Final “As-Built” Report

December 13, 2024

Team Members

Dylan Anderson, Jennie Butch, Noah Gooby
Nathan Hill, Jade Meskill

Sponsored By

Dr. Eck Doerry

Team Mentor

Vahid Nikoonejad Fard

Capstone Instructor

Isaac Newton Shaffer

Version

1.0

Overview: This document outlines the overall development of the Floodbusters project.

Table of Contents

Introduction

Flooding is the single most common and destructive natural disaster, causing over $3.7 billion in damage and claiming more than 120 lives annually across the United States. Nationwide, the frequency of disastrous flood incidents has more than doubled since 2000 and is expected to more than triple by 2050. Here in Coconino County, a recent analysis of post-wildfire flood risks has shown that local water flows could soon reach between 3x – 16x normal levels. The growing frequency of these natural disasters demonstrates the urgent need for an efficient, accurate, and innovative solution.

Traditional flood detection systems are often cumbersome, expensive, and too impractical to be deployed on a large scale. Many current solutions rely heavily on substantial physical infrastructure, such as complex gauges and even entire meteorological stations in order to collect data on flood risks and local water levels. Newer, “smart-camera”-based solutions, while an improvement, require an extensive calibration process, requiring a team of surveyors to utilize expensive, non-trivial equipment, who then must travel to remote areas to record accurate measurements between gauge points and a chosen zero point in order to calibrate their cameras. This approach in gathering calibration data is expensive, requires a large amount of manpower, and is too error-prone to be considered practical.

The HydroCams project, sponsored by Dr. Doerry, seeks to mitigate these issues. Through the use of cheap, easily installable, solar-powered, cellular-connected smart cameras, our software will automatically detect gauging points and perform necessary metrological calculations using computer vision and pixel-based distance calculation techniques. This allows water levels to be measured automatically based on which gauge points are obscured, providing local entities with accurate, real-time information on water levels and flood risks. This approach aims to be foundational to all future flood detection systems, as it fully addresses the key issues that plague current solutions: cost, accuracy, access to real-time information, and efficiency, all of which will save lives and potentially millions of dollars in damages.

Process Overview

To develop the Image Workbench, we primarily used Git and a task manager in the form of a spreadsheet. New features or important changes were made in Git branches, which were then merged back into the main branch upon review. Individual team members had roles with specific responsibilities.

Team Members and Responsibilities:

- Jennie Butch: Team Leader; Responsible for team communication and deadlines.
- Noah Gooby: Client Communicator; Responsible for communicating with the client alongside the team leader.
- Jade Meskill: Archivist; Responsible for recording meeting minutes.
- Nathan Hill: Architect; Responsible for ensuring code matches the project architecture.
- Dylan Anderson: Release Manager; Responsible for ensuring the project is versioned properly, reviewing commits and pull requests.

Requirements

We acquired requirements for the HydroCams Image Workbench through a detailed review of both technical needs and overall application objectives. This involved focusing on core functionalities and technical constraints, along with ensuring the application is robust, maintainable, and accurate. Below, you will find an overview of our updated key requirements defined in our Requirements Specification Document.

Functional Requirements:

- Users can upload images (PNG, JPEG, etc.) from their local devices for processing.
- Automated marker detection based on configurable color and size parameters.
- Users can view, edit, delete, and organize detected markers.
- Users can designate a marker as the zero-point for distance calculations.
- Users can zoom and pan within the image canvas.
- Marker and distance data can be exported via a JSON file.

Non-Functional Requirements:

- Computer Vision operations must complete within 10 seconds.
- Saved images retain original quality.
- The full webpage must load within 2 seconds on average.
- User interactions and file operations must occur within 1 second.
- The application must include clear error messages.
- Code will be comprehensively documented and adhere to standards.

Environmental Requirements:

- Images will be provided by a HydroCam camera.
- Markers will be made of a durable, painted material.
- The application must be cross-browser compatible, supporting Chrome, Firefox, and Safari.

In summary, our requirements reflect a blend of essential functionalities, performance expectations, and environmental considerations to help create an efficient and user-friendly application. These requirements served as our foundation for developing a reliable, intuitive tool that meets the technical requirements while maintaining high performance and usability standards.

Architecture and Implementation

We designed the architecture of the HydroCams Image Workbench with the goal of balancing modularity, scalability, and maintainability. We decided to use a layered architecture, allowing us to clearly separate responsibilities across the system and to promote future maintenance and flexibility. The system consists of two primary layers: the front end for user interaction and distance calculations, and the back end for processing and data management. Within these layers, each module has a clearly defined set of responsibilities, making the application highly modular and easy to scale.

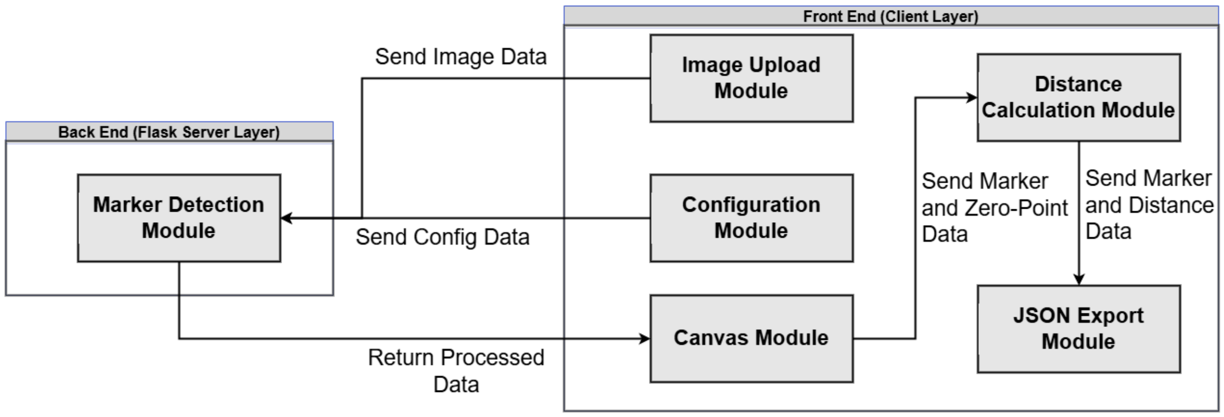

Figure 1 | Architectural Overview Diagram

Key Responsibilities of Each Component:

1. **Marker Detection (Back End):** Handles all computer vision tasks based on configuration data received from the front end. It performs these CV tasks and sends the results back to the front end, specifically the canvas module.
2. **Image Upload (Front End):** Responsible for uploading images from a user's local device. This module sends the image data to the back end for processing.
3. **Configuration (Front End):** Provides intuitive tools for users to configure parameters such as marker colors, contour area ranges, and known marker size. This data is sent with the image data to the back end for processing.
4. **Canvas (Front End):** Handles image rendering, marker placement, and user interactions such as zooming, panning, and editing markers via clicking on the canvas. Also allows users to select a marker as a zero-point with known dimensions.
5. **Distance Calculation (Front End):** Computes vertical distances between markers and a zero-point based on the known positions and size of each marker. It determines a scaling factor based on the pixel dimensions and known dimensions of each marker to accomplish this.

6. **JSON Export (Front End):** Facilitates the export of marker and distance data as JSON files, which can then be downloaded by the user to their local device.

The communication between each of our modules and layers is highly streamlined and straightforward. First, the image and configuration data are sent to the back end via a POST request. The back end receives this data, performs the computer vision operations, then returns the results in JSON format. Upon receiving the response, the front end calls on the Canvas module to render the image with the detected markers.

After the user has made their reviews and selected their zero-point, our Distance Calculation module makes all relevant calculations and updates the canvas with this data. Finally, the user can select to download the JSON results, which triggers the JSON Export module to format all data appropriately before allowing the user to download the file to their local device.

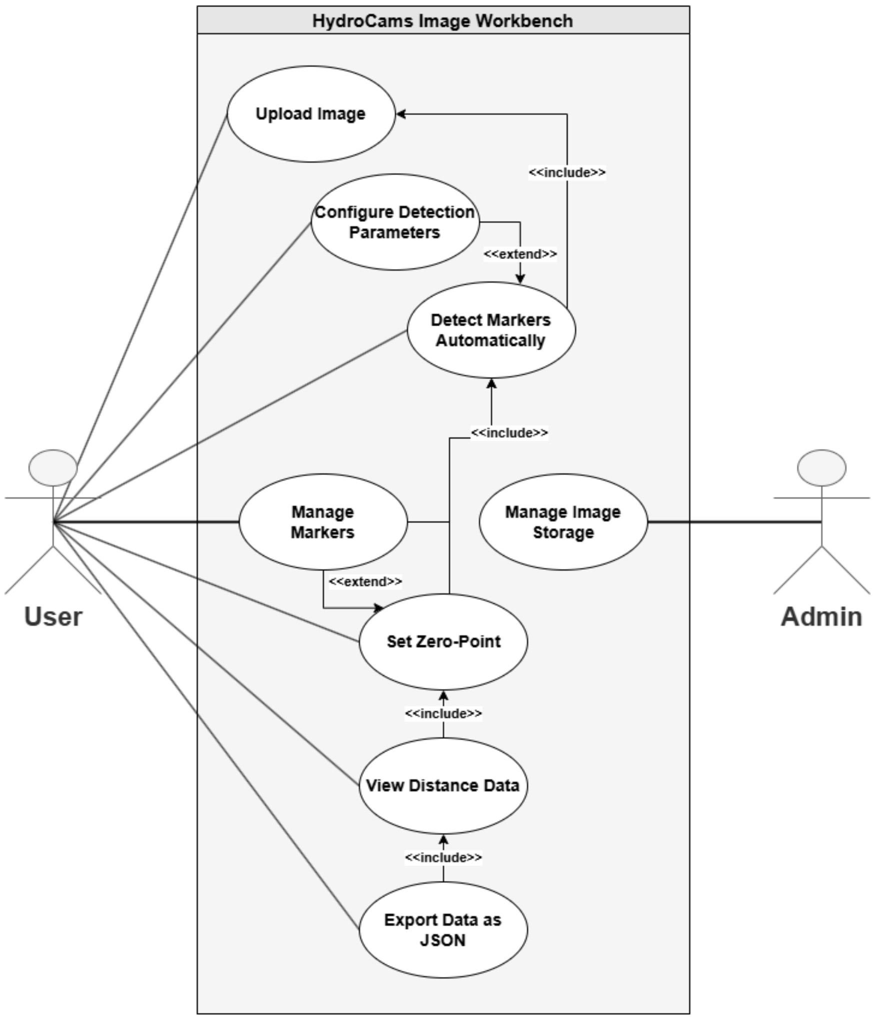

Figure 2 | Standard Use Case Diagram

Throughout our development of the HydroCams Image Workbench, there have been several key differences between our prescriptive (“as-planned”) and descriptive (“as-built”) architectures. These changes stemmed from changes in priorities, technical challenges, and evolving requirements that rose to our attention during implementation.

Our most significant deviation was the decision to move away from Structure-from-Motion (SfM). While we initially planned on using SfM to calculate 3D marker detections, it became clear that it was far too resource intensive, complex, and overall burdensome for the scope of our application and the given timeline. The processing overhead and complexity of implementing SfM would have drastically increased our development time and would require far greater testing, which conflicted with our other critical deliverables. We resolved this issue by switching to pixel-based distance calculations, which retained acceptable accuracy while significantly reducing resource overhead and development time.

Another major adjustment was our shift away from a monolithic architecture, towards a more modular layered architecture. We initially utilized a monolithic pattern during early prototype development, but it quickly became evident that separating the front and back end into distinct layers would be a dramatic improvement. Additionally, we fully refactored the codebase to provide significantly better organization and modularity, as well as bolstering our new architectural pattern.

In addition, some of our planned functionalities were implemented differently than we had originally anticipated. For instance, we initially expected the marker detection process to rely on predefined marker configurations (size, color, etc.), but ultimately realized that was not sufficient. Instead, we added configurable detection parameters, allowing users to define specific color ranges and detection areas. Furthermore, we implemented a prototype feature to specifically detect staff gauges provided by our client, allowing for even more customization regarding automated computer vision detection.

Together, these changes illustrate how our development process required continuous reassessment of priorities in order to deliver a system that meets user and client needs, while remaining within practical constraints. Our focus on modularity, maintainability, and usability allowed us to refine our architecture to a state that we believe is far superior to our original vision.

Testing

Delivering a functional, reliable, and accurate product was top priority for our team. Throughout development, each feature was regularly tested, through both hands-on and automated testing. Product testing was divided into three main categories: unit, integration, and usability testing. While multiple tests were implemented in each category, specialized tests were also run to ensure key requirements were met for the project. The final phase of testing focused on accuracy and consistency of images and simulated real world scenarios.

While unit testing and integration testing were considered two separate categories, we found that a majority of the testing activities were performed either in tandem with one another, or would roll from one test item to another. For example, once the marker detection module was thoroughly vetted, its data was then fed into the marker calculation module, and so on. A similar approach was taken with the usability testing. While the main focus of the usability testing was to test the user experience and ensure ease of use, it also allowed for additional verification and validation of module integration.

The final phase of testing consisted of ensuring that the application could be hosted both locally and on a server, testing multiple browsers, and testing multiple images with varying markers. The application can be reliably run on a local machine or hosted on a server, using Linux, window, or macOS. It also supports all modern browsers including Firefox, Chrome, and Edge. Over 75 images were processed with marker sizes ranging from 2 to 6 inches at distances up to 30 meters. Various lighting and weather conditions were also represented in the images ranging from rain and snow to various hours of the day and lighting conditions. Markers consisted of both circles and squares along with a staff gauge. Blue and red colored markers tested and found to both be consistently detectable.

The results of the image testing showed that with ideal lighting conditions, 2 inch markers could be detected consistently at 10 meters and 3 inch markers could be consistently detected at 20 meters. At 30 meters, markers could not be consistently detected at all. Markers within 2 feet of the zero point marker could be measured with an accuracy of approximately 95%. Beyond 2 feet, accuracy dropped as low as 65% at 20 meters.

Project Timeline

Our project timeline, outlined in the Gantt chart below, showcases the phased approach we took to develop the HydroCams Image Workbench.

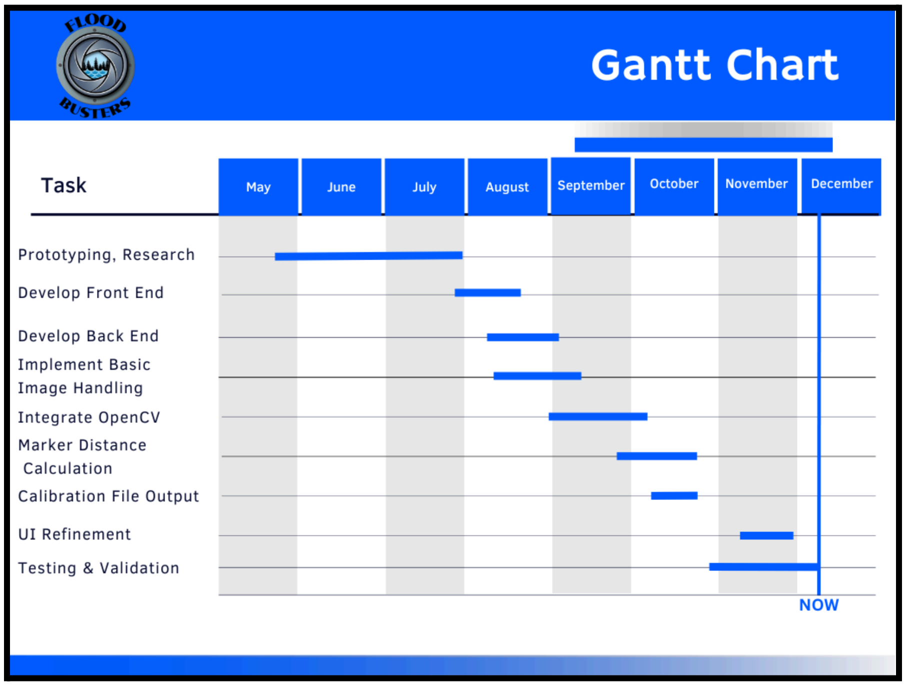

Figure 3 | Gantt Chart

Gantt Chart Sections:

- **Prototyping and Research:** During this time, our team focused on understanding the technical requirements of the project, and began researching and testing technologies like OpenCV, Flask, and SfM.
- **Front End Development:** Here, we built the user interface with a strong focus on intuitive and responsive design.
- **Back End Development:** At this stage, we developed a Flask-based server to eventually handle image processing and marker detection. Our key focus was ensuring communication between the front and back ends.
- **Image Handling:** After laying a foundation for the front and back ends, we implemented image upload and canvas functionalities, allowing users to load and dynamically interact with images.
- **OpenCV Integration:** Next, we implemented marker detection via OpenCV. This section was pivotal as it serves as the core functionality of our application.

- **Marker Distance Calculation:** We began implementing marker distance calculations towards the end of our OpenCV integration and continuously refined them along with our marker detection algorithm throughout the project.
- **Calibration File Output:** After implementing robust detection and distance calculation modules, we shifted towards implementing the export of calibration data in JSON. This was a very simple process, benefiting from the flexibility of our detection and distance calculation systems.
- **UI Refinement:** Even though the workbench was usable and functional, the UI still needed some improvement after receiving feedback from our client. We implemented some small visual changes, and some slightly larger functional changes that allowed for a more refined marker detection process.
- **Testing and Validation:** As we approached the end of the semester, testing was required to ensure our product was polished enough to deliver. We used testing frameworks like Jest and pytest to ensure each component of the workbench functioned as intended.

As of the end of the semester, we have produced a functional, polished product for our client, consisting of an image workbench with a browser front end, and a Flask server back end. Despite completing every point listed above, there are still improvements (outlined below) to be made to the workbench, which our client may choose to pursue.

Future Work

As for the future of the HydroCams project, the larger FloodAware project has a vested interest in improving and maintaining it. The team has identified three main concepts for future work: porting the existing Python code to JavaScript, improving marker detection algorithms, and expanding user testing and UI refinement. These concepts are essential for enhancing the functionality and user experience when it comes to the HydroCams workbench.

The first concept is porting the current Python code to JavaScript. By moving the Python code to JavaScript, the workbench has the opportunity to become a fully browser-based application. This change can bring improved speeds and easier maintainability. A browser-based solution will also allow for easier scaling and deployment which can help streamline the overall development process.

The second concept is the refinement of the marker detection algorithms. While the current system is functional, there is still room for improvement, especially in terms of accuracy and efficiency. By improving the algorithms, the system will not only improve its performance, but also have the chance to integrate different detection methods in the future, allowing for adaptation to a variety of real-world conditions.

Lastly, user testing and user interface refinement. Receiving feedback from field users provides valuable insight into the advantages and disadvantages of the workbench. Once more feedback is collected, it can then be used towards making the system more usable and intuitive.

It is important to reiterate that these improvements can improve the overall technical capabilities and user experience of the workbench. These improvements could potentially help the system not only better monitor and detect flooding events, but also other natural disasters such as tornadoes and hurricanes, ultimately broadening the system's impact.

Conclusion

The FloodAware project seeks to reinvent the flood detection process, using solar-powered, cell-connected cameras. The calibration process for cameras like these is incredibly complex and error-prone, consisting of many steps involving trained personnel. The HydroCams system greatly simplifies this process, allowing for an individual to install and configure a camera with ease. The HydroCams system provides:

- Configurable computer vision-based marker detection
- An intuitive user interface, designed to optimize the workflow
- Calibration data, exportable in JSON

The HydroCams system should allow the FloodAware project to quickly and easily configure any newly-installed cameras. The increased flood detection capabilities could save a number of lives and large amounts of money. The Capstone experience has shown us how to work effectively as a team, how to communicate with a client, and how to construct a project professionally.

Appendix A: Development Environment and Toolchain

Hardware

Our application was developed across all major operating systems (Linux. macOS, and Windows). The macOS system had 16 GB RAM, and the Apple M1 processor. One Linux system had lower specs, with 8 GB RAM and an Intel i5 series processor, while the lowest spec system we had was an AWS t2.small, with only 2 GB RAM and a single virtual 3.3 GHz processor. The hardware requirements are likely extremely minimal, as OpenCV can run on hardware as minimal as a Raspberry Pi.

Toolchain

The most widely-used development environment was Visual Studio Code, in addition to the myriad plugins used, but most importantly including Python and JavaScript linters, flagging errors or stylistic issues. The use of Python requires the use of its built-in package manager, pip, which is essential for installing new Python packages.

Setup

1. To ensure that Git is installed, download the latest version from https://git-scm.com/downloads, or follow the instructions there for your package manager of choice.
    a. To check that it installed correctly, open a terminal and run "**git --version**". The output should be something like "git version 2.4X.Y".
2. To ensure that Python is installed, download the latest version from https://www.python.org/downloads, or install it using your package manager of choice.
    a. To check that it installed correctly, open a terminal and run "**python3 --version**". The output should be similar to "Python 3.1X.Y".
3. The source code of the workbench can be obtained by cloning the Git repository or downloading the code as a .zip file. To clone the repository, run "**git clone https://github.com/HydroCams-Team/Image-Workbench**", or to download the source code, visit the above URL and click "Code -> Download Zip".
4. Once the source code has been obtained, open a terminal window in the "Image-Workbench" directory, and simply run "**python3 app.py**". This should allow you to access the workbench at "**http://{ip}:5000**".

Production Cycle

As both Python and JavaScript are interpreted languages, there is little complexity involved in making and deploying changes. If a change is needed, simply alter the source file associated with your change, restart the Flask server by killing the existing process, and run “**python3 app.py**” again.