Software Design Document

August 30, 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 main implementations for the HydroCams development process.

Table of Contents

1 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. Furthermore, these approaches typically require a team of surveyors utilizing expensive, non-trivial equipment who must travel to remote areas to record accurate measurements between gauge points and a chosen zero point. This approach is expensive, requires a large amount of manpower, and is unable to provide real-time data or warnings of imminent flood risks.

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. 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.

At this stage, we are deep into the implementation phase of the project, focusing entirely on core functionality. This document will cover the modules we have developed, including image upload, marker detection, and distance calculation, detailing their responsibilities, interfaces, and how they interact within the system. We will also provide an overview of the current system's architecture and workflow, demonstrating how the individual modules contribute to solving the flood detection problem efficiently and effectively.

2 Implementation Overview

The HydroCams project aims to provide an affordable, real-time flood detection system using solar-powered, cellular-connected smart cameras. For our purposes, these cameras will collect images that will be uploaded to the workbench for processing. The workbench utilizes several technologies: OpenCV for image processing, JavaScript for overall functionality, and HTML/CSS for user interface. Each component is essential in making the workbench functional as well as appealing.

The design pattern for our tool is Model-View-Controller (MVC). In our scenario, OpenCV is the “Model”, handling the core data processing and image analysis for the markers. HTML/CSS is the “View”, providing the ability for the user interface to display the image and interact with the user. JavaScript functions are the “Controller”, managing the user interactions by sending data to the “Model” for processing, and updating the “View” based on these interactions.

2.1 | OpenCV

OpenCV is a key part of our solution, as it provides the computer vision algorithms we need to analyze the images received from the cameras.

2.2 | JavaScript

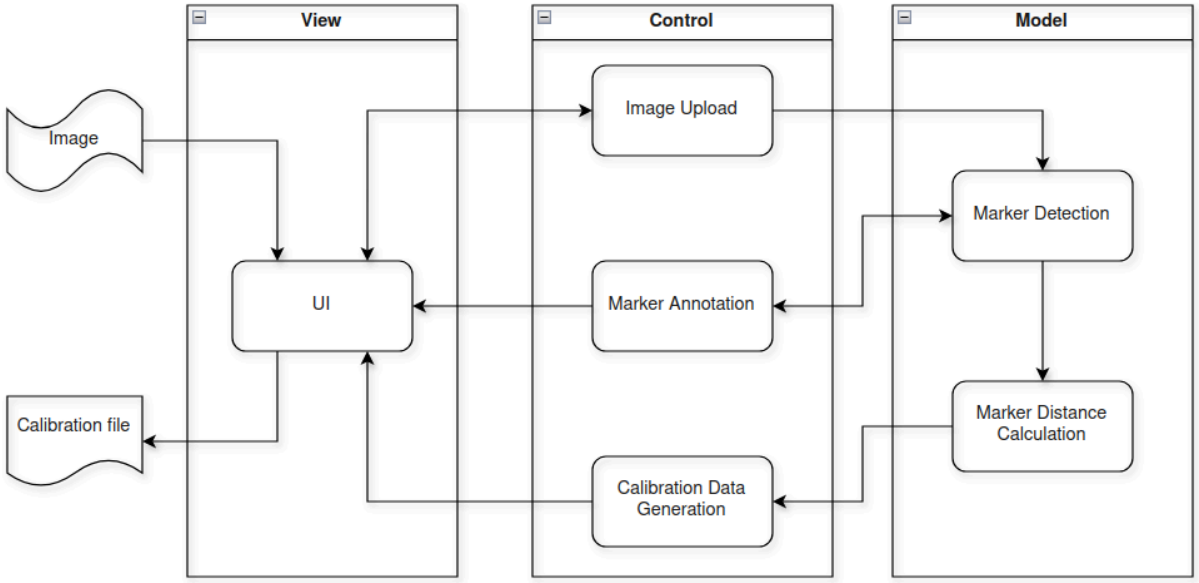
The functionality of the workbench is enabled by JavaScript, allowing users to interact with the system. JavaScript provides the tools essential for manually annotating the images, just in case the computer vision does not detect all the markers present in the image.

2.3 | HTML/CSS

HTML and CSS are responsible for the design and structure of the workbench. While JavaScript provides the workbench functionality, HTML and CSS create a visually appealing user interface. We aim to ensure that the workbench is straightforward to use and navigate.

3 Architectural Overview

3.1 | Architectural Diagram



3.2 | Key Features and Responsibilities

3.2.1 | User Interface Module

The user interface (UI) facilitates the interaction between the user and supporting modules. It also provides an easy-to-use, intuitive interface which enables the user to interact with the system.

3.2.2 | Image Upload Module

The image upload module enables an image to be brought into the web application for processing. Upon successful uploading, the image uploader will also manage displaying the image on the UI canvas and update any annotations made by either the user or supporting modules. The image uploader also manages uploading multiple images and coordinates which images to display.

3.2.3 | Marker Detection Module

The marker detection module automatically scans and detects markers located in the image based on criteria set by the user. Marker criteria include the color and shape parameters of the markers.

3.2.4 | Marker Annotation Module

The marker annotation module uses data from the marker detector module to render and overlay information on the image about each marker detected. This allows the user to quickly view how each marker is being detected and assess if an accurate marker detection has been accomplished.

3.2.5 | Marker Distance Calculation Module

The marker distance calculation module takes data in from the marker detection module and calculates the real world distance between each marker. Once this is completed, it is passed the data onto the calibration data generator module.

3.2.6 | Calibration Data Generation Module

Once the user is satisfied with the marker detection, the user can choose to either download or save the output to the server. The calibration data generation module formats the data from the marker distance calculation module into a structured data file and exports it to the desired location.

3.3 | Information and Control Flow

As shown in the diagram above, information (images) originates with the user and is introduced into the system via the UI with support of the image uploader module. From there, data is passed between the marker detection, image uploader, marker annotation and the UI as the user adjusts available parameters. From the marker detection, data is then passed to the marker distance calculation and out through the calibration data generation module.

3.4 | Architectural Style

The HydroCams project team chose to develop a web application based around a view-control-model architecture. This architecture enabled the software modules to be grouped into three distinct roles; view, control, and model. This allowed the development team to divide modules and utilize specialized frameworks for optimal performance.

4 Module and Interface Descriptions

This section outlines the core modules that make up our system, along with the interactions between them. Each module plays a distinct role, from managing user inputs, to uploading and processing images and displaying results. The purpose of this section is to give a deeper understanding of the responsibilities, interactions, and pivotal functions of each module, which will provide a clear overview of the system as a whole.

4.1 | User Interface Module

4.1.1 | Responsibilities

This module ensures users can view uploaded images on the canvas, adjust preferences for processing (e.g., color, tolerance, and size), and interact with detected markers. It handles dynamic updates on the canvas, allowing users to click on markers and see relevant details instantly. It also offers simple navigation for working with multiple images and controls for features like drawing (for manual marker creation) or deleting annotations.

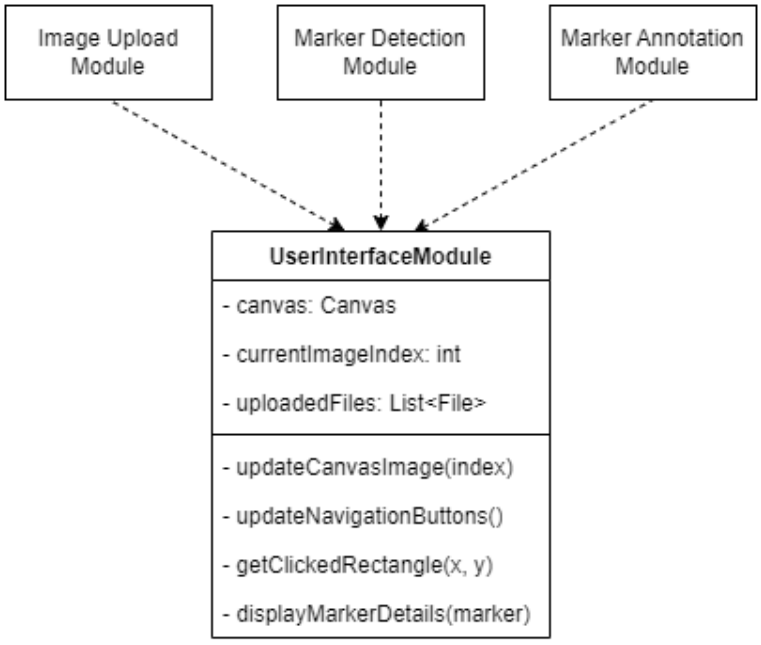
4.1.2 | Interaction and Communication

The UI module interacts directly with the Image Upload module to display the images users upload. It also works with the Image Annotation module to show detected markers and allow interaction with them on the canvas, along with sending the user preference form data. Lastly, it communicates with the Image Annotation module for drawing, editing, and removing annotations on the canvas.

4.1.3 | Core Functions

- **updateCanvasImage(index)**: Ensures that the selected image is correctly displayed on the canvas and adjusts the canvas dimensions to match the image size.
- **updateNavigationButtons()**: Determines whether navigation buttons are needed based on the number of uploaded images, and updates their visibility accordingly.
- **getClickedRectangle():** Determines if a user has clicked on a detected marker. If they have, this function identifies which marker was selected and triggers additional UI updates.
- **displayMarkerDetails():** Displays detailed information about the selected marker when a user clicks on it, such as its position and size, within the UI panel.

4.1.4 | UML Diagram



4.2 | Image Upload Module

4.2.1 | Responsibilities

The main responsibility of this module is to facilitate image uploads and manage how these images are presented on the canvas. For single file uploads, the image is displayed immediately for the user to interact with or submit for processing. When multiple images are uploaded, this module enables users to navigate between the different images using arrow button elements, allowing them to review each image in sequence. It also tracks the currently active image and ensures that all uploaded images are properly formatted and ready for processing.

4.2.2 | Interaction and Communication

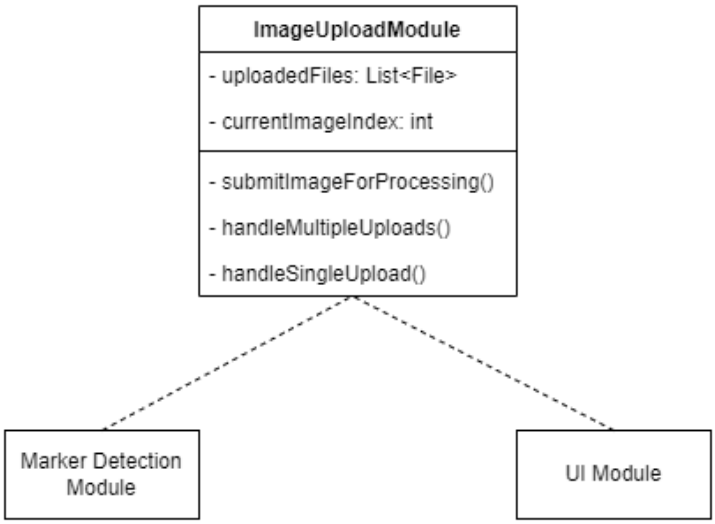
Once an image is successfully uploaded, the module communicates with the Marker Detection and Marker Annotation modules to prepare the image for processing. It also updates the User Interface module, allowing users to view and handle their images seamlessly.

4.2.3 | Core Functions

- **submitImageForProcessing()**: Handles the upload of a single image, checks its validity, and displays it on the canvas for user review.

- **handleMultipleUploads(files)**: Manages the upload of multiple images and allows users to navigate through up to 6 images using navigation arrows. Communicates with the UI module to update the interface.
- **handleSingleUpload():** Manages the upload and validation of a single file. Communicates with the UI module to update the interface.

4.2.4 | UML Diagram



4.3 | Marker Detection Module

4.3.1 | Responsibilities

This module processes the uploaded image by scanning it for markers based on the parameters selected by the user, including color, color range tolerance, and minimum size requirements. It applies color filtering to isolate parts of the image that match these specified criteria. Once these regions are identified, the module calculates the size and position of each marker (in pixels), filtering out areas that don't meet the user-identified requirements. For example, markers must meet certain size ratios, and only contours within a defined size threshold are considered valid.

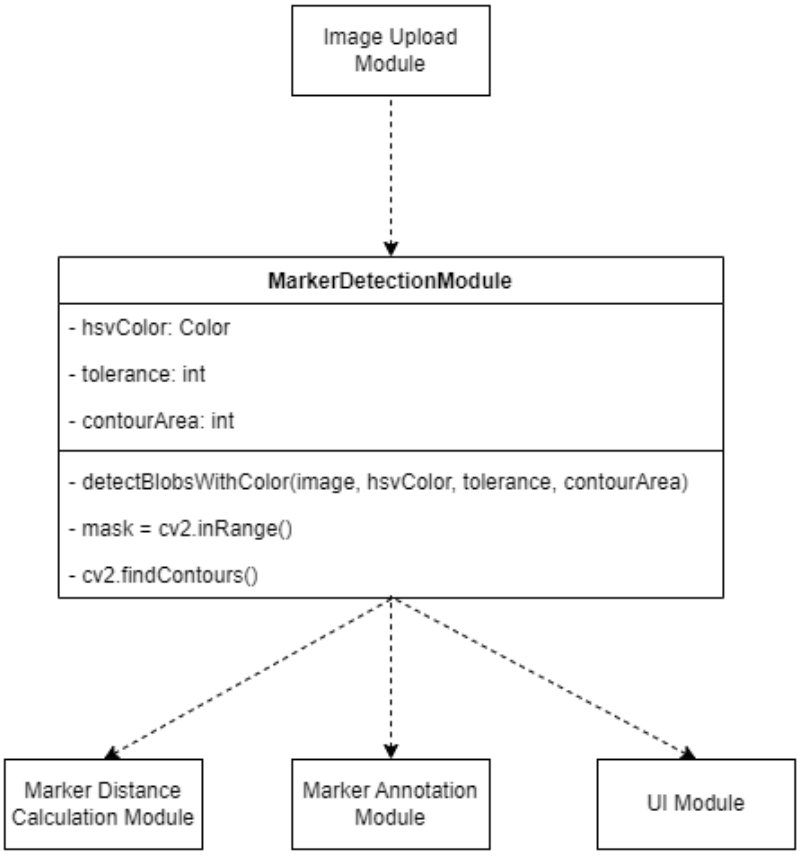
4.3.2 | Interaction and Communication

After detecting markers, this module communicates with the Image Annotation module to draw around and highlight the detected markers on the canvas. It also provides feedback to the user by returning marker details (including coordinates and dimensions) back to the User Interface module. Lastly, it interacts with the Marker Distance Calculation module by sending the processed data and identified markers.

4.3.3 | Core Functions

- **detect_blobs_with_color(image, hsv_color, tolerance, contour_area)**: Processes the image, and applies color filtering and color tolerance levels to isolate areas that match the user's wishes. It detects and returns the markers (blobs) that meet the defined size and shape criteria.
    - **mask = cv2.inRange()**: Creates a mask based on the selected color and tolerance, which is used to identify areas in the image that contain markers (blobs) of the desired color.
    - **cv2.findContours()**: Finds and isolates the contours of detected markers, allowing the system to outline each marker and calculate its position and size.

4.3.4 | UML Diagram



4.4 | Marker Annotation Module

4.4.1 | Responsibilities

This module's main responsibility is to take the detected markers and overlay them on the original image as interactive elements. As mentioned above, this module redraws the outlines around each marker and adds labels to help users easily identify each marker. Since this redrawing happens dynamically, the marker annotations update in real-time as the users interact with the image, giving immediate feedback and allowing for quick edits.

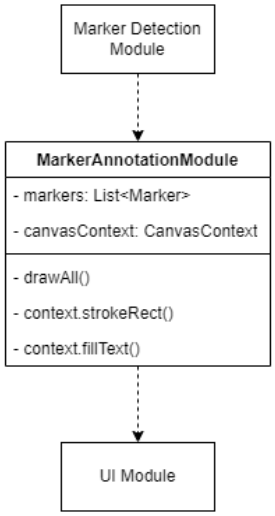
4.4.2 | Interaction and Communication

The Marker Annotation module receives information about the markers (such as their position, size, and contour) from the Marker Detection module. It also interacts with the User Interface module, allowing the user to click on markers to view details and edit the marker, if necessary.

4.4.3 | Core Functions

- **drawAll()**: Draws the detected markers on the canvas using the data acquired from the Marker Detection module. Ensures that each marker is outlined and labeled clearly.
- **context.strokeRect()**: Draws thick rectangular outlines around each marker to make them stand out visually.
- **context.fillText()**: Adds labels next to each marker (in the format of M_1, M_2, ..., M_n), making it easy for users to identify them.

4.4.4 | UML Diagram



4.5 | Marker Distance Calculation Module

4.5.1 | Responsibilities

This module is responsible for calculating the real-world distances between markers detected by the Marker Detection module, based on the 2D image and some known camera and marker attributes.

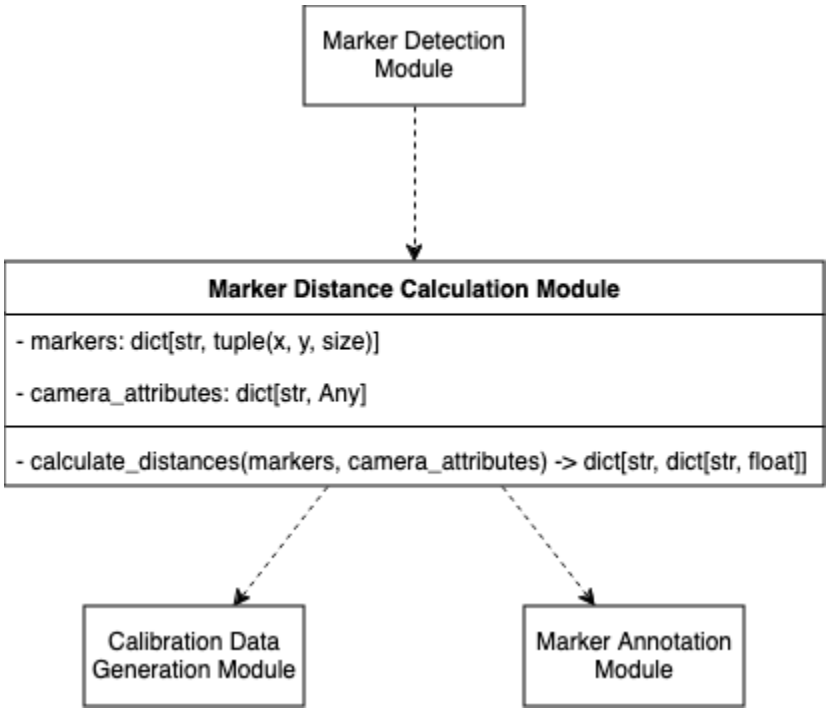
4.5.2 | Interaction and Communication

In order for this module to function, it first needs to receive data from the Marker Detection module. After this module has processed the incoming data, it will forward the real-world distances to the Calibration Data Generation module, for final formatting and output.

4.5.3 | Core Functions

calculate_distances(markers, camera_attributes): For each detected marker, calculate the real-world distances between itself and every other marker, using a Euclidean distance calculation, and some known information about the camera.

4.5.4 | UML Diagram



4.6 | Calibration Data Generation Module

4.6.1 | Responsibilities

This module is responsible for outputting the calibration data for the camera to an extensible file format, which will later be used to accurately detect water levels. It may output the created file directly to the user, as a browser download, or may store it on the host, in a folder with a unique camera identifier.

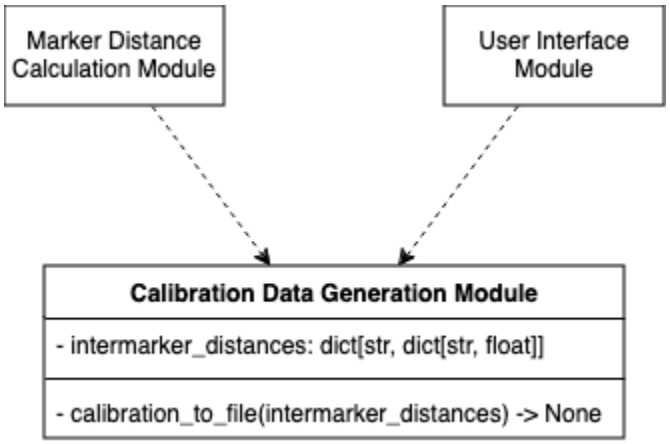
4.6.2 | Interaction and Communication

This module receives data from the Marker Distance Calculation module, which will then be processed and output to some extensible file format for use in the water level detection at the core of HydroCams. This file is requested and saved according to information received from the User Interface module.

4.6.3 | Core Functions

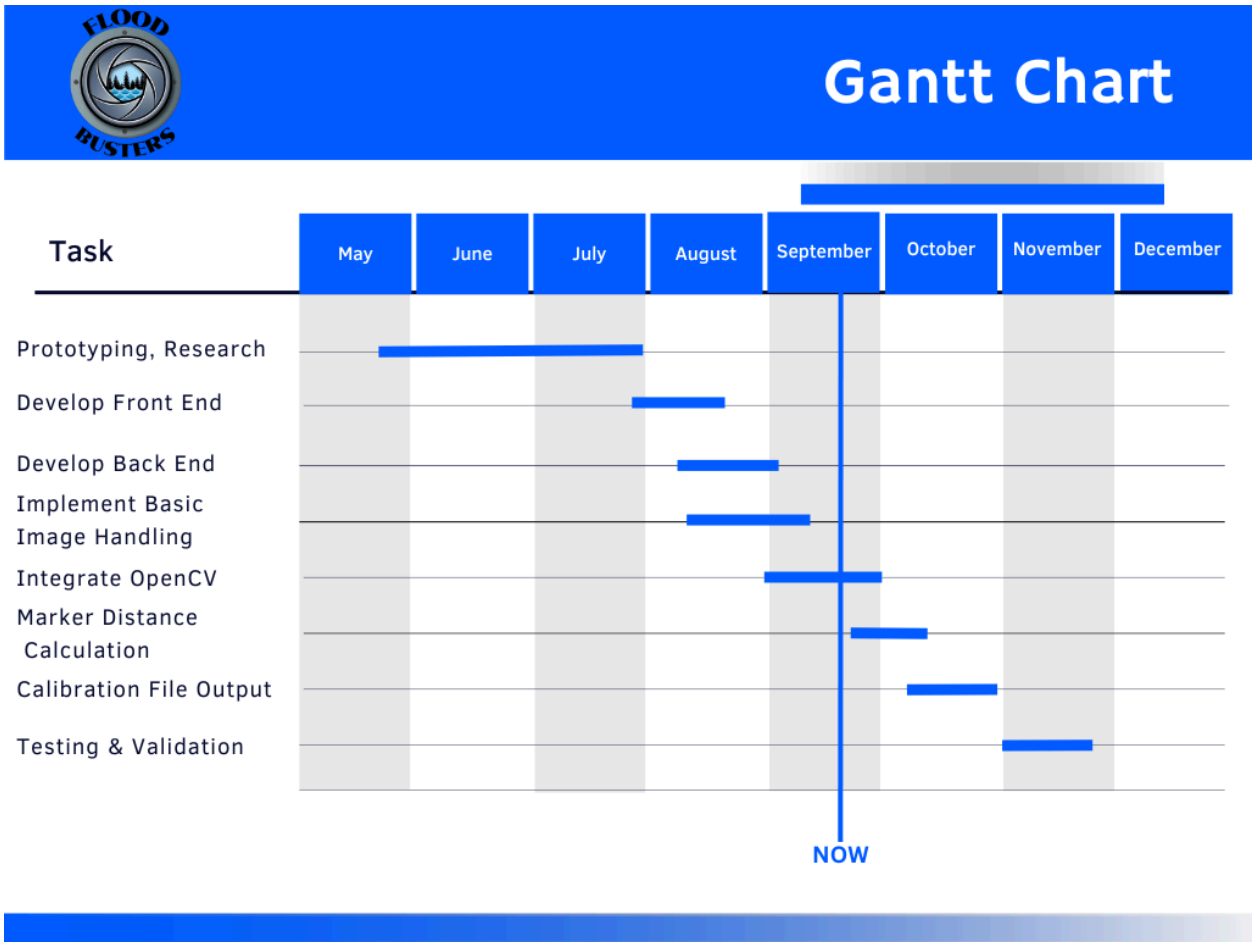
calibration_to_file(intermarker_distances): Outputs the created calibration data to a file for easy use (likely JSON).

4.6.4 | UML Diagram



5 Implementation Plan

5.1 | Gantt Chart



5.2 | Develop Front End

This module focuses on creating the visual elements of the workbench, including the buttons and sliders. This phase is nearing completion, with final touch-ups planned.

Design and development were completed in Weeks 11-12, with final touch-ups ongoing, the front end is mostly complete. The team needs to complete the minor adjustments in progress.

For the front end, we have Noah, Jennie, and Dylan finishing up the minor adjustments to the workbench. Noah is communicating where he left off for Jennie and Dylan to complete the remaining visual touch-ups. This will be completed before the first week of October.

5.3 | Develop Back End

This module focuses on developing the back end components to the front end elements of the workbench. It is nearly complete with touch-ups planned.

The main development of the back end was completed in early September and touch-ups will be implemented as required during future development.

Jade and Nate have been working on the back end and are making minor changes as the project progresses. The back end will be completed before the first week of October.

5.4 | Implement Basic Image Handling

The image handling module is responsible for uploading and saving images within the workbench.

This module is complete, and was completed in mid-September.

Jade and Nate completed most of the work on this segment of the project.

5.5 | Integrate OpenCV

The OpenCV module is the main module on the project and is still in progress. Currently, OpenCV functions on the workbench but needs significant work to get it to full functionality, as well as port it over from Python to JavaScript.

Work began on OpenCV in late August, and it is expected to be completed before the first week of October.

Jade and Noah have done most of the work so far on the OpenCV and will continue to work on this module.

5.6 | Marker Distance Calculation

This module uses the data from OpenCV to calculate the real-world distances between the markers.

Work on this module is expected to begin in late September and to be completed in mid-October.

Individual group members have not yet been assigned to work on this module.

5.7 | Calibration File Output

This module uses the data from OpenCV as well as the marker distance calculations in order to create outputs that are usable by HydroCams hardware.

This is the last module before testing and validation. It will be started in early October and finished in late October.

Individual group members have not yet been assigned to work on this module.

6 Conclusion

Flooding is a worldwide problem that impacts people's lives significantly every year, and is expected to only grow in frequency. Whether it's property damage, disruption of infrastructure, or even loss of human life, flooding is one of the most destructive natural disasters; due not only to its frequency, but the immense inefficiencies present in and difficulties involved with modern flood detection systems. The HydroCams project aims to revolutionize traditional flood detection systems with innovative, modern technology, reducing the time and resources required, in comparison to a traditional approach. With the current process requiring skilled personnel, expensive equipment, and archaic methods of data recording requiring excessive man-hours to perform, we feel we can significantly improve this process by implementing our proposed solution.

Throughout this document, we have delved further into the overall structure of our product, its architecture and partitioning, as well as a realistic development timeline for all modules. This documentation serves to better assist us in planning our solution implementation, as well as requiring us to think critically about how the project can be broken down and assigned to team members.