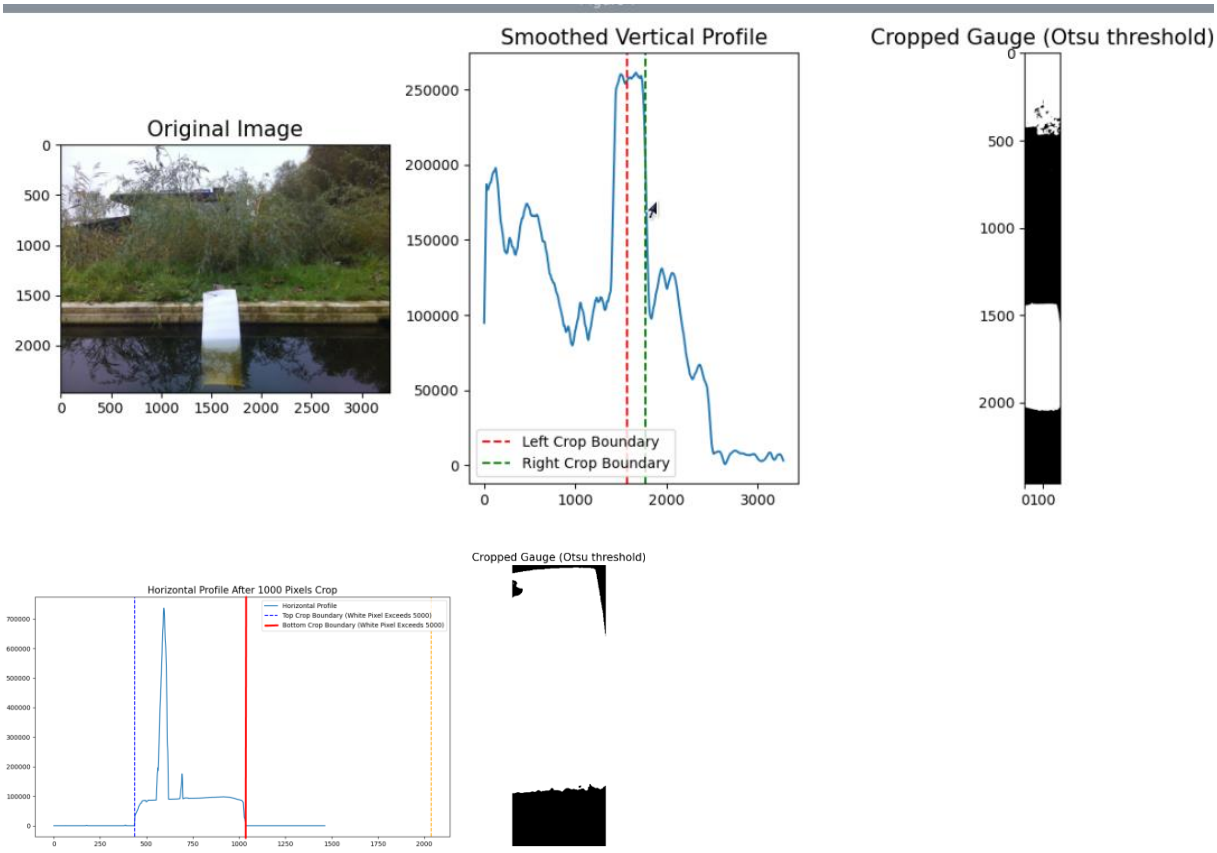
This relatively computationally intensive approach was intended for development and analysis purposes. Future iterations of the product aim to output limited data to conserve power.

Alternative staff gauge detection methods

An alternative approach utilized **peak detection** to identify the staff gauge.

1. The image was cropped to remove the sky.
2. A peak detection algorithm was applied vertically and horizontally to locate the staff gauge.
3. The image was further cropped, and the gauge's vertical pixel count was measured.

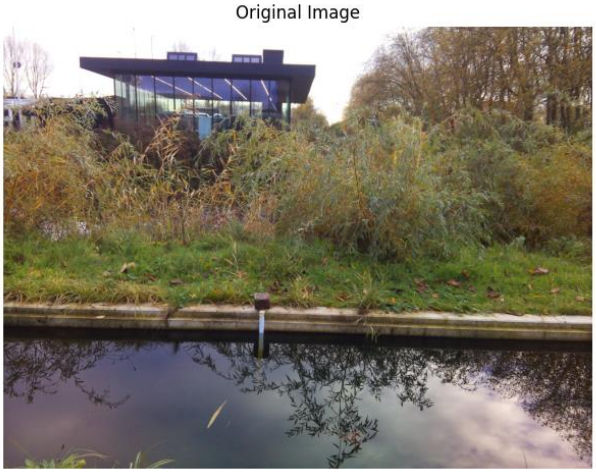
This method required highly accurate thresholding. In cases with significant noise, such as cloud reflections, the cropping and measurements were inaccurate, resulting in unreliable results.



7.2.3 Experiment V2

For this experiment, the gutter was slowly drained over 30 minutes. A timelapse script captured an image every 5 minutes, processed it using the image processing algorithm, and logged key parameters and debugging information into a text file. Images with a blue overlay highlighting the object of interest (staff gauge) were stored in the output folder.

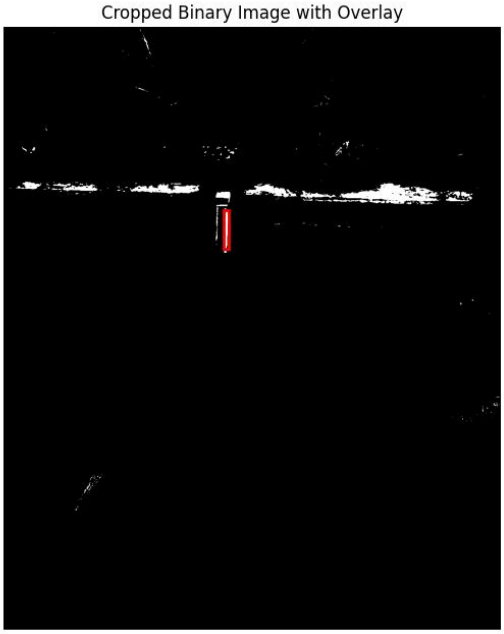
Original Image



Grayscale Image

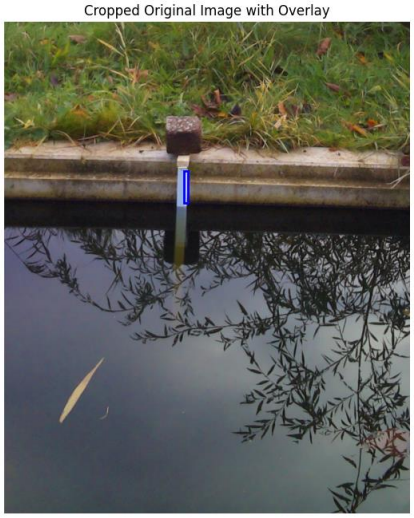
Cropped Grayscale Image

Cropped Binary Image with Overlay



System: pc
Length of the longest vertical object: 79 pixels
Location of the largest vertical object: (Y: 368-447, X: 442-452)
Total script runtime: 6.60 seconds
RUNTIME DETAILS:
Image loading and preprocessing took 0.32 seconds
Thresholding took 0.59 seconds
Finding vertical component took 0.02 seconds
Length of the longest vertical object: 79 pixels
Location of the largest vertical object: (Y: 368-447, X: 442-452)

Cropped Original Image with Overlay



Observations

- Running the script on the Raspberry Pi resulted in a runtime approximately **10 times longer** (around 40 seconds per image) compared to running from pc.
- In the second image taken (2/6), the algorithm incorrectly prioritized a cloud reflection in the water over the staff gauge.
- The dark background of the gutter wall provided good contrast for the staff gauge. However, such contrast may not exist in natural environments.
- Adjustments to the thresholding method were necessary to improve sensitivity. For this experiment, a manual factor of 2x was applied, though this factor heavily depends on environmental conditions, such as lighting and reflection.

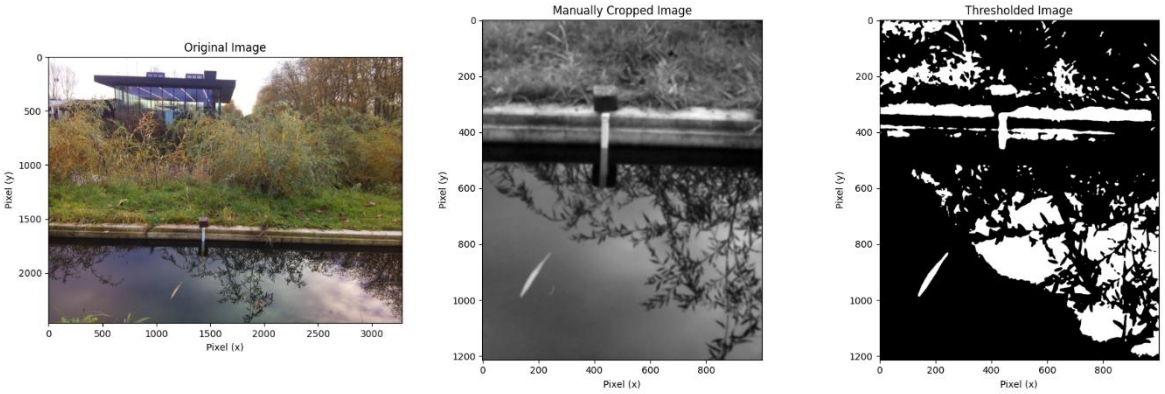


Figure 2: image 1 - threshold factor: 1.2

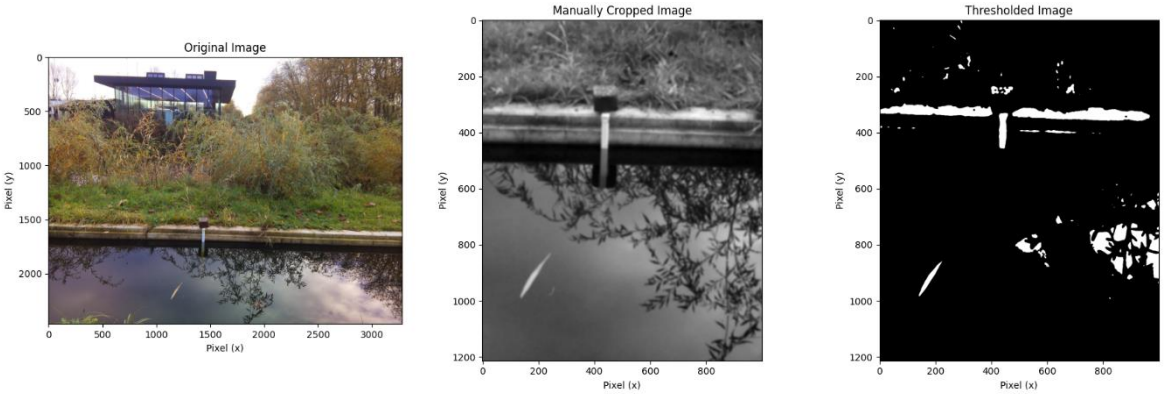


Figure 3: image 1: threshold factor: 1.5

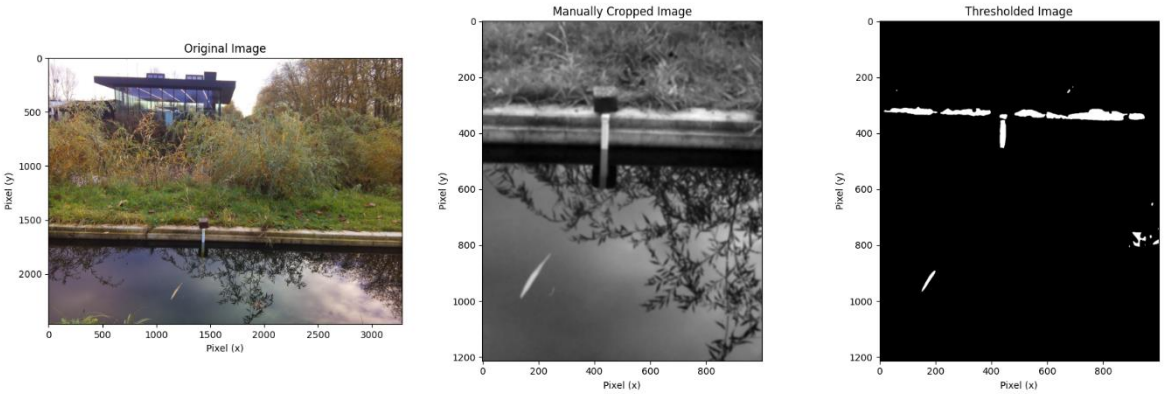


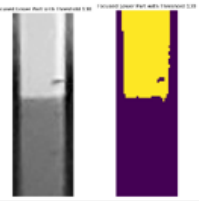
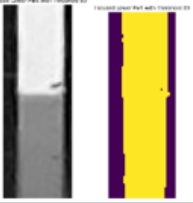
Figure 4: image 1: threshold factor: 2.0

Code adaptations made

To optimize performance of the vertical element algorithm, the entire image is rotated in the preprocessing section of the code so the staff gauge is aligned vertically. This increases the succesrate of finding the longest vertical element as the staff gauge in the binarized image.

To reduce noise like background and reflection on the water surface, the image can be cropped so only the staff gauge is visible. This greatly reduced the noise but introduced new problems. The correct crop for the staff gauge has to be determined manually for every experiment setup. Furthermore, may the camera orientation shift or move due to wind or other factors, the crop would be offset and this would negatively impact the water level detection capabilities.

The way the water level height is being determined by this method is making use of the difference in pixel intensity of the above water staff gauge versus the below water staff gauge. additional testing has been conducted to explore this mechanism. This concluded that the otsu method is not very consistent when detecting the correct waterlevel. The transparency of the water causes the algorithm to sometimes consider the underwater part of the staff gauge as object of interest while it should not.

#	Script output	description
1.		Waterline detected
2.		Waterline **not** detected

The script automaticcaly sumerises all interesting parameters for analysis in a separated file per cycle:

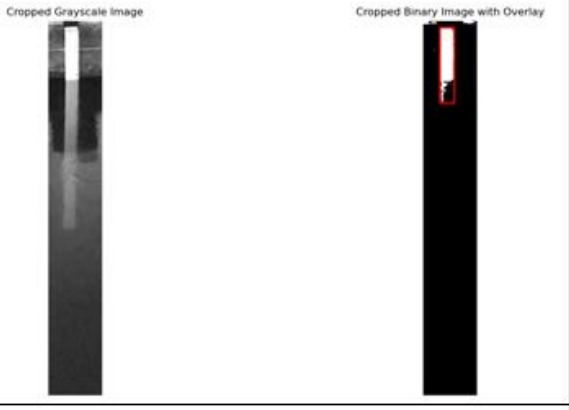
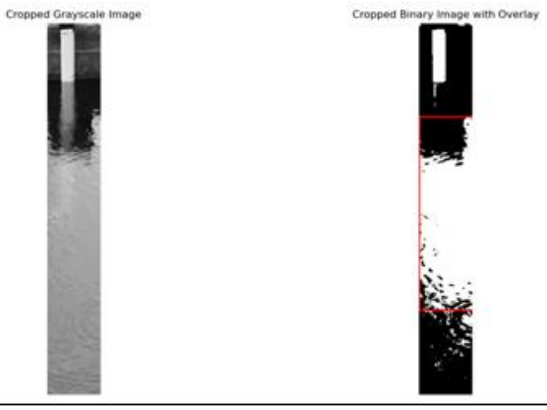
```
SUMMARY REPORT
====================
System: pc
Length of the longest vertical object: 680 pixels
Location of the largest vertical object: (Y: 48-728, X: 0-100)
Total script runtime: 5.10 seconds

RUNTIME DETAILS:
Image loading and preprocessing took 1.07 seconds
Thresholding took 0.03 seconds
Finding vertical component took 0.00 seconds
Length of the longest vertical object: 680 pixels
Location of the largest vertical object: (Y: 48-728, X: 0-100)
Best threshold before iteration: th=150.0
Best threshold found after 16 iterations: th= 75

COMPLETE DEBUG LOG:
====================
System: pc
Processing image: C:\Users\bjorn\Desktop\rpi_mirror\Timelapse_Images\Delft_20241119_154549.jpg
Step 1: Image loading and preprocessing completed.
Step 2: Image cropped.
Step 3: Thresholding completed with best threshold 150.0.
Step 4: Longest vertical component found with height 680 and width 100.
```

Hardware adaptation made

To reduce the glare and reflection on the water surface, the introduction of polarization filter has been tested. Preliminary test has been promising towards noise reduction. Using polarized sunglasses, a simple experiment has been conducted.

Polarized	Regular
	
	

To test the polarization filter, more extensive testing has been conducted in the gutter. Every 10 minutes, 2 images were taken, one with and one without polarization filer, while the gutter was drained to capture different water levels. 4/4 times the polarized setup performed better or ass good as the non polarized setup. Reflection on the water surface is greatly reduced.

Polarized	Regular
	
	

Eventhough this might be prommesing, more extensive testing with larger sample sizes and more variable environmental conditions should be performed to confirm this bias.

7.2.4 Notes on prototype 2 and feedback loop for next prototype

SCAMPER Heuristics

- Substitute: What materials or processes can be replaced?
  - The Otsu method is not ideal for detecting pixel intensity transitions. More options should be explored.
- Combine: What elements can be integrated?
  - A way of capturing images at night should be investigated, while lighting conditions are dark.
- Adapt: How can the concept be adjusted for improvement?
  - Instead of detecting the staff gauge, cropping can be utilized to filter out most of the background noise. A method of detecting intensity changes should be investigated.
- Modify: What aspects (e.g., size or shape) can be altered?
  - A more weatherproof setup should be developed to simplify field testing.
- Put to another use: Can the concept serve other purposes?
  -
- Eliminate: What features can be simplified or removed?
  - Otsu method can be optimized or substituted by a more effective method to detect intensity changes.
- Reverse: Can processes be reversed or re-sequenced?

The improved staff gauge and image capturing capabilities of the current prototype has enabled proper testing. However, the detection method (otsu method) does not seem to be verry effective when it comes down to detecting the water level. With the right manual intervention, it is a useful tool to detect the staff gauge within the natural environment, but this can be simplified by just cropping to the real world coordinates. Testing has made clear that the waterline can be detected making use of the intensity difference between above water staff gauge and below water staff gauge. A more effective way to detect this phenomenon should be explored.

The attachment of an polarized filter are promising. This should be incorporated in future prototype designs. However, a more robust testing setup should be developed to make it more realiable to perform testing outside without putting the electrical components at risk.

7.3 PROTOTYPE V3

7.3.1 Hardware V3

No changes

7.3.2 Algorithm V3

Water level detection algorithm

A alternative to the Otsu method has been developed with a focus on the intensity differences between the dry and wet staff gauge. The requirements for this method are:

- Staff gauge detection
- Water level deterction on the staff gauge as background;

To achieve this, a function is created that logs the mean intensity of a predefined box over the lengt of the staffgauge or image. Using this function, two boxes of x hight are slid down the gauge's image. The mean intensity of the pixels from the top box is compard to the bottom box for every position on the staff gauge. The boxes with the highest contraxt is labelled as the water level.

```python
def calculate_intensity_differences(image_np, box_height=10):
    """
    Calculate the intensity differences between two box_height-pixel-high boxes
    as we slide down the image. Returns the y-coordinate with the max difference.
    """
    height, width = image_np.shape[:2]
    intensity_differences = []

    # Iterate over all possible positions for the top of the first box
    for y in range(height - 2 * box_height):  # Ensure both boxes fit
        box1 = image_np[y : y + box_height, :]  # Top box
        box2 = image_np[y + box_height : y + 2 * box_height, :]  # Bottom box

        # Calculate mean intensity for each box
        mean_intensity_box1 = np.mean(box1)
        mean_intensity_box2 = np.mean(box2)

        # Calculate the absolute intensity difference
        intensity_difference = abs(mean_intensity_box1 - mean_intensity_box2)
        intensity_differences.append(intensity_difference)

    # Find the y-coordinate with the maximum intensity difference
    max_diff_index = np.argmax(intensity_differences)
    max_diff_y = max_diff_index + box_height  # This is the top of the first box
    waterline_y = max_diff_y + box_height // 2  # Shift to the middle between the two boxes
    return waterline_y, intensity_differences
```

This function logs the intensity differences of every box and determines the water line (y-coordinate of the image). the intensity differences can in turn be plotted over the height of the image which results in a graph where the peeks point out the highest pixel intensity difference. This function is adapted so that its unable to result in peek detection where its not probable, such as the border of the image (tests resulted often in a false peek at the border of the image). the second most probable waterline (second highest peek with a minimum distance from the highest peek) is also visualised. This gives insight into what the algorithm is detecting as intensity differences. Often the real waterline and the bottom of the gauge

(slightly visible under water) were labelled as primary or secondary waterline by the algorithm, performinig already much better than the Otsu method.

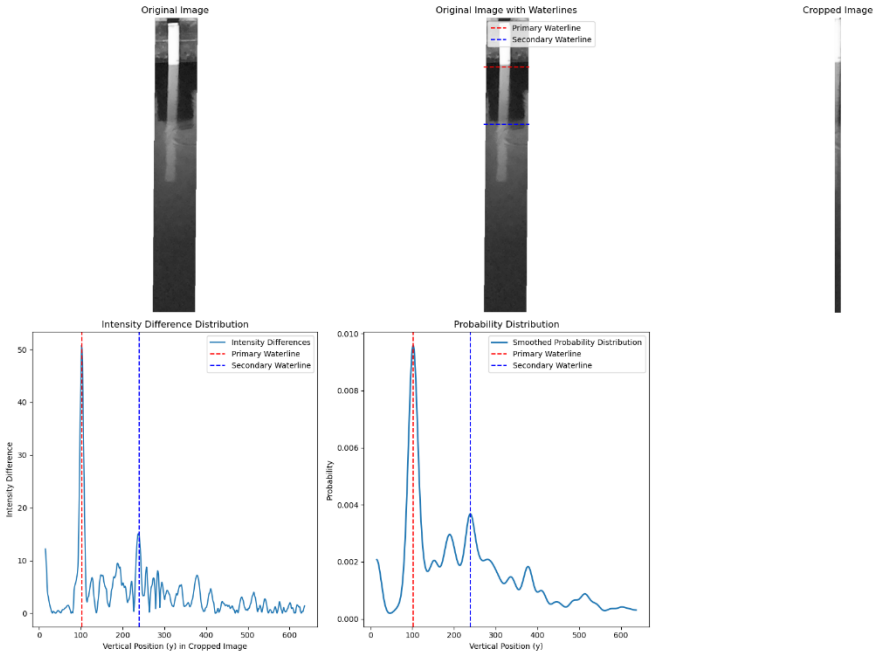
```python
def find_waterlines(intensity_differences, box_height=10, min_distance=50, edge_buffer=10):
    """
    Find the primary and secondary waterlines based on intensity differences,
    ensuring that they are separated by at least `min_distance` and are not
    within `edge_buffer` pixels from the edges.
    """
    # Convert edge_buffer and min_distance to indices in the intensity_differences array
    intensity_differences_filtered = np.copy(intensity_differences)
    intensity_differences_filtered[:edge_buffer] = -np.inf  # Exclude first edge_buffer pixels
    intensity_differences_filtered[-edge_buffer:] = -np.inf  # Exclude last edge_buffer pixels

    # Find the primary waterline (highest intensity difference)
    primary_index = np.argmax(intensity_differences_filtered)

    # Zero out a region around the primary waterline to find the secondary waterline
    low_limit = max(primary_index - min_distance, 0)
    high_limit = min(primary_index + min_distance, len(intensity_differences))
    intensity_differences_filtered[low_limit:high_limit] = -np.inf

    # Find the secondary waterline (next highest intensity difference)
    secondary_index = np.argmax(intensity_differences_filtered)

    # Return the indices of the two most probable waterlines
    if intensity_differences_filtered[secondary_index] == -np.inf:
        raise ValueError("Could not find a valid secondary waterline. Adjust constraints or check input.")

    return primary_index, secondary_index
```

The images are processed by rotating so that the general waterline aligns with the horizontal. This is nessecairy as the image is not transposed to the real world coordinates and the intensity boxes are iterated downwards over the image. furthermore, the image is cropped to the staff gauge as much as possible to filter out noise and converted to greyscale to visulise the intensity of every pixel.

```python
def process_image(image_path, save_path, system="pc"):
    try:
        # Check if file exists
        if not os.path.exists(image_path):
            print(f"ERROR: Image file does not exist at: {image_path}")
            return None

        # Create output directory if it does not exist
        if not os.path.exists(save_path):
            os.makedirs(save_path)

        # Step 1: Load and preprocess the image
        original_im = Image.open(image_path)
        angle = -0.5
        original_im = original_im.rotate(angle, resample=Image.BICUBIC, expand=True)
        original_np = np.array(original_im)  # Keep the original image in color

        # Step 2: Crop the image
        crop_box = (50, 15, original_np.shape[1] - 45, original_np.shape[0] - 45)
        cropped_im = original_im.crop(crop_box)
        cropped_np = np.array(cropped_im)
        crop_offset_x, crop_offset_y = crop_box[0], crop_box[1]

        # Step 3: Convert the cropped image to grayscale for intensity analysis
        cropped_grayscale_np = np.mean(cropped_np, axis=2) if len(cropped_np.shape) == 3 else cropped_np

        # Step 4: Calculate intensity differences on the cropped grayscale image
        box_height = 10  # Must be consistent with calculate_intensity_differences
        waterline_y_in_cropped, intensity_differences = calculate_intensity_differences(cropped_grayscale_np, box_height=box_height)
```

```
# Step 5: Find primary and secondary waterlines
try:
    primary_index, secondary_index = find_waterlines(intensity_differences, box_height=box_height, min_distance=50)
    # Convert indices to y-coordinates in the cropped image
    waterline_y_cropped = primary_index + box_height + box_height // 2
    second_waterline_y_cropped = secondary_index + box_height + box_height // 2
    print(f"Detected waterlines at y={waterline_y_cropped} (primary) and y={second_waterline_y_cropped} (secondary) for {os.pat
except ValueError as e:
    print(f"Error finding waterlines: {e}")
    return None

# Map waterline positions to the original image coordinates
waterline_y_original = map_cropped_to_original_y(waterline_y_cropped, crop_offset_y)
second_waterline_y_original = map_cropped_to_original_y(second_waterline_y_cropped, crop_offset_y)

# Step 6: Create a 2x2 combined plot with the specified format
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
```

Finally the coordinates are transformed from the croppedimage to the original image to enable visualisation plots.



Continuous image subtraction algorithm

(Liu, W.-C. (2024). Evaluation of deep learning computer vision for water level measurements in rivers. Environmental Monitoring and Assessment; paragraph 2.6)

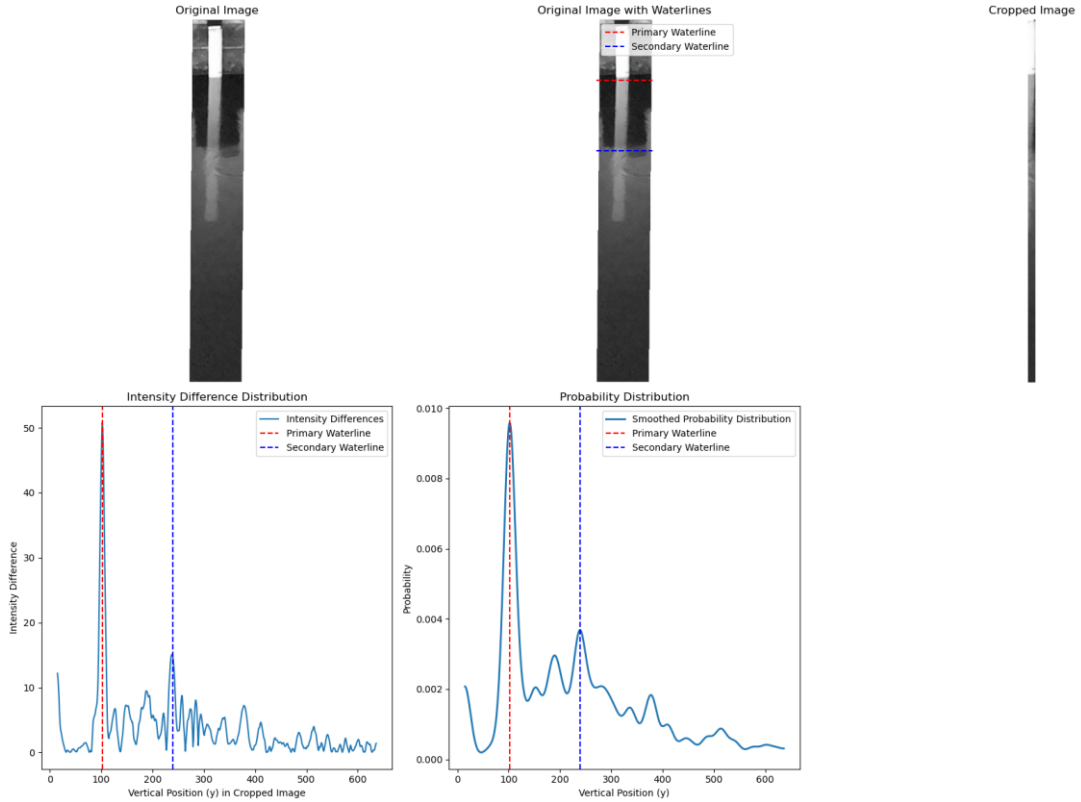
Code to replicate this method is created but the results differ from the work conducted in the article. The article uses time interval of 0.05s for consecutive images. Test images made to test own code are made with an interval of several seconds. This might be the reason that the result is not up to standards.

```
def process_images(img1_path, img2_path):
    """Process two consecutive images to highlight water areas."""
    # Load images as grayscale using Pillow
    img1 = Image.open(img1_path).convert("L")
    img2 = Image.open(img2_path).convert("L")
    print(os.path.basename(img1_path))
    print(os.path.basename(img2_path))

    # Convert images to NumPy arrays
    img1_array = np.array(img1, dtype=np.float32)
    img2_array = np.array(img2, dtype=np.float32)

    # Subtract the images
    diff = np.abs(img2_array - img1_array)

    print(diff)

    # Enhance the difference: Dilate and square pixel values
    enhanced = np.clip(diff ** 2, 0, 255)
    print(enhanced)

    # Binarize the image
    binarized = (enhanced > threshold_value).astype(np.uint8) * 255  # 0 or 255
    print(binarized)
    # Convert the result back to an image
    binarized_image = Image.fromarray(binarized, mode="L")
    return binarized_image

# Iterate through image pairs
for i in range(len(image_files) - 1):
    img1_path = os.path.join(image_dir, image_files[i])
    img2_path = os.path.join(image_dir, image_files[i + 1])

    # Process images
    output_image = process_images(img1_path, img2_path)

    # Save the processed image
    output_image.save(os.path.join(output_dir, f"processed_{i+1}.png"))

print(f"Processed images saved to {output_dir}.")
```

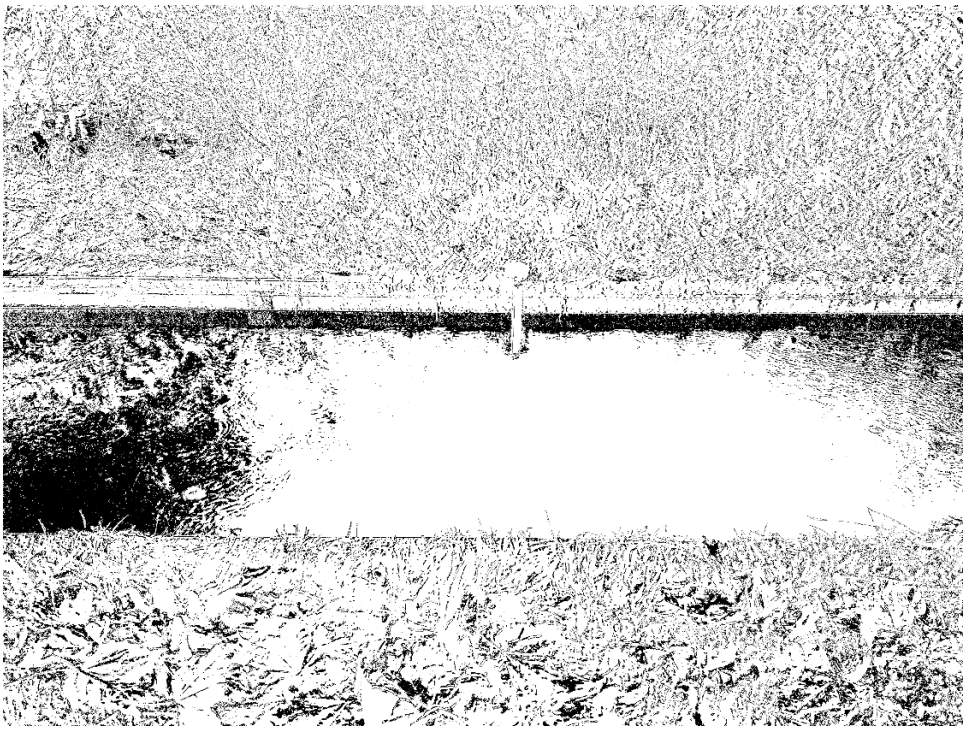
7.3.3 Experiment V3

Water level detection algorithm

The intensity box method works much better compared to the Otsu method. Cropping to the staff gauge greatly reduces background noise but contributes to more work setting up the system. The crop dimentsions have to be determined manually (for now) and precousions have to be made so that the camera orientation does not change (by winds or other environmental impacts).

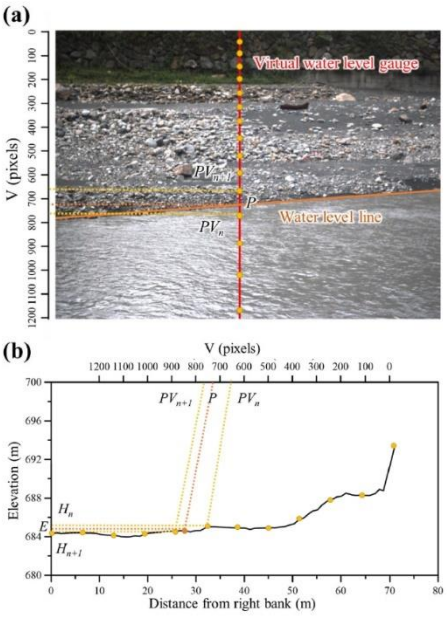


CIS



The method takes the difference between pixel intensity and dilates the difference. After binarization it reconstructs the images showing difference between dynamic and static pixels. The goal is to locate the static bank and dynamic water, but a lot of factors play a role in this workflow. Natural surroundings like vegetation or water puddles on the bank result in noise. Also camera orientation has to be very secure to make sure each pixel between the frames are compared with the same pixel of the previous orientation, linked to the same coordiantes in the real world.

- Interval of consecutive images needs to be smaller to decrease variables other then water dynamics such as moving grass, change in illumination / shading or glare.
- Method works best if riverbank is made up out of stone/earth instead of moving elements like vegetation
- Method works best when water velocity is high



- Method can be paired with digital water level gauge
- Noise can be reduces using an rolling average. **Updated Processing Pipeline (method not tested yet) :**
    - Compute a rolling average of N consecutive frames to smooth out transient changes like grass motion.
    - Subtract the current frame from the rolling average.
    - Enhance and binarize the resulting image to highlight water areas.

7.3.4 Notes on prototype 3 and feedback loop for next prototype

The water level detection method seems promising. This method can be integrated in the device workflow, combining timelapse imaging with automatic processing. The CIS method is probably not best choice for this setup. The raspberry pi is unable to consistently take short interval images [0.05s]. as of now, the device is a raspberry pi connected with an camera module, held together on a cartwood box and some ducttape. It would help to create a more weather proof casing so test are les risky. Problems with damaged camera modules also point to better protection for the sensitive parts of the system. Finaly, different filters can be tested to increase the ability to detect the waterline based on intensity differences. A high quality polarization filter can be utilized and a ir bypass filter can be tested for night measurements. Night measurements also point towards the need of IR illumination.

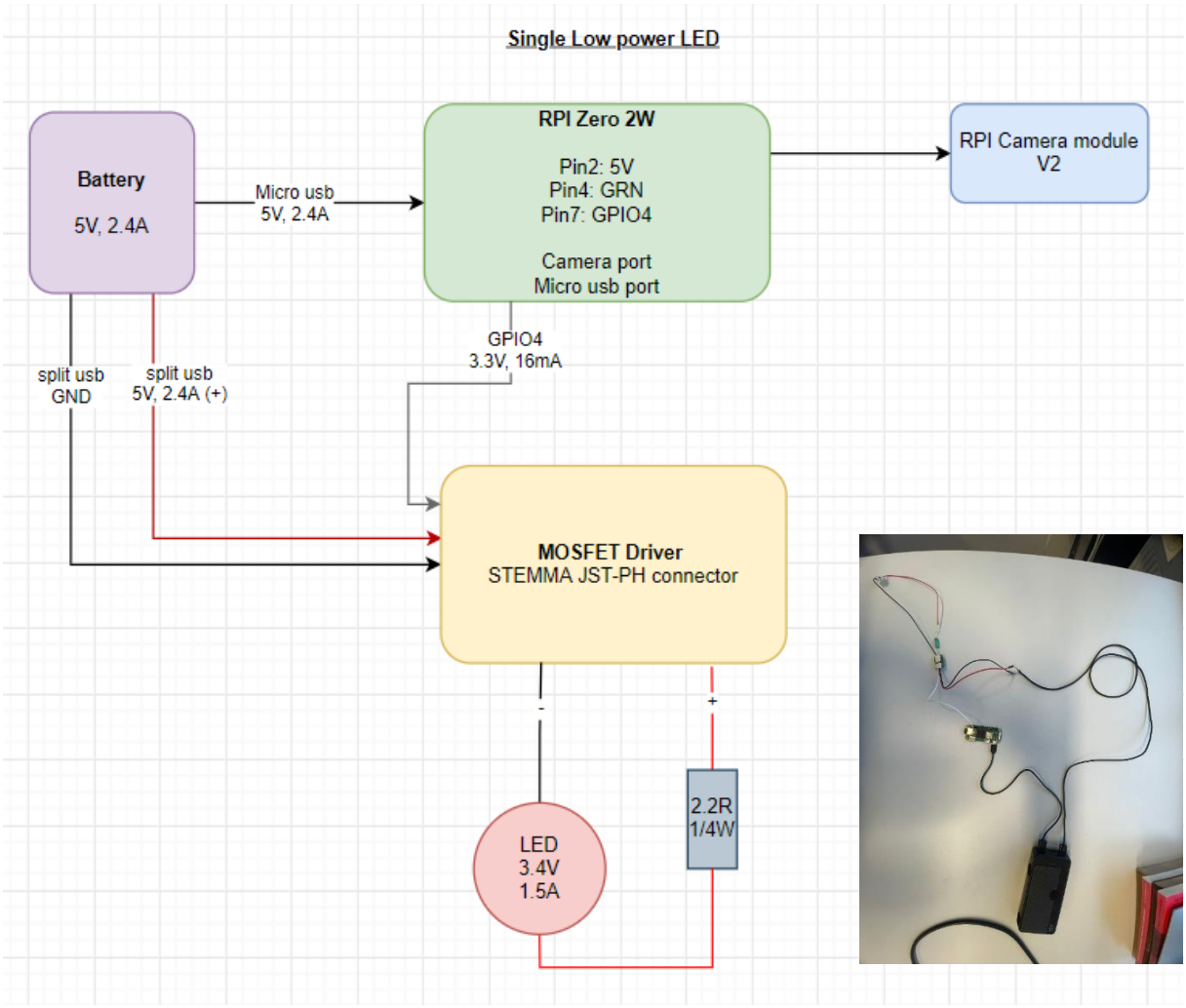
SCAMPER Heuristics

- Substitute: What materials or processes can be replaced?
  - Device casing has to be more practical for experimenting and somewhat waterproof to reduce risks during outside testing.
- Combine: What elements can be integrated?
  - All components of the device should be integrated in a simplistic casing forming the first practical prototype.
- Adapt: How can the concept be adjusted for improvement?
  - Filters such as polarization filters and IR bypass filters can be introduced to the system, together with an IR illumination method.
- Modify: What aspects (e.g., size or shape) can be altered?
  - To connect every component, a breadboard can be used to help with closing the circuit and making sure every component has a safe current and voltage. A USB hub can be added to enable modifications in the script during field tests because this enables the use of a mouse and keyboard.
- Put to another use: Can the concept serve other purposes?
  -
- Eliminate: What features can be simplified or removed?
  - The otsu method can be eliminated, together with the very preliminary casing of the system which mostly consisted of cartwood box and ducttape and elastic bands.
- Reverse: Can processes be reversed or re-sequenced?

7.4 PROTOTYPE V4

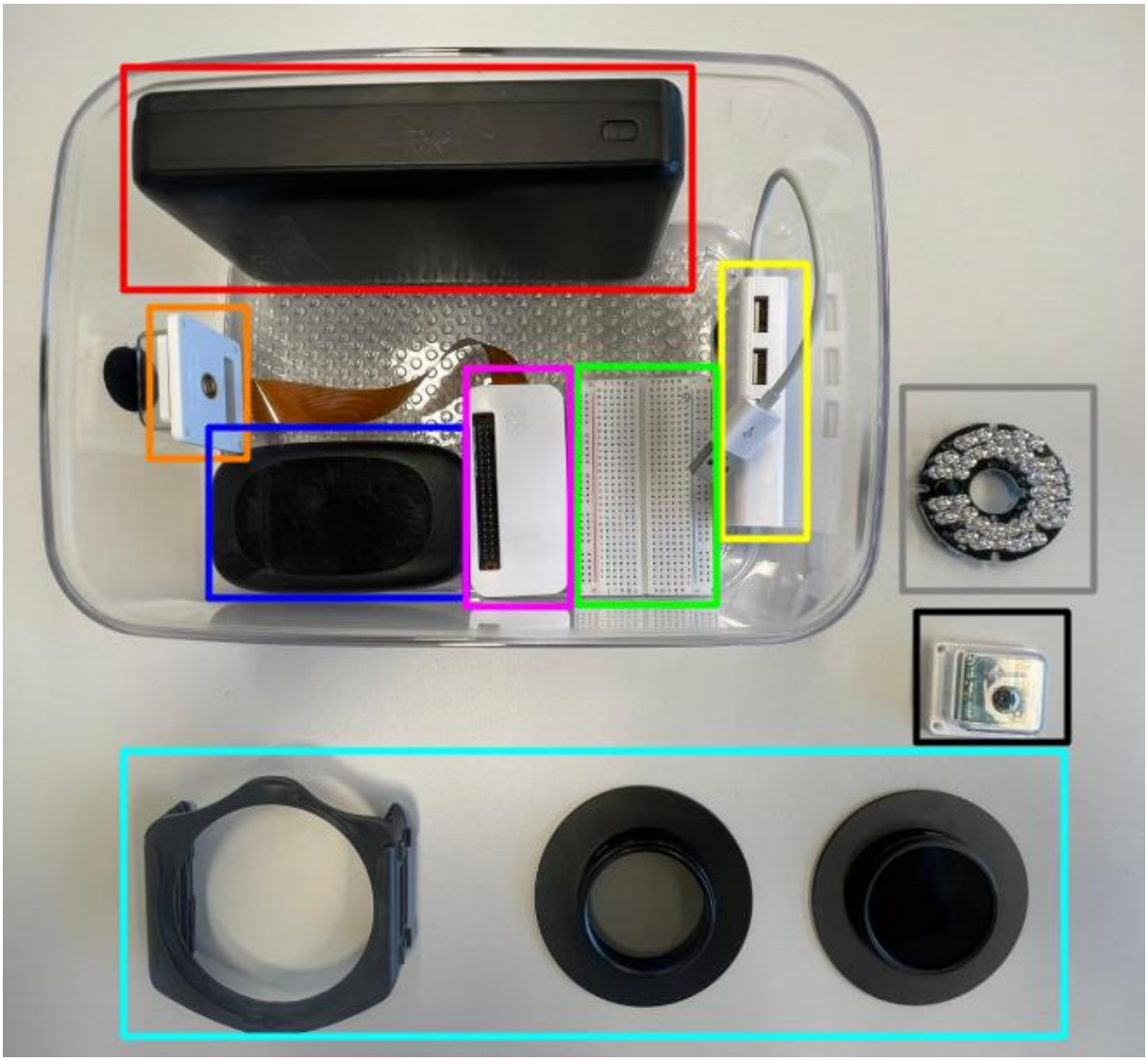
7.4.1 Hardware V4

To power a IR led, a mosfet driver is used as transistor. This is required as the raspberry pi gpio can only output a low current signal, and not power the led by itself. The led needs to be powered separately. To save power, the led only turns on at night and during the image capturing. So realistically its only turned on for 1 or 2 seconds every cycle (cycle being every 10 minutes usually).



The led is protected by a resistor (1.2Ohm, 5W), and powered separately form the powerbank. The powerbank has a shared capacity but is able to output 2.4A 5V from each output.

Polarization filter and an 850nm bypass filter has been acquired together with a filter holder used in photograpy. These filters are relatively cheap and easy to acquire. A 12V led ring has been purchased from Ali express, able to output 0.3A * 12V = +- 4W. however this cant currently be powered as the current power supply is only capable of outputting 5V.



- Box
  - Temporary prototype
- **ISY Powerbank**
  - 20.000 mAh
  - 2x usb output [5V, 2.4A]
- **Rpi NoIR camera module V2**
  - 8 MP
  - 3280 x 2464 pixel static
  - 1080p30 video
  - No IR filter
- **Tp-Link M7200 portable router**
  - 2.000 mAh (charge by battery)
- **Rpi Zero 2W**
  - 5V input
  - 3.3v, 5V output
  - 512MB RAM

- **Breadboard**
- Usb / ethernet port
  - For connecting mouse and keyboard to RPI if VNC fails
- **IR LED ring**
  - 12 V (not suitable for this setup)
- **Rpi camera module V2**
  - Optional for daytime image capturing
  - 8 MP
  - 3280 x 2464 pixel static
  - 1080p30 video
- **Filter holder + Polarized filter + 850nm pass trough filter**
  - Polarized filter for reducing noise from glare on water surface
  - IR pass trough filter for reducing noise from light pollution at night

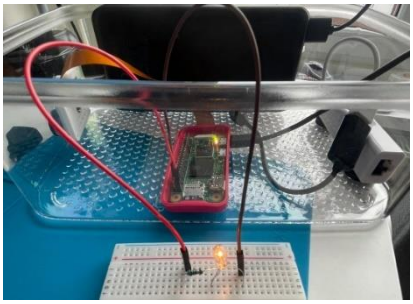
A food container is used to act as a temporary casing for the project. Currently the focus is on daylight capturing and night time capturing, where during the night, the IR led has to illuminate the project area to be able to detect the water level.

Daytime Capturing
The current setup is capable of capturing bursts of images at predefined intervals during the daytime. The first image of each burst is stored in the local memory and processed by the waterline detection algorithm. The results of the algorithm are also stored locally. Subsequently, the device uploads the raw image and the detected water level to the OpenRivenCam server. Finally, the program enters sleep mode until the next predefined cycle (every 10min). A high-quality polarization lens is available to reduce noise caused by glare on the water surface.

Night-Time Capturing
Currently, the prototype cannot process images at night. While a NoIR camera is operational, the IR LED required to illuminate the project location is not yet functional. However, a lens to filter out light waves shorter than 850 nm is already available to minimize noise from other light sources.

Project Difficulties at This Point
Desk research suggested using a 12V IR Led ring to illuminate the project area at night, considering that the object of interest is outdoors and approximately 5-10 meters from the camera. However, this would require an additional power source, as the current one only outputs 5V. Also switching to a 12V setup would contradict the criteria of being low power.

Preliminary experiments with simple LEDs powered via the GPIO pins of the Raspberry Pi encountered challenges. The GPIO pins are designed to transmit utility signals, such as turning LEDs on or off, but are not ideal for powering features. However, powering an LED using the constant output pin did work.



What's Next for the Coming Week?
To enable night vision, we plan to obtain a 5V IR led ring to illuminate the project area at night. This IR Led ring will be powered directly by the Raspberry Pi, eliminating the need for an additional power source. A MOSFET driver will be used to control the Led ring, ensuring it only turns on during the image-capturing bursts (2 seconds every 10 minutes) to reduce power consumption and minimize disturbance to wildlife.

To determine the required **W/sr (watts per steradian)** for your **3.3V, 1.5A IR LED** to illuminate a water body at a distance of **5 meters** with a beam angle of **50°**.

Required radiant intensity

$$\boldsymbol{I} = \boldsymbol{E} \cdot \boldsymbol{d}^2$$

I = radiant intensity (power per steradian needed to achieve sesired irradiance at specified distance)

E = minimal detectable irradiance (not specified by rpi, estimate at least 0.1 [W/m2]

D = distance = +- 5 [m]

$$I = 0.1 \cdot 5^2 = 2.5\, W/sr$$

Required power

$$\boldsymbol{P = I \cdot \Omega}$$

P = total power [W]

Ohm = Solid angle [R]

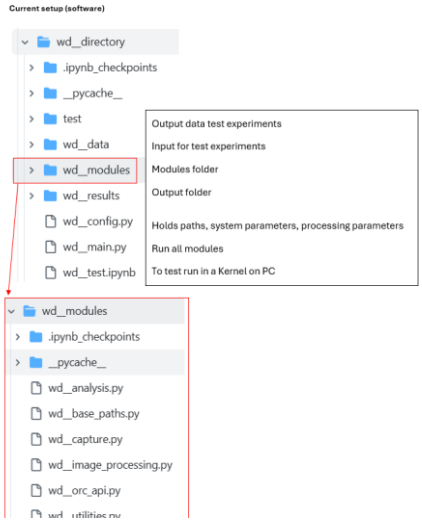
$$\boldsymbol{\Omega = 2\pi(1 - cos(\theta / 2))}$$

$$\Omega = 2\pi(1 - cos(50° / 2))$$

$$P = 2.5 \cdot 0.588 = 1.47\ W$$

7.4.2 Algorithm V4

Organised code and enhanced user friendliness of script. no functionality changes were made to the code.
The algorithm is restructured using different .py files and organised in folders. This enables better work dynamics and code can be more easily adapted.



7.4.3 Experiment V4

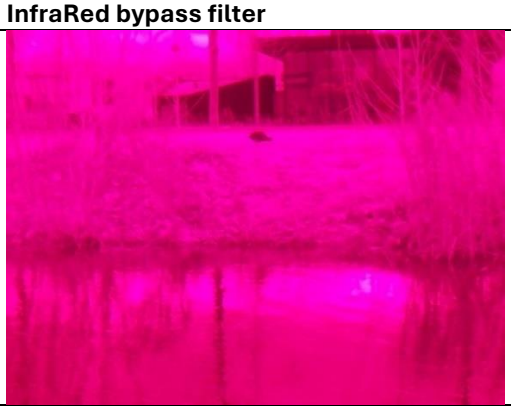
IR light testing

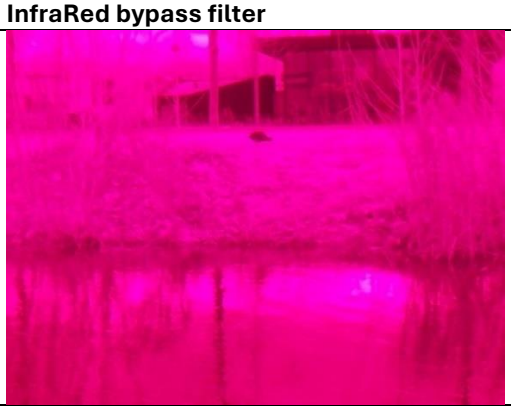
The IR Led seems to be working as it is able to illuminate a small dark room. However, the intensity of the LED seems too low to illuminate a waterbody in a natural environment. For now, one LED is connected in the system. The powerbank that is used as a power source at the moment has two outputs both capable of emitting 2.4A of current and 5 Volts. This makes it non logical to connect the led's in series, as the current would be shared across the series and would not achieve the leds maximum output (1.5 A), as the 2.4A would be shared between them (max 1.2A). However, a test experiment could be setup that tests one led on max capacity (1.5A) and two leds on reduced capacity (1.2A). A beam casing that focuses the IR Light on the target area could be interesting for testing as well and could be combined with the function of keeping the filters dry.

After experimenting it seems that the single led only draws 1A on ~3 volts, which mean the led only emits with 3Watts. According to preliminary estimates, this should be enough but it is not close to illuminating the project area enough. For night measurements, upscaling of the IR led is necessary.

Filter testing

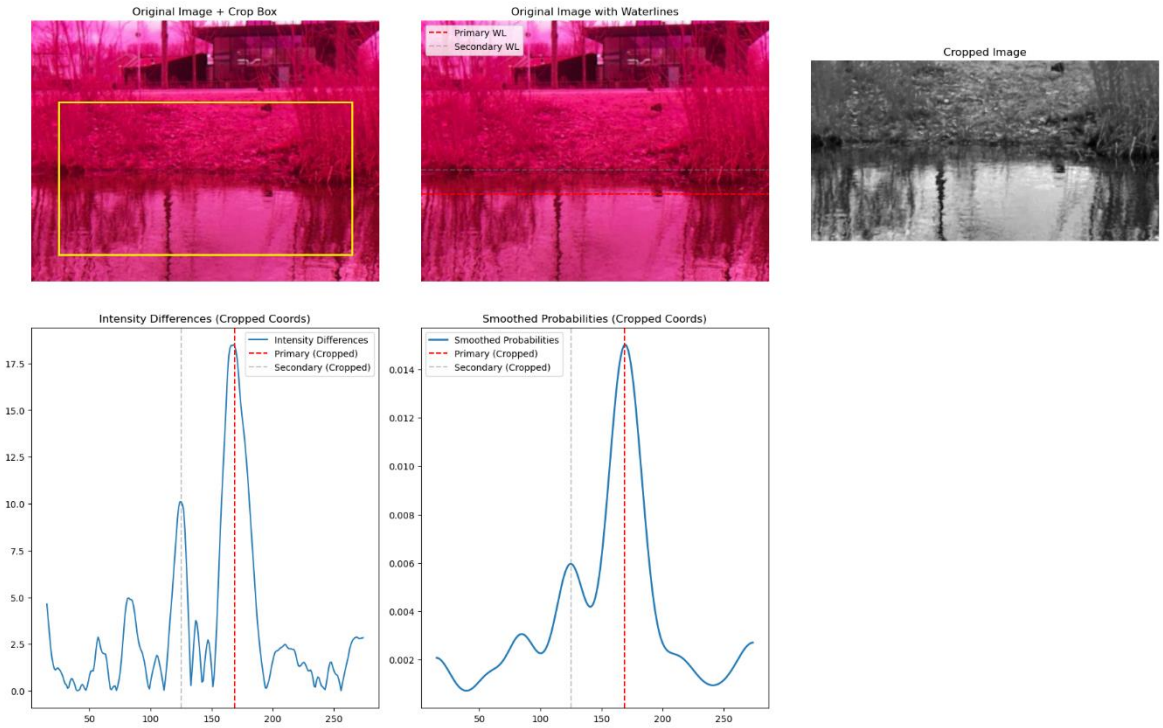
Different combinations of filters have been experimented with. The experiment took place in the afternoon about 14:00 while it was cloudy and dark. The polarization filter should reduce the noise from reflections on the water surface. The IR bypass filter should only pass +850nm wavelength. This should stabalize the image as it only receives the light that the IR Light is emitting and reduce noise from other light sources at night time. During the experiment a NoIR camera module was utilized.

No filter	**InfraRed bypass filter**
	
The contrast of the image utilizing no filters is the highest of all the experiment setups	The contrast decreases when using the IR filter during daylight
IR + Polarization	**Pol**
	

The contrast decreases from the IR filter, the polarization filter smooths out the white reflection noise a little	The white reflections on the grass and water surface are smoothed out but the contrast is decreased compared to no filter.
No filter	**InfraRed bypass filter**
	
The contrast of the image utilizing no filters is the highest of all the experiment setups	The contrast decreases when using the IR filter during daylight
IR + Polarization	**Pol**
	
The contrast decreases from the IR filter, the polarization filter smooths out the white reflection noise a little	The white reflections on the grass and water surface are smoothed out but the contrast is decreased compared to no filter.

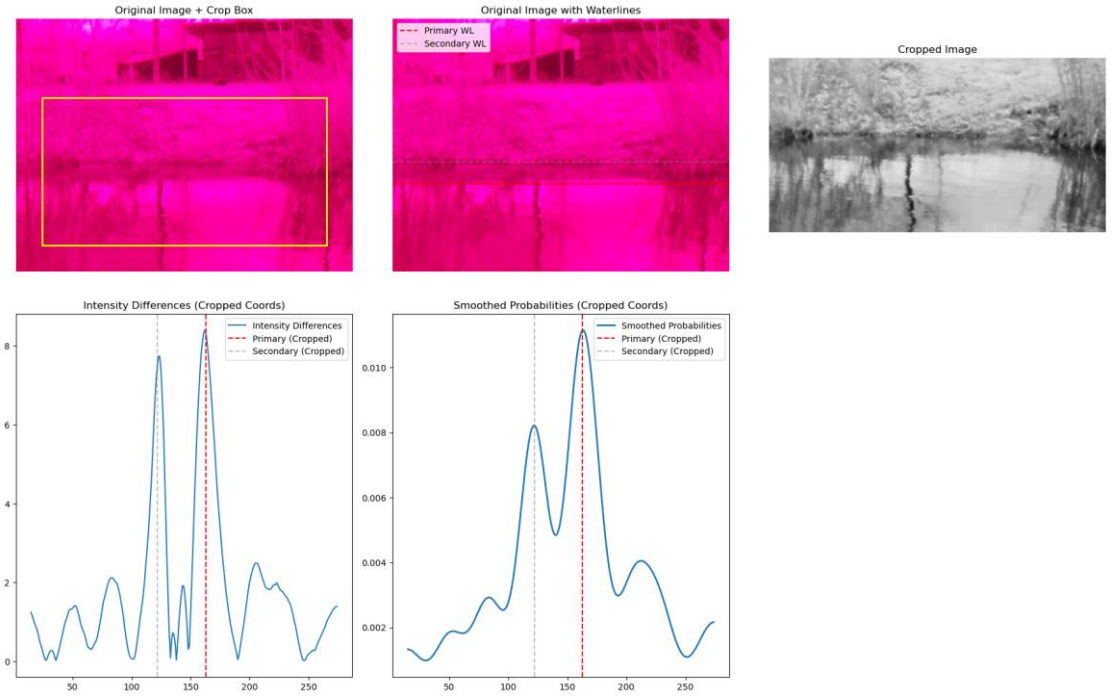
None

Secondary waterline is mostly correctly placed on real waterline. But the contrast of the reflection /shadow of the slope have higher contrast. The contrast peeks on 10.0 & 17.5.



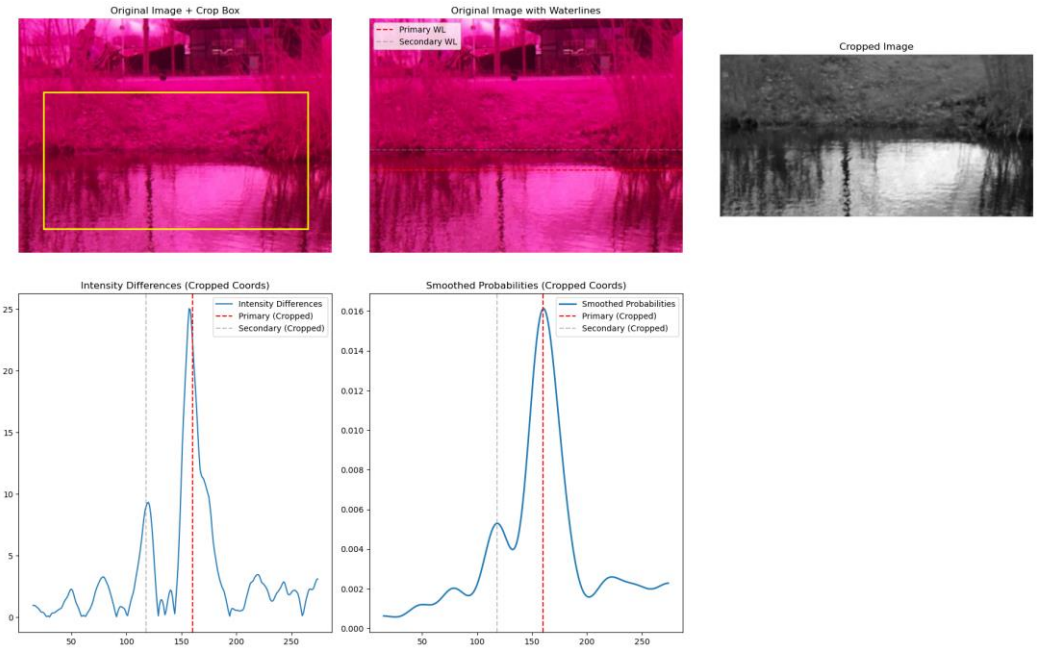
IR

The IR filter image catches the same contrast peeks, however the contrast seems lower overall. The contrast peeks only at +- 8.0.



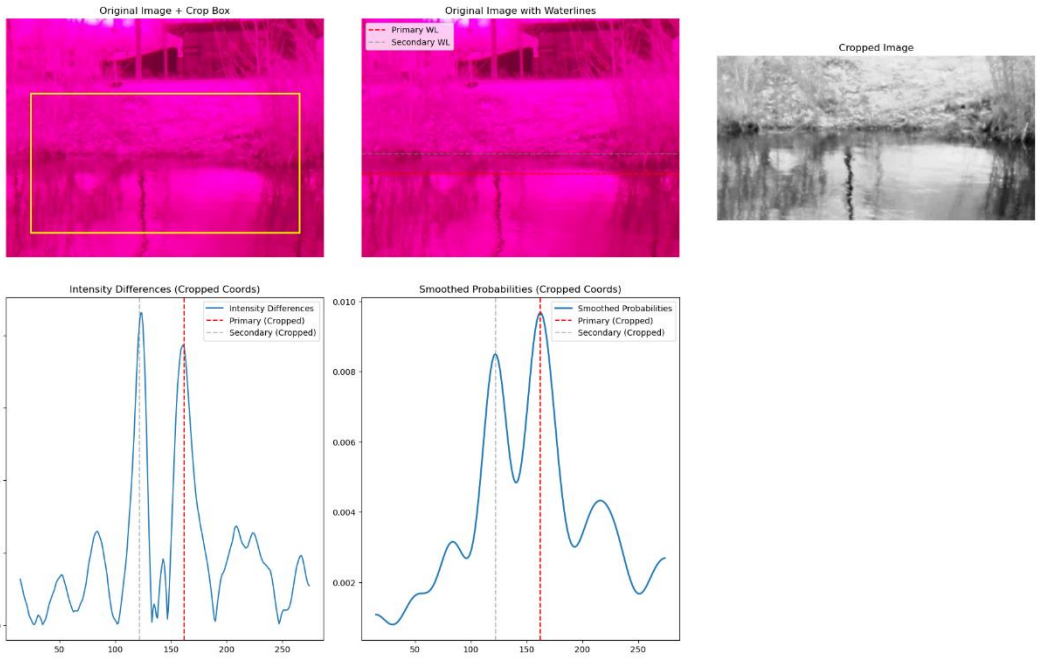
Polarized

The same results occur using the polarized filter, however the contrast is higher than the original image peeking at 10.0 and 25.0.



Polarized & InfraRed

Both filters result in the lowest contrast and the primary and secondary peeks are very similar in intensity. Both peek around 8.

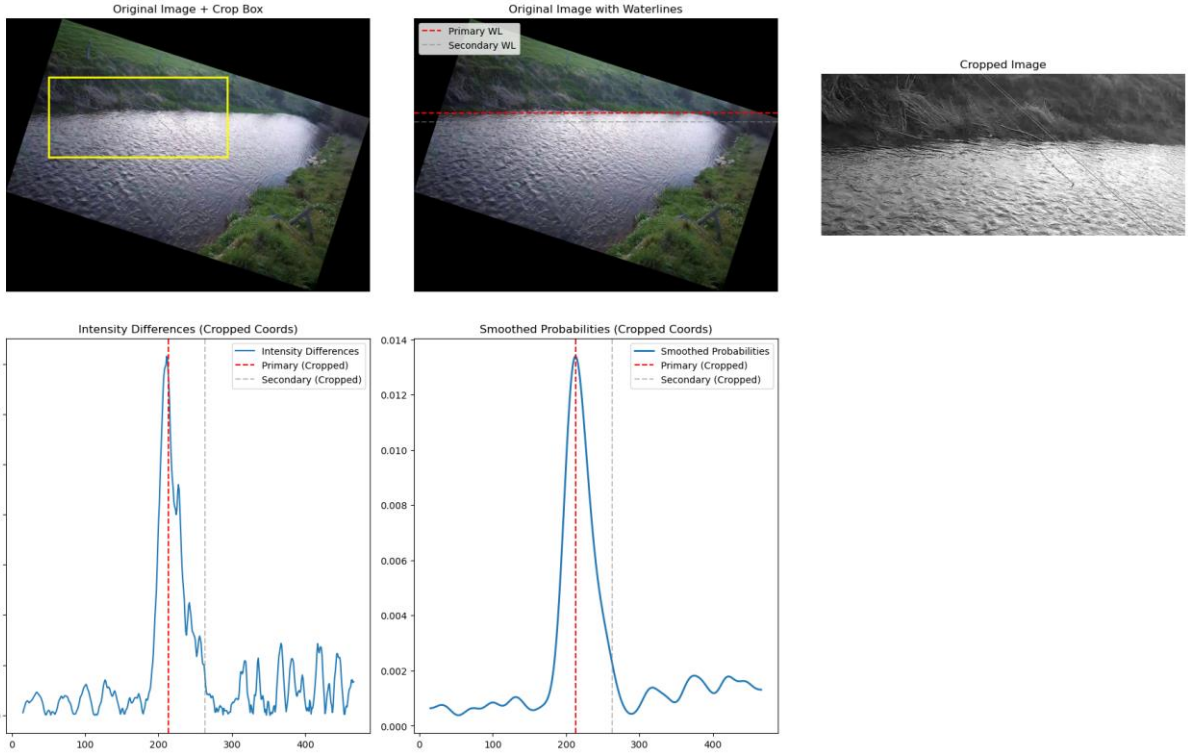


Overall, the polarized images have the highest contrast over all images. However, more testing should be done to test the consistency of this behaviour on different times and illumination conditions. Also, a very wide intensity comparison box is taken resulting in a diffusive contrast. Taking a narrower comparison box should result in a more distinctive intensity difference.

7.4.4 Test data hommerich

A dataset of 11.146 images taken of a stream in Hommerich, Limburg is provided for analysis of the algorithm. The images were taken from December 2022 until April 2023 every hour. At night images were taking using a flashlight.

A random selection of 100 images (taken during daylight without flashlight) were selected and run trough the current algorithm. Preprocessing steps that were taken were rotation of -18 degrees, so the waterline mostly aligned with the horizontal. This way, no further transformations had to be made to run the algorithm of water level detection.



In hindsight, a too wide of an intensity comparison box is selected to determine the water level. A stretch of +- 5 meter is cropped of the original image to run trough the steps of the algorithm, which smooths out the pixel intensity of the comparison boxes by a lot. Next time, a narrower slice should be selected in the preprocessing process.

The statement that should guarantee the secondary waterline to be at least 50 pixels away from the primary waterline caused that the secondary waterline was not mapped to a peek but just the highest value at least 50 pixels away from the primary waterline, this was fixed later.

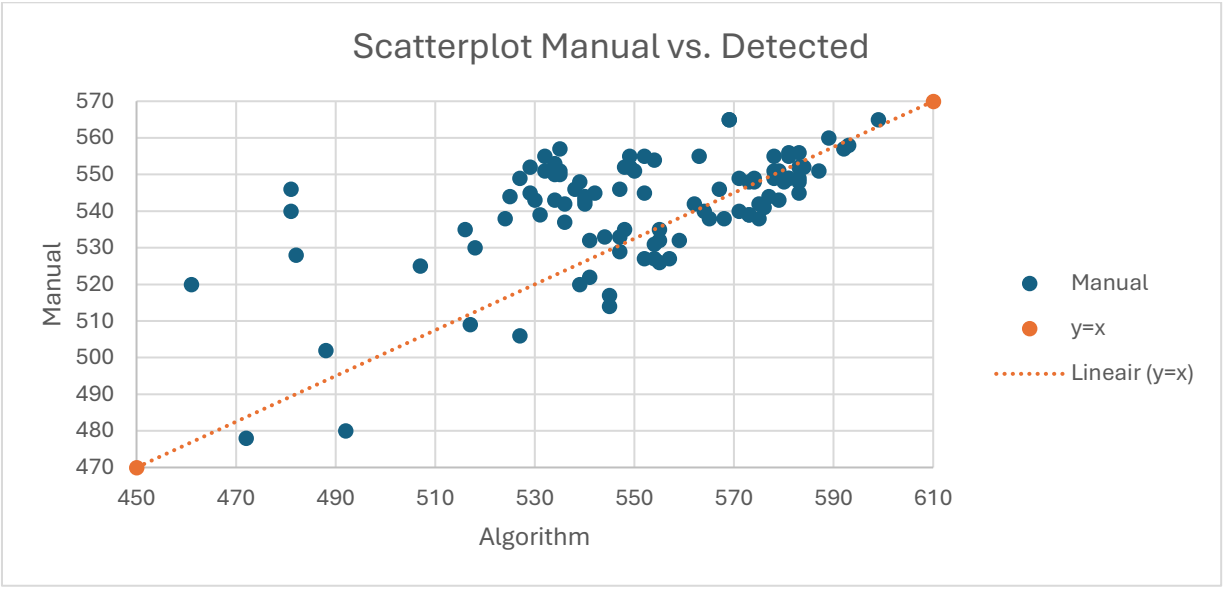
Using a script, an effort was made to map the real waterline to compare the algorithm outputs to. This was accomplished by manually clicking two points in the image. The mean y coordinate of this line is labelled as the real water level (for now). This pixel y coordinate can be compard to the detected water

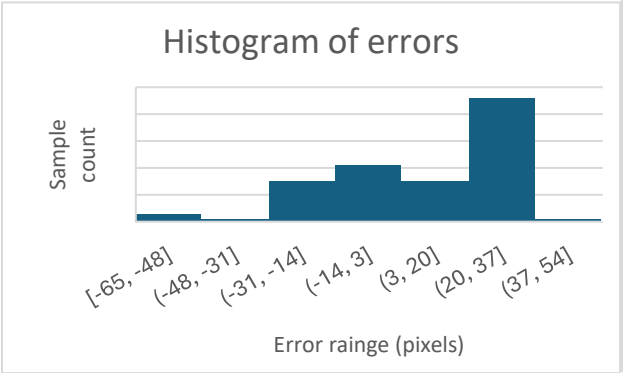
level from the algorithm. After visual analysis, this way of determining the real water level is way to inconsistent an inaccurate to realy derive any conclusions. However, the metohod of comparing the detected waterline with the 'real' water level is developed.

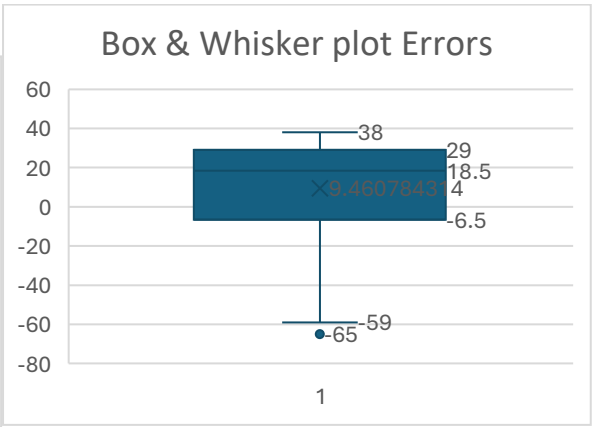
image	Algorithm	Manual	Error
schedule_20220831_045046.jpg	569	565	4
schedule_20220831_062436.jpg	549	552	-3
schedule_20220831_045046.jpg	569	565	4
schedule_20220831_062436.jpg	549	552	-3
schedule_20220831_102356.jpg	593	558	35
schedule_20220831_102646.jpg	589	560	29
schedule_20221220T085601.jpg	599	565	34
schedule_20221228T074600.jpg	584	552	32
schedule_20221228T123100.jpg	587	551	36
schedule_20221229T080100.jpg	548	552	-4
schedule_20221229T124600.jpg	583	552	31
schedule_20221230T080101.jpg	592	557	35
schedule_20230105T123101.jpg	539	548	-9
schedule_20230106T104601.jpg	540	543	-3

...

MAE	RMSE	Bias
21.38235	3.075839	9.460784



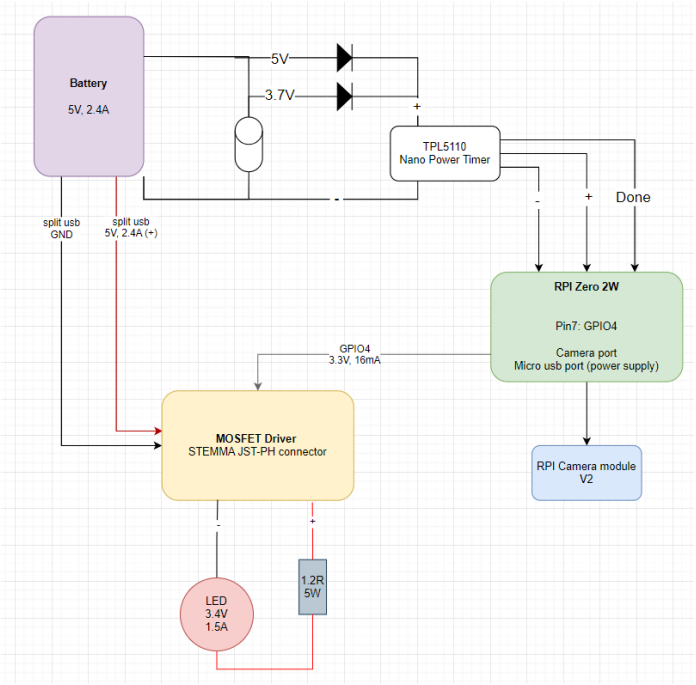




7.4.5 Power consumption experiment

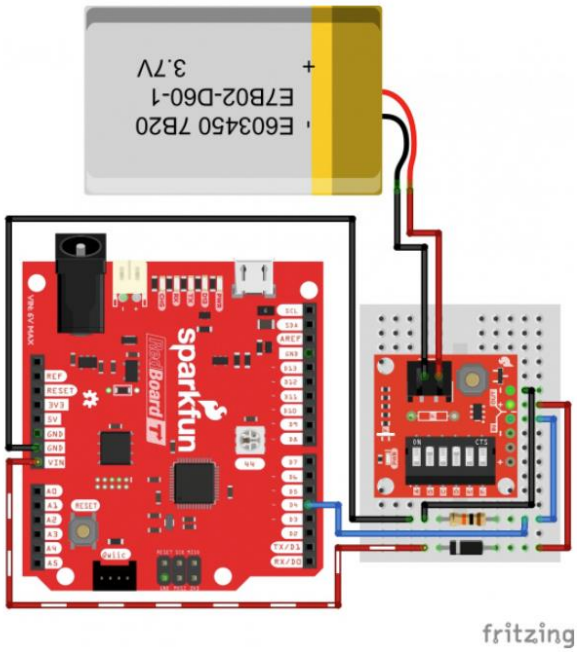
The TPL5110 nano power timer is a board that controls power consumption of a system. It runs at +- 85 nA when in standby mode and controls the delay of which the device can stay in standby mode. When it gets a done signal, it turns power from the powersource off for x amount of time. X is regulated by a series of resistors.

The first tryout was to connect the main power source and an separate power source for the timer. This however resulted in instable voltagges to the pi (between 4.2 and 5.0). When the pi goes below 4.7V it has difficulties booting up. The boot up process is repeated indefinitely. See figure below for board setup. The reason for this instability is the use of diodes, that regulate the direction of the current. The 5V current cant flow in the opposite direction of the current from the battery, hence the diode is used. However, the diode has a forward voltage drop of +- 0.8V, which probably caused the instability.



The second try was to connect the nanotimer directly between the main power source and the pi. However, also this setup was very inconsistent. While introducing a power measurement tool (multi tool or usb power measurement tool), the nano timer shuts down and therefore shuts down the boot up of the pi. The Voltage is stable at the input pins of the timer, but gets unstable after it passes the internal mosfet of the timer board and get distributed trough the jumper wires.

The cause of the instability within the timer could be the result of short circuiting of some sort because during testing, some smoke and burning smell was detected. 2 new times are bought to test this setup and the voltage instability.



findings

- The powerbank has a powersaving mode that shuts down the powerbank after low power demand (80nA is low power) after about a minute. This makes it so that a boot up after standby is impossible
    - A workaround is connecting an usb hub in between the power source and the nano timer. The little led in the hub keeps the battery on while in resting mode. This takes power but for now it's a good way of testing the nano timer.
- When the delay is set < the boot up time, the nanotimer shuts itself down before getting the done signal. It keeps booting in cycles but never finishes.
- N14007 diode has Forward Voltage drop of 0.8V, which causes instability in power supply which results in insucessfull boot up of pi.
- The pi internal timer starts 30 seconds after turning on the power supply. When a rest time of 120 seconds is programmed before sending the done signal, it wil take +-150 secons to shut down after turning the power on.

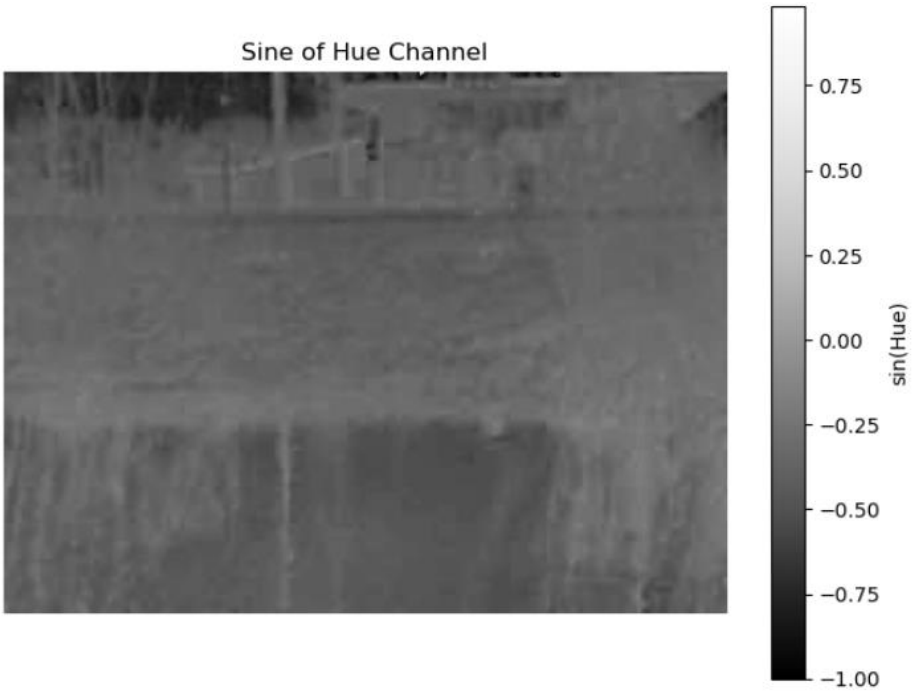
Nano power timer seem to be working if only using a simple script that produces a print statement and a resting state before forwarding the done signal.

7.4.6 Colour schemes experiment

NOIR camera

This is the original image displayed in RGB format. Each pixel consists of red, green, and blue values that together define its color.

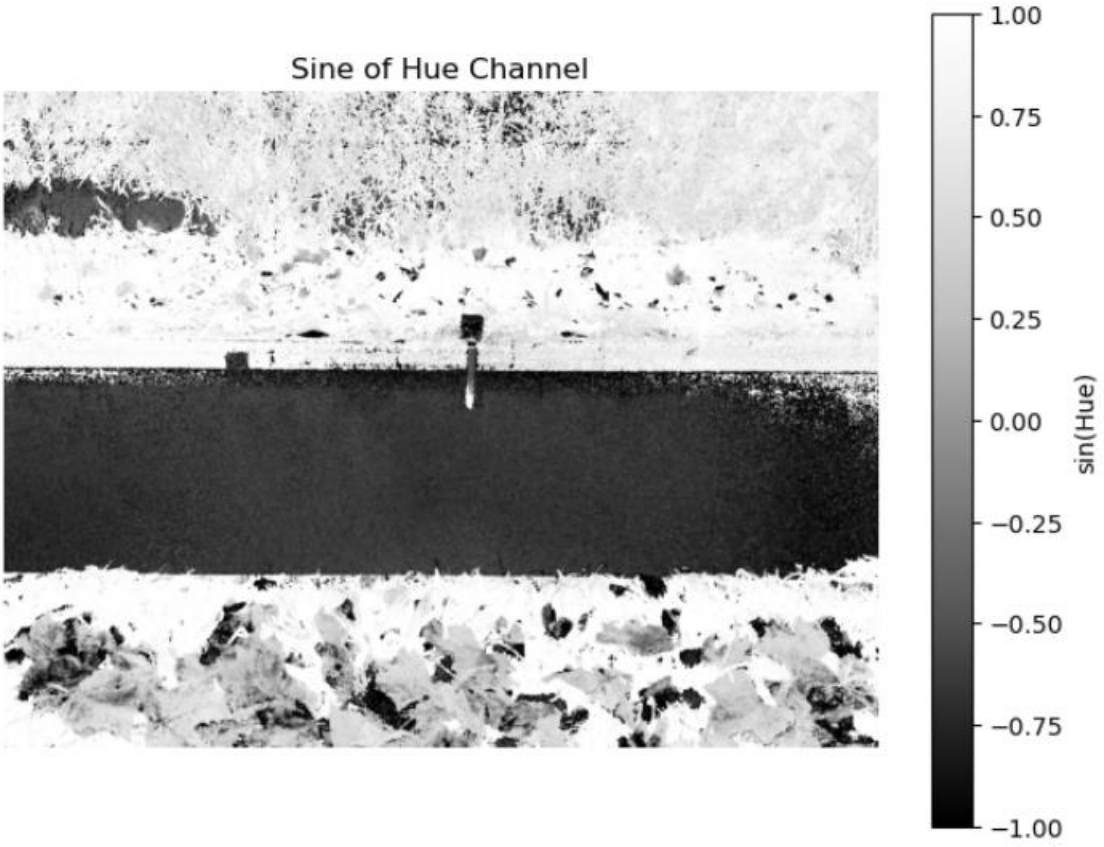
The Value (V) channel represents the brightness of each pixel. Higher values correspond to brighter regions, while lower values indicate darker areas.

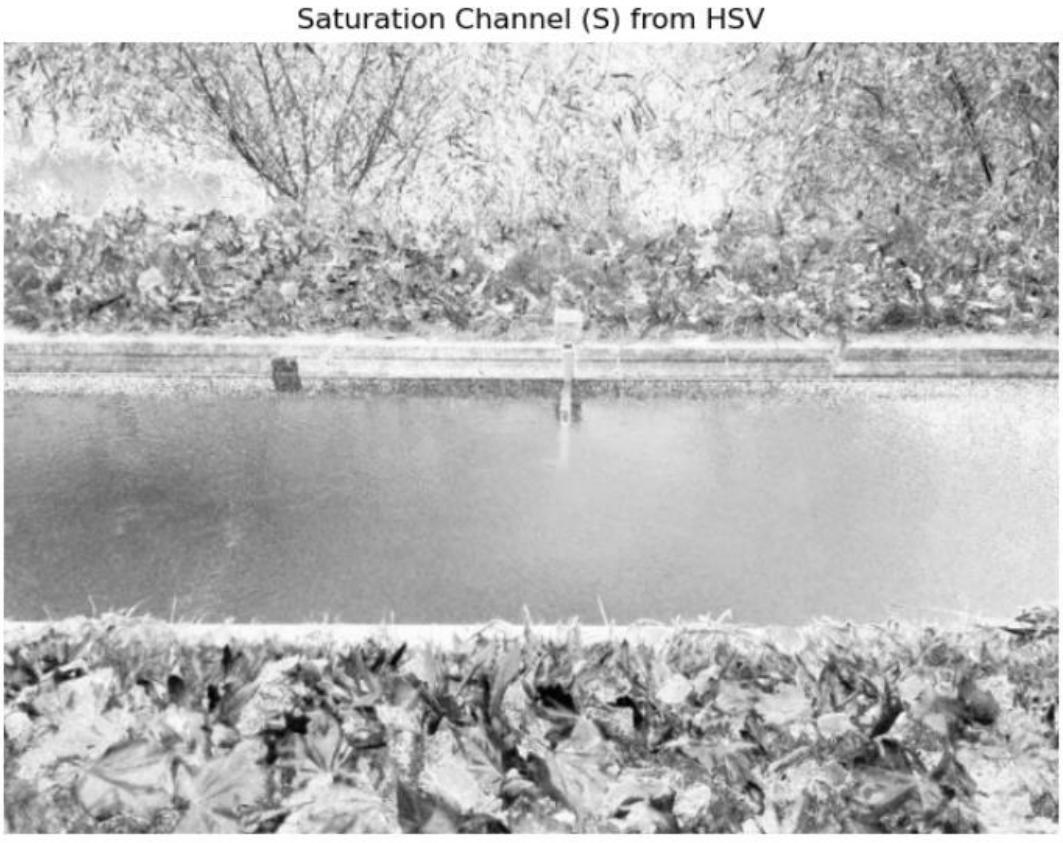


The Hue (H) channel represents color tones as angles in a circular color space. By applying a sine transformation, we can visualize periodic variations in hue, making color transitions clearer.

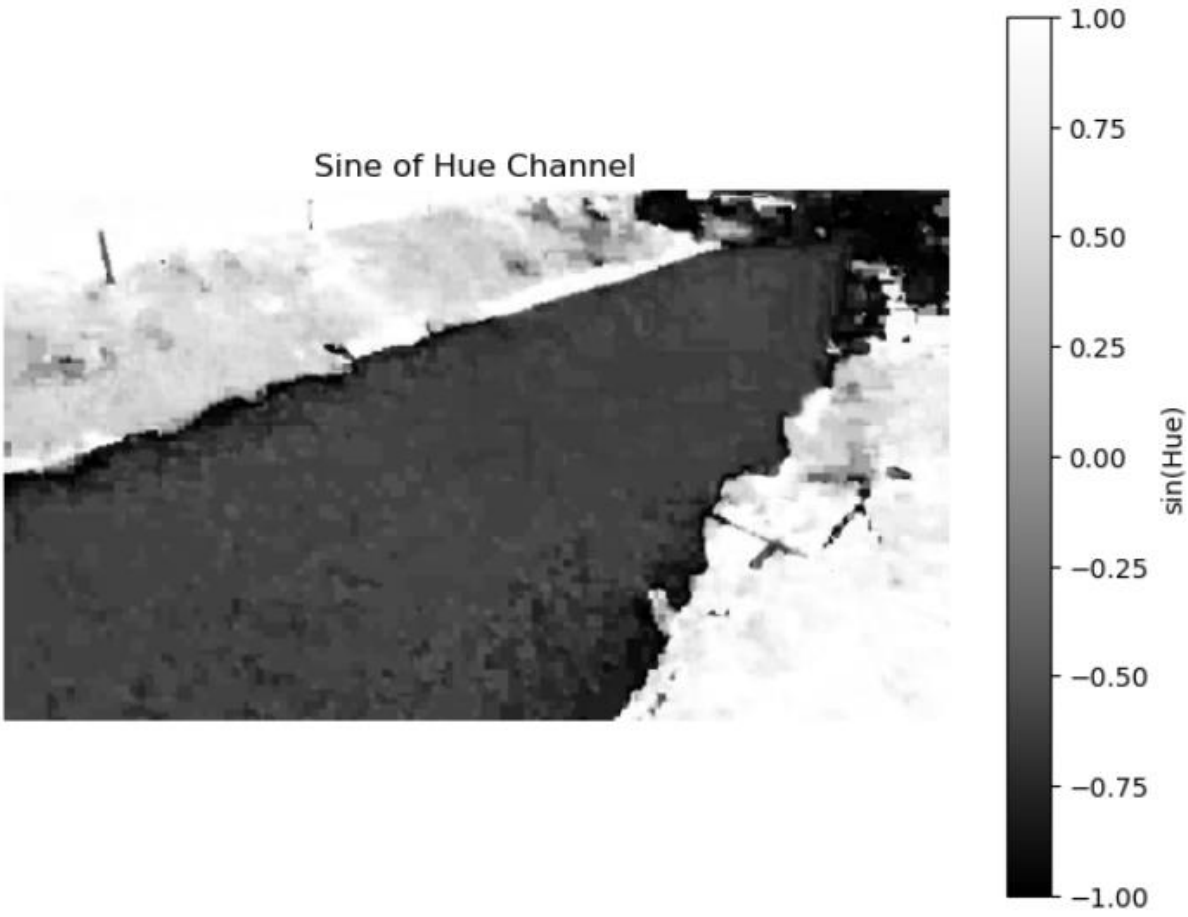
The Saturation (S) channel indicates color intensity. A higher saturation value means a more vivid color, while a lower value results in a duller, grayer color.

Normal camera


Sine of Hue Channel
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
sin(Hue)


Saturation Channel (S) from HSV

Hommerich dataset


Sine of Hue Channel
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
sin(Hue)


Saturation Channel (S) from HSV

7.4.7 Testing differen color scemes on hommerich dataset

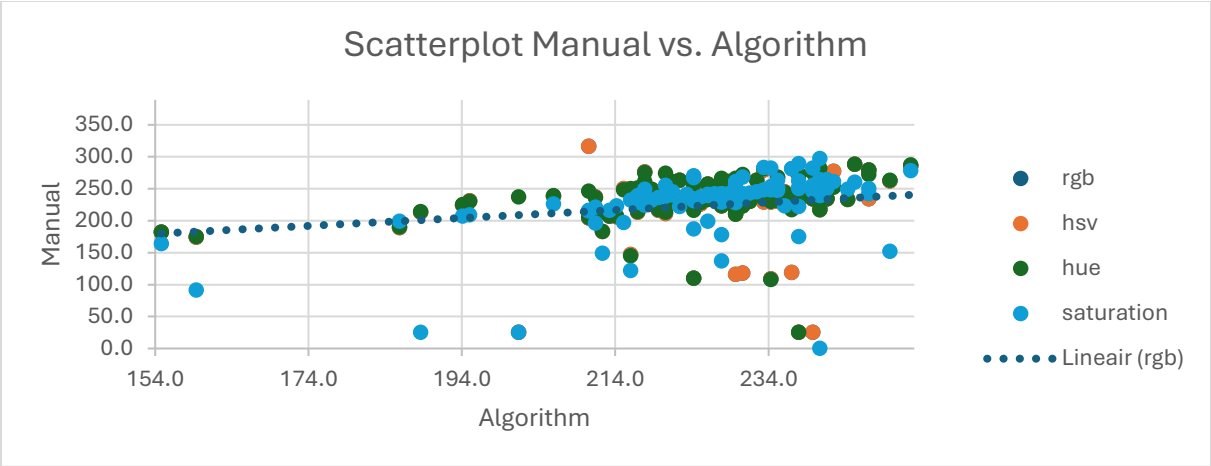
The algorithm and manual water level labelling process is run to analyse the consistency of different colour settings as input for the waterlevel detection algorithm. From first estimation, HUE seems to have the highest contrast between land and water, as green and blue has a higher contrast compared to pixel intensity. However, after running the script on a sample size of 100 randomly selected images from Hommerich dataset (during daytime), there seems to be no clear difference between the accuracy of the different color settings.

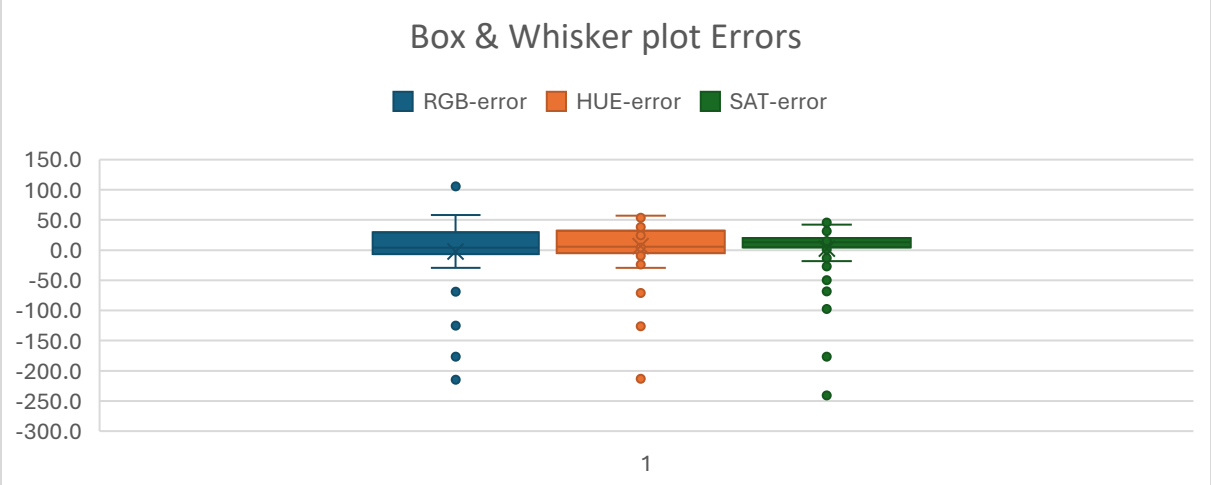
Two tests were conducted. Firstly, the raw images were cropped to a small portion of the entire frame to save computation demand. A manual labelling algorithm was run to determine the 'real' water level visually. After that the algorithm was run on the cropped dataset. These results were compared.

	HSV-error	HUE-error	SAT-error
MAE	28.3174	22.8026	27.1788
RMSE	5.3	4.8	5.2
Bias	-2.3	6.8	2.3

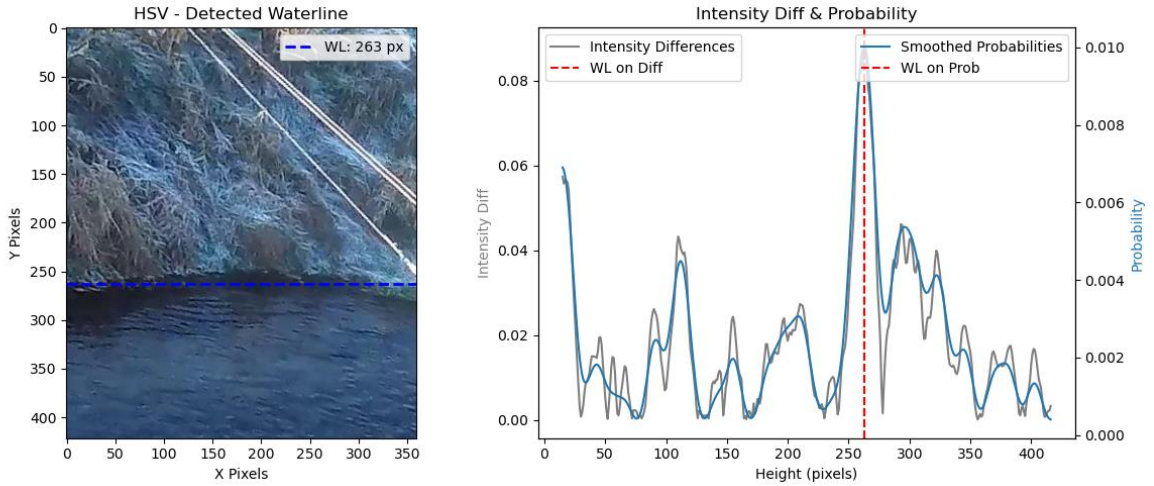
Here the hypothesis and expectation of the HUE being the most effective collour setting for land/water contrast was not met. The mean estimated error and root mean squared error were not significantly different.

In the scatterplot we see a decent correlation between algorithm results and manual estimation of the waterline. However, there are quite a lot of outliers.

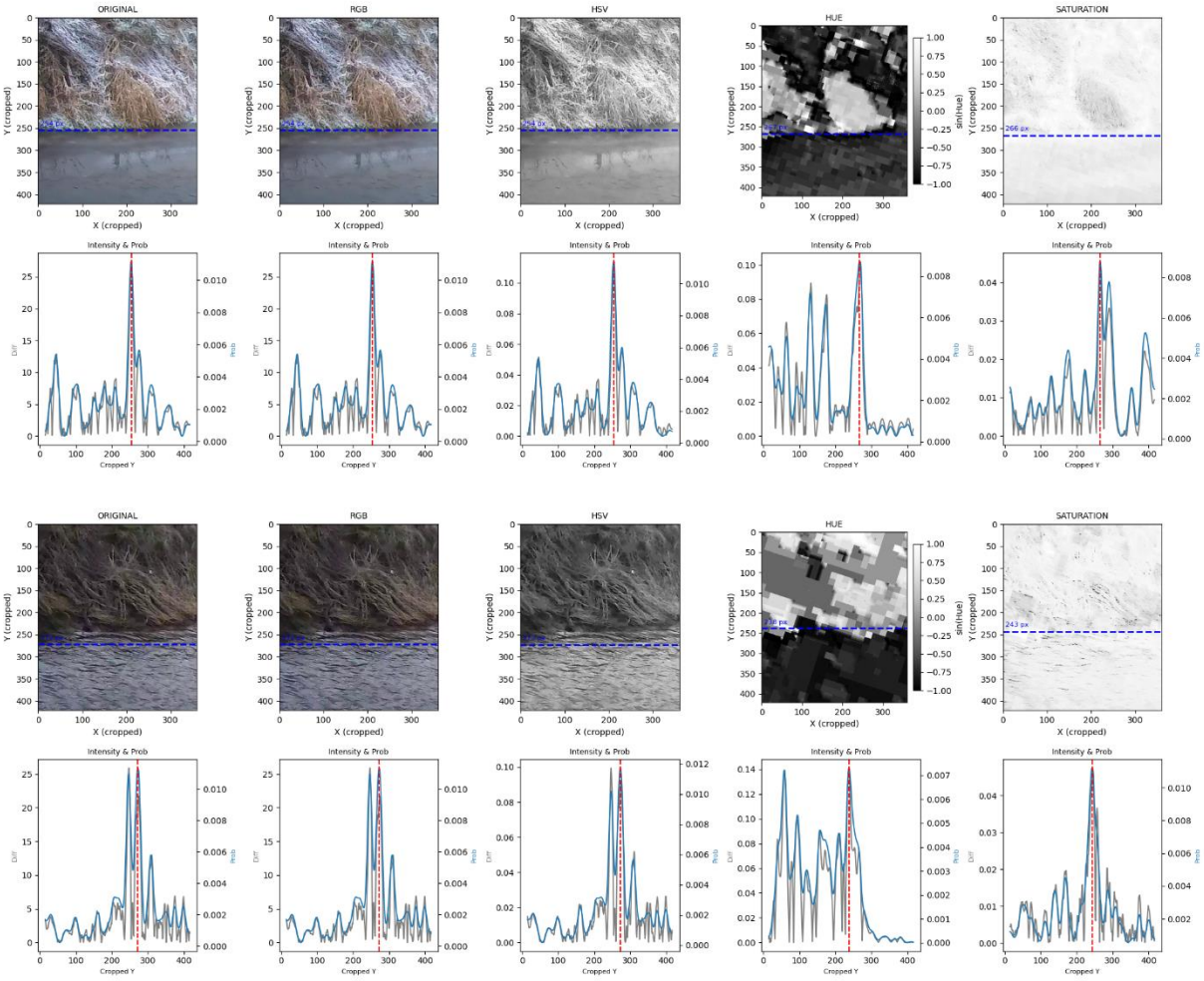




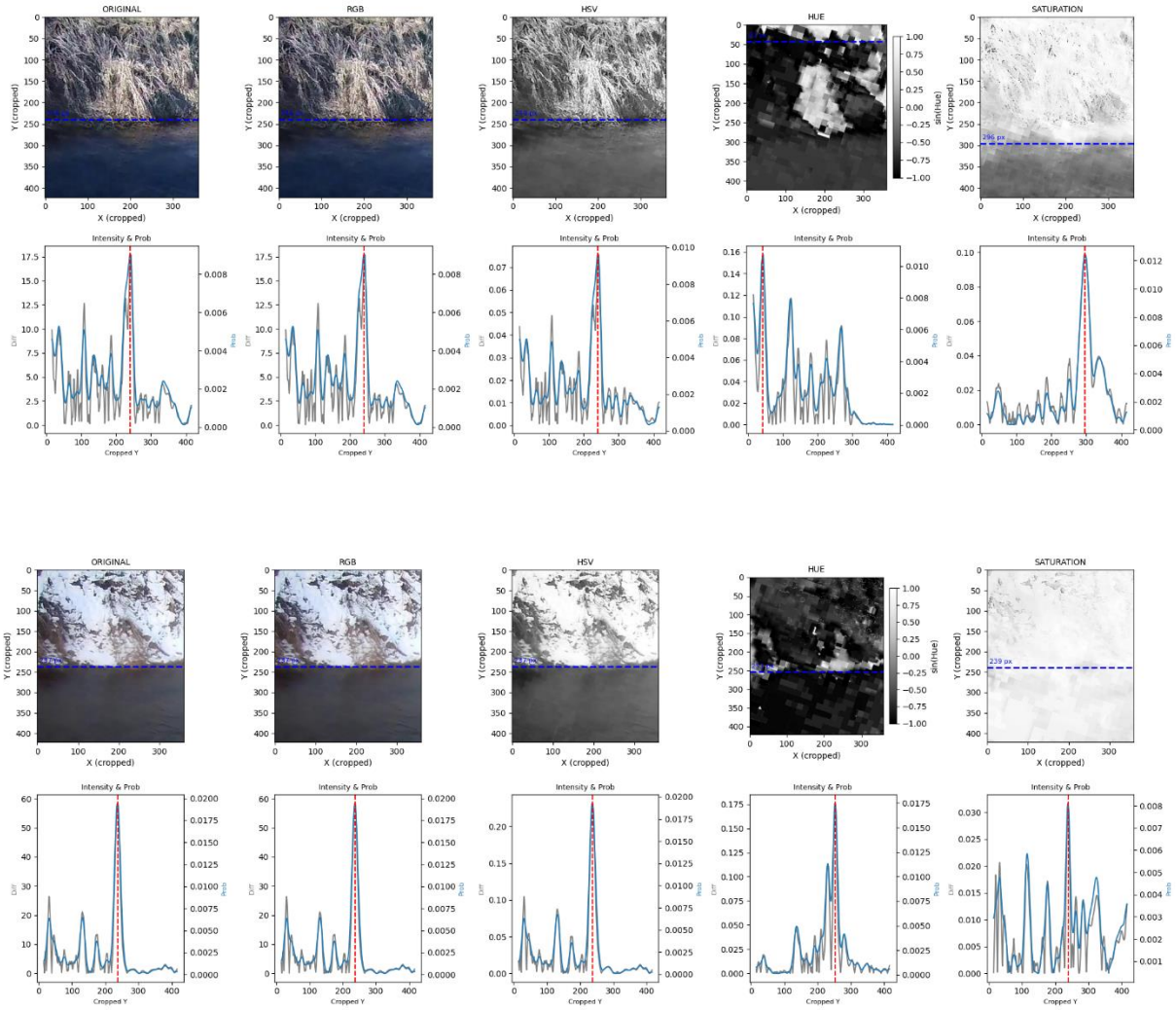
Pixel scale:



Sometimes, the water level detection algorithm performs realy well on all color settings. Other times, only one or none of the methods depict a representation of the real water level



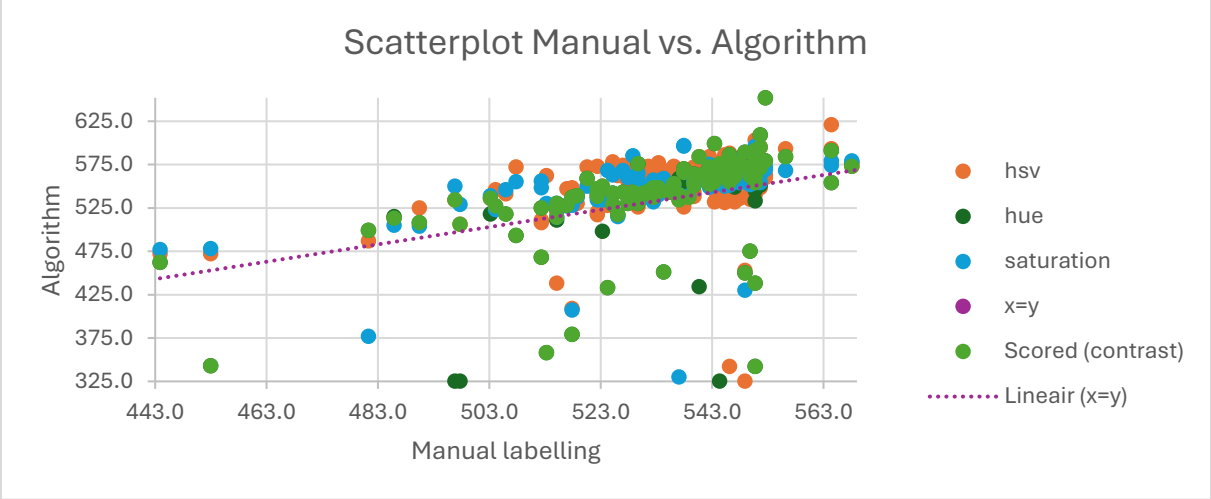
Under certain environmental conditions (frost or low sun angle), the hue image is very distorted or diffuse.

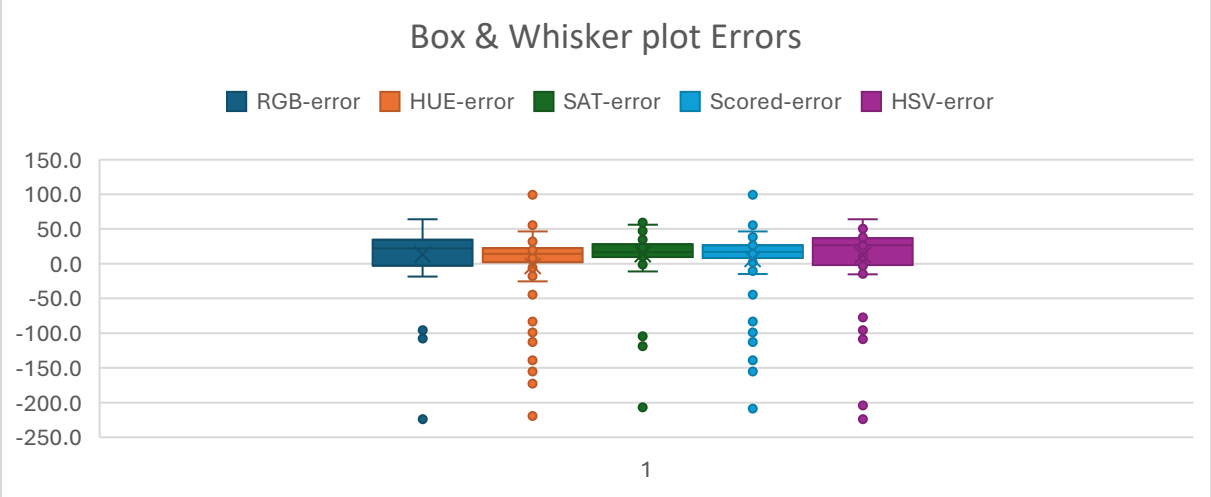


- Image_data_variants_dynamic cropping.ipynb
    - Utilises a dynamic cropping and rotating script to preprocess images and then run the waterlevel detection algorithm
- Image_data_variants_cropped_input.ipynb
    - Takes preprocessed images to reduce computation time and power demand. Runs the same script and should give the same results
- General images get visualized using RGB color coding. Each pigment gets a value between 0-255 which determines the color of the pixel. Besides RGB, a couple other color codings are available like the following. Hue, corresponds to where the color lies on the traditional color wheel. This means it ranges from 0-360* or 0-1. In coding it is recommended to connect the ends so that 1* 360* are similar instead of 359* apart. Saturation refers to the purity of a color. It measures how intens and vivid a color appears. Combine the last two and we get HSV (hue saturation value), which is in itself a alternative color coding.
- Using a labaling algorithm, the ‘real waterlinie’ is again assigned visually on a random set of 100 images from the Hommerich dataset. These were compared with the algorithm output for every color coding and with a scored output. The scored output selects the detected waterline with the highest contrast in pixel intensity of the intensity boxes across all color codings. However,

looking at the statistics, this scored method has not significantly increased the water line detection capabilities as shown in the statistics.

	RGB-error	HSV-error	HUE-error	SAT-error	Scored-error
MAE	26.18	29.61	34.50	24.65	29.83
RMSE	5.12	5.44	5.87	4.96	5.46
Bias	13.19	12.40	-3.05	13.72	6.74
Pearson r	0.39	0.34	0.50	0.50	0.53
R^2	0.15	0.11	0.25	0.25	0.28





7.4.8 Power monitoring experiment

The current setup is tested in standby mode (no active script running) and active detection mode (script running on repeat). The results concluded that with the current setup (powerbank without the use of a nano timer), experiments can be conducted as long as the battery can be changed every day.

Standby mode:		~275 mA/h	
Active mode:	running script constantly:	~522 mA/h	
Realistic use case:	running every ~10min:	~333 mA/h	~60h on 20.000mA battery

The current IR LED runs on a current draw of ~1A. however, because its only lit 2seconds every 10 minutes. The power consumption is not significant.

Several tests were conducted running the script. The inconsistency of the power consumption points to the possibility of more acting variables such as temperature, device efficiency after running updates ect.

7.4.9 Notes on prototype 4 and feedback loop for next prototype

The base for the hardware and algorithm is set. The main issue for now before we can transition to the next phase (fieldwork) is a robust and inclusive design so the prototype can be tested in a real world environment. A power saving mode is under development, but not successful yet. The nano power timer is causing power issues and the system is not yet capable of running the algorithm from bootup without any manual intervention. These things need to be addressed before we are able to transition to the testing phase.

SCAMPER Heuristics

- Substitute: What materials or processes can be replaced?
  - The device casing is not yet waterproof and has to be more robust for real world deployment. Also the option of switching to a 12V power source should be explored so night time measurements can be tested.
- Combine: What elements can be integrated?
  - The NoIR camera module should be tested for day measurements so only one camera module has to be present in the prototype. Is the NoIR camerea module as effective as the standard one?
- Adapt: How can the concept be adjusted for improvement?
  - The need for low power and power saving mode has to be improved so the system is only active when It has to. The nano power timer is now still giving issues that need to be resolved. Also a way of accessing the system outside would be great (portable display for example).
- Modify: What aspects (e.g., size or shape) can be altered?
  - The new casing has to be modified so rain is not damaging the camera or causing droplets or condensation that influence the image capturing.
- Put to another use: Can the concept serve other purposes?
  - The current device is able to be used as a monitoring system for security or the consumer market like a diy ring doorbell or birdhouse camera.
- Eliminate: What features can be simplified or removed?
  -
- Reverse: Can processes be reversed or re-sequenced?

7.5 PROTOTYPE 4B

Hessel has provided me with an python environment and script to transform the image to a 3D referenced system. A cross section is measured in the field and these coordinates are linked to the image. poligons run across this cross section and make pixel distributions across the entire cross section. These distributions can be compared to find the land/water line.

This function is optional and additional to the project. it gives insight on how to process the data on a separate server. The project however focusses more on taking measurements locally and on board processing.

7.5.1 Algorithm

A chi-squared method is applied to the script results to try to find the most probable land/water line. However, after running a couple of images trough the wle script, the chi2 does not seem to perform verry well (using RGB). As of right now, the WD script performs better. The WD differs from the WLE on a couple of aspects:

WD;

- Sweeps row by row over the ROI
- Measures mean intensity difference of the intensity boxes
- Score based on highest intensity difference
- Creates a 1D profile mean intensity difference

WLE:

- Creates polygons over an 3D cross section
- Displays intensity distribution (histograms) for every polygon
- Creates an CDF displaying ordered intensity counts in every polygon
- The cdf can be compared to show differences in texture/brightness/color of surfaces

Some adaptions are made to the code from pyorc:

- Color coding is changed so HUE is used. This seems to work better than RGB
- Added chi2 for wle
- Applied a piece of code that applies a penalty when both waterlines are not corresponding t othe same water height. This optimalisation step makes sure it detects land/water levels that imply an horizontal water surface.