Technological Feasibility Paper

April 5, 2024

FLOOD BUSTERS

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

Sponsored By
Dr. Eck Doerry

Team Mentor
Vahid Nikoonejad Fard

Capstone Instructor
Michael Leverington

Version
2.0

Overview: This document outlines the main challenges in the development of the HydroCams project and how the team plans to approach them.

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 in order 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 all necessary metrological calculations through the use of computer vision and Structure from Motion. This allows water levels to be measured automatically based on which gauge points are obscured, providing local entities with accurate, real-time information of water levels and flood risks. This approach will likely be foundational to all future flood detection systems, as it fully addresses the key issues that plague nearly all current solutions: cost, accuracy, access to real-time information, and efficiency, which will save lives and potentially millions of dollars in damages.

As of now, we are determining the roadblocks that will impact the development of the project. In the following pages, we will analyze those roadblocks, determine some valid solutions and alternatives, and compare and contrast them to determine which solution we are likely to pursue. Finally, we will determine how those solutions combine to form a cogent solution to the overall problem/project. We will demonstrate that solution in a system diagram outlining the final expected workflow.

2 Technological Challenges

This section will outline the main technological challenges we expect to face throughout this project. Our team has identified four major challenges that lay ahead: designing a visual workbench web application, implementing computer vision (CV) workbench computations, implementing the use of Structure from Motion (SfM), and lastly, deploying our software in a mobile application.

In designing the visual workbench as a web application, there are a few things to consider. It must be able to communicate with our server or otherwise obtain the images. The front end of the application must run on most browsers, in low power environments, and maybe even without Internet access. This requires offline processing and a degree of caching.

The CV computations are the focal point of the whole project, as they would allow for accuracy in measuring flooding. Ideally, this would be consistent to a certain degree with the SfM implementation. It must be consistent, efficient, and compatible in some form with the web app and mobile app. CV is a popular technology, and many options are available that will generally provide the same result, albeit with different languages or levels of support.

SfM is a supplemental feature to the main CV and is required to increase the accuracy of CV measurements. SfM is decidedly less popular than CV, but is a degree more complex. This leaves us with relatively fewer options, but those options are also based on some CV options.

A mobile application is just a matter of convenience for the overall HydroCams project. It would allow technicians to get accurate 3D measurements using CV and SfM while installing the physical camera, assuming they have a smartphone on hand. Capabilities are minimal, only requiring the ability to take and send pictures from the app.

3 Technological Analysis

3.1 | Web Application - Front End

This section of our technological analysis will explore the challenge of creating a robust and user-friendly front end for our web application, which is essential in order to ensure an optimal user experience. Furthermore, our front end must integrate and communicate seamlessly with our back end in order to facilitate real-time data exchange and dynamic updates.

3.1.1 | Ideal Solution

- **Ease of Use:** Choosing a framework that is easy to learn and implement is critical in achieving quality software, minimizing development times, and ensuring maintainability in the future. Our team will place a heavy emphasis on choosing a framework that we are all comfortable working with.
- **Intuitive and Responsive Interface:** Our web application must have an easy-to-use, highly-responsive user interface that functions seamlessly across various browsers. It is also crucial that our workbench tool can be easily operated by non-developers, and that interaction happens dynamically, without the need for browser reloads.
- **Performance:** It is paramount that we select a front-end framework that is lightweight and capable of quick load times. This also involves selecting a framework that can quickly and easily communicate with our back end.

3.1.2 | Alternatives

- **React:** React is a very popular JavaScript framework developed by Jordan Walke, a Meta software developer, and released in 2013. React is known for creating dynamic UI systems that communicate with a server in real time. React also offers component reusability and a suite of highly reliable development tools that make the framework a promising choice, but can also slightly raise the learning curve.
- **Svelte:** Svelte was released in 2016 by creator Rich Harris, making it relatively new when compared to other frameworks. Svelte is unique as a web framework as it is a compiler and does not use a virtual DOM like React. In essence, much of the work that is traditionally performed by the browser at runtime is instead completed at build time. This allows Svelte to offer incredible speed when compared to other frameworks. Svelte also relies on traditional HTML, CSS, and JS/TS syntax and concepts, making it very easy to learn.
- **Vue.js:** Created by former Google engineer Evan You in 2014, Vue.js has rapidly grown in popularity and is now similar in terms of usage statistics to React and

Angular. Vue.js is a progressive JS framework, meaning features can be gradually incorporated into a project as needed, and can even be integrated alongside other frameworks. Vue is largely respected for its simplicity and flexibility.

3.1.3 | Analysis

React:

- **Ease of Use:** React's giant community and widespread implementation means that there is a substantial amount of documentation, tutorials, and forums to ease the learning process and support troubleshooting. Furthermore, the extensive library ecosystem of React enables developers to quickly and easily find tools for nearly any challenge. However, building a React project from scratch can require a large amount of setup and unintuitive boilerplate code which adds unnecessary complexity to the development process, and increases development time. Building a React project can also require the implementation of complex state management, especially in large applications, which also steepens the learning curve.
- **Intuitive and Responsive Interface:** React allows for the creation of intricate UI elements that can be reused thanks to React's component-based design, which also enables developers to maintain cleaner, more organized codebases. React also has a massive community and library ecosystem, offering a plethora of options when it comes to designing a responsive interface. Many of the most popular web applications utilize React, including Instagram, Imgur, and Airbnb.
- **Performance:** React optimizes and reduces render times by relying on a virtual DOM. A virtual DOM is a virtual representation of the DOM in memory, meaning that the only parts of the DOM that have changed need to be re-rendered. This makes React web applications significantly faster and more efficient than those that do not utilize a virtual DOM-based framework. However, React's massive ecosystem can result in unnecessary overhead in the case where additional feature sets or state management libraries are needed, ultimately increasing the bundle size and lowering performance.

Svelte:

- **Ease of Use:** Svelte syntax is extremely similar to normal HTML, CSS, and JavaScript, making it very easy to learn for most developers. The reactive nature of Svelte allows developers to avoid dealing with state management systems or a virtual DOM. The compiler approach also minimizes the amount of boilerplate code required for setup, leaving more time for meaningful development. However, as Svelte is a newer framework, there is a smaller community and therefore fewer learning resources available. Similarly, some IDE's have not yet integrated Svelte support, which may affect ease of use in specific cases.

- **Intuitive and Responsive Interface:** Svelte supports responsive design by including various transitions and animations out of the box. Svelte is also actually a language in itself, meaning it allows code to be reactive by default. The minimal boilerplate code required for Svelte allows developers to spend more time designing and implementing intuitive UI's, rather than having to go through a lengthy setup process.
- **Performance:** Svelte shines in terms of performance relative to other front-end frameworks. Its key feature is that it performs a bulk of the work at compile time, rather than having to be performed by the browser at runtime, by translating the developer's code into highly optimized vanilla JavaScript. This dramatically improves performance by eliminating runtime overhead and eliminates the necessity for something like React's virtual DOM.

Vue.js:

- **Ease of Use:** Similarly to Svelte, Vue uses HTML-based syntax, making it easy to pick up for most developers. Vue is also often seen as having a softer learning curve when compared to React. While not as massive as React's, Vue still boasts a large community that offers plentiful learning resources and extensive documentation. Vue also has a respectable ecosystem with a variety of libraries and developer tools. Lastly, the modularity and flexibility offered by Vue along with its focus on gradual adoption allows for a softer learning curve.
- **Intuitive and Responsive Interface:** Vue's emphasis on gradual adoption also allows developers to incorporate Vue elements only where needed, without relying on it entirely for the front end. This flexibility offers developers more freedom when developing UI elements. Similarly to React, Vue is component-based, meaning complex UI elements are easily reusable, saving development time and improving consistency of the interface in general.
- **Performance:** Vue.js can achieve excellent performance, outshining React in smaller applications. Overall, Vue is also more lightweight than React and similar frameworks, allowing for a smaller bundle size and greater performance. For large applications, Vue can require more specific tuning, heightening the learning curve and, in some cases, lowering performance. Vue also supports server-side rendering through external add-ons, which can increase performance, but can also dramatically increase the learning curve and development time.

3.1.4 | Chosen Approach

We will now assign a point value out of 5 for each of the frameworks from the previous section and calculate the average score (also known as the Average) for each characteristic the team is looking for in each framework. The Average that is the highest will determine which of the frameworks we want to use for our approach.

Table 1 | Web Application Front End Framework Scores

Framework	Intuitive and Responsive UI	Performance	Ease of Use	Average
React	4	4	3	3.66
Svelte	5	5	4	4.66
Vue.js	5	4	4	4.33

As shown in the table above, our analysis has shown that Svelte will likely be the best choice for our front-end framework. While React offers an expansive ecosystem of libraries and tools, it is likely more bloated of a framework than we require. Furthermore, React's virtual DOM system is typically great for performance, but falls short of Svelte compilation approach. React's learning curve is also typically regarded as being steeper than that of Vue's, while Svelte boasts the easiest learning curve. Vue offers greater UI customization and personalization than React, as well as greater performance on smaller applications, yet still falls short of Svelte in terms of ease of use. We are confident that Svelte will enable us to create a highly responsive and efficient web application, while minimizing development time thanks to Svelte gentle learning curve and easily understandable syntax.

3.1.5 | Feasibility

Our team plans to prove feasibility of our chosen front-end framework by developing a prototype webpage using Svelte, with all of our necessary UI elements integrated. This will help our team further familiarize ourselves with Svelte, and will show that it is fully capable of developing the front end for a reactive and high-performance web application.

3.2 | Web Application - Back End

3.2.1 | Ideal Solution

- **Scalability:** A vital part of a web application is its ability to keep up with user demand and the back-end tools need to be able to accommodate that. Scalability is necessary to ensure a smooth user experience, allowing for growth of the user base, and ensuring that the application is always available.

- **Community Support:** A tool that has more community support is more likely to have guides for use and would make error correction considerably easier. These tools would also have more libraries that lower development time.
- **Efficient Processing:** Data and requests being processed faster leads to a significantly smoother user experience. Because of this, tools with more efficient processing will lead to a more usable application.

3.2.2 | Alternatives

- **Django:** The Django framework was designed as a way to create web applications extremely quickly and easily for users with mid level experience, but also offers features that more advanced users would be looking for. Django was released in 2005 and is still a very popular option today.
- **Flask:** Flask is considered a microframework and was developed in Python. It is a minimalist option and does not have support for third party libraries. Flask was released in 2010 and is very popular among python developers.
- **FastAPI:** FastAPI was developed to be easy to learn and use with built-in features that help to reduce development time, reduce bugs, and promote efficient programming. FastAPI is the most modern of the considered frameworks having been released in 2018.

3.2.3 | Analysis

Django:

- **Scalability:** Django prides itself on its ability to scale. Its primary scaling method is to use more hardware to compensate for implementation time. This framework is used in very high traffic applications proving its ability to handle high user loads while maintaining a good user experience.
- **Community Support:** Django uses Python, which is a very common language, ensuring a base level of support but also goes beyond that. It is very popular and therefore has a large community of users.
- **Efficient Processing:** The Django framework was designed with performance in mind. Its ability to handle high load applications and maintain reasonable response times ensures efficient processing.

Flask:

- **Scalability:** Flask has the ability to scale well, however, its lack of built-in scalability options mean that it would take significantly longer and would be more difficult to match the scalability of the other frameworks.

- **Community Support:** Flask is heavily reliant on its large and contributing community. It has an incredible amount of extensions available and having been around for a while, it is safe to say that it has strong community support.
- **Efficient Processing:** Flask's lack of built-in features also leads to a longer and more difficult development process to achieve efficient processing. While this is somewhat compensated for by its high community support, it is still not enough to keep up with the other frameworks.

FastAPI:

- **Scalability:** FastAPI has the modern tools available to scale without too much development time, however, it does not seem that scalability is a focus or strong suit of this framework.
- **Community Support:** Being the newest of the considered frameworks, FastAPI has a heavy disadvantage in the community support category. It lacks the various extensions and plugins that the Flask framework has.
- **Efficient Processing:** FastAPI has a definite focus on quick and efficient processing as well as development. It prides itself on its performance and uses modern techniques to achieve this.

3.2.4 | Chosen Approach

We will now assign a point value out of 5 for each of the frameworks from the previous section and calculate the average score (also known as the Average) for each characteristic the team is looking for in each framework. The Average that is the highest will determine which of the frameworks we want to use for our approach.

Table 2 | Web Application Back End Framework Scores

Framework	Community Support	Performance	Scalability	Average
Django	4	5	5	4.66
FastAPI	2	5	4	3.66
Flask	5	4	3	4

3.2.5 | Feasibility

Our team plans to prove the feasibility of our chosen back-end framework by creating a prototype using Django. We will use this prototype to test Django's performance and scalability under load and also use the opportunity to explore community extensions.

3.3 | Implementing Computer Vision Computations

3.3.1 | Ideal Solution

With computer vision computation being the primary focus of the application, it is imperative that a well-supported and stable library is chosen to facilitate the implementation. Computer vision computations consist of complex, computationally expensive algorithms which are non-trivial to implement. Due to their complex nature and expectations of future development, we have chosen to use an open source library instead of developing them in-house. Here, we will explore three well known open source libraries and evaluate them on the basis of documentation/community support, interoperability, and performance.

- **Documentation / Community:** Having good documentation and community support will ensure correctness and ease of implementation which is crucial for a complex library. This will also aid in the implementation with other libraries and platforms.
- **Interoperability:** With this project being implemented as a web application, mobile applications, and the possibility of it being expanded beyond its initial scope of marker identification and calibration, interoperability is required for a computer vision computation toolset.
- **Performance:** Due to the application being hosted on both a server and a low powered mobile device, such as a tablet or mobile phone, performance, and processing time will be an important factor to consider. Also, the library must have multiple, easy to implement, modules/functions to choose from.

3.3.2 | Alternatives

- **OpenCV:** OpenCV is a free, open-source, cross-platform library released under the Apache License. The library was created by Intel and released to the public in 2000 and is still receiving stable updates. It provides a vast library of modules for image processing.
- **Pillow:** Pillow is the forked version of the original Python Imaging Library which supports Python 3.x. Pillow was released in 2010 and continues with stable releases.
- **scikit-image:** scikit-image is an open-source image processing library that is part of the SciPy Toolkit. The library was initially released in 2009 under the BSD License and is still currently in use receiving stable updates.

3.3.3 | Analysis

OpenCV:

- **Documentation / Community:** Has extensive documentation on the OpenCV website with examples and tutorials. It also has active communities on GitHub, Stack Overflow, Reddit, and other educational sites.
- **Interoperability:** Operating system support includes Linux, macOS, Windows, Android, and IOS. Provides language support for C/C++, Java, Python, MATLAB/OCTAVE, and other APIs. Also has a subset of functions for use with JavaScript that was released as OpenCV.js for use with web application platforms.
- **Performance:** Vast library providing both basic and advanced image processing modules/functions, documentation, community support, and multiple language options can expedite implementation time. Provides both functional and statistical machine learning libraries giving options for computational and performance restraints. It also provides optional GPU acceleration support.

Pillow:

- **Documentation / Community:** Provide some documentation and examples on Pillow's website along with tutorials. Tutorials can also be found on various educational websites. With its basic functionality, there are limited resources available.
- **Interoperability:** Operating system support includes Linux, macOS, and Windows, with language support in Python
- **Performance:** Provides primarily more basic image processing modules, which provide the flexibility of implementation but would require extensive in-house development to expand the library to meet project requirements.

scikit-image:

- **Documentation / Community:** Provides some documentation and examples on scikit-image website. There is an active community on various forums such as GitHub, Stack Overflow, and other educational sites, but it is also intermixed with the SciPy community as a whole.
- **Interoperability:** Operating system support includes Linux, macOS, and Windows, with Language support in Python, Cython, and C. It Requires support from NumPy and SciPy libraries.
- **Performance:** Provides primarily more basic image processing modules, which provides the flexibility of implementation. However, this would require more development time and additional in-house development to further expand modules.

3.3.4 | Chosen Approach

We will now assign a point value out of 5 for each of the frameworks from the previous section and calculate the average score (also known as the Average) for each characteristic the team is looking for in each framework. The Average that is the highest will determine which of the frameworks we want to use for our approach.

Table 3 | Implementing Computer Vision Computations

Library	Documentation \ Community	Interoperability	Performance	Average
OpenCV	5	5	4	4.67
Pillow	2	2	1	1.67
scikit-image	3	2	2	2.33

As shown in the table above, OpenCV will most likely be the library chosen for the majority of our computer vision computations. It is a vast library, has good documentation / community support, and meets our desired performance needs. While scikit-image appeared to be promising, it did not have as robust of a library and would have required additional in-house development to match the capabilities of OpenCV.

3.3.5 | Feasibility

To prove the feasibility of the chosen library, our team plans to develop several testing scripts utilizing modules we expect to need for the application. The test method will consist of capturing a set of test images, implementing functions from existing libraries, and further evaluating them based on ease of implementation and results.

3.4 | Structure from Motion Implementation

The use of Structure from Motion (SfM) is crucial to the measurement of gauging points at varying depths. While measurement on a 2D plane is possible, it will never be as accurate as it cannot account for depth. In conjunction with the mobile app, SfM can be implemented quite easily upon HydroCam installation. By simply taking 2 (or more) images of the same area from different angles, accurate 3D measurements can be obtained very quickly.

With Python as our chosen language, there are a handful of open-source SfM libraries already available, and it would potentially be possible for us to implement the required methods ourselves.

3.4.1 | Ideal Solution

- **Documentation / Community:** For a library written by someone else to be useful, it must be extremely self-descriptive. This generally requires extensive documentation or a very responsive community built around the library.
- **Interoperability:** When a project has multiple parts, those parts must be able to communicate with each other in some fashion. For computations, especially those of basic computer vision and later, SfM, interoperability is key.
- **Performance:** As these computations could be running on relatively low-power devices (like smartphones), the library must be quick, efficient, and capable.

3.4.2 | Alternatives

- **In-House:** An in-house production of a basic SfM library built by the team over the course of the project. Since we do not need many of the advanced functions of other SfM libraries, this solution would provide us with the minimum needed to use in the final phase of the project. This is generally not a suggested approach, especially for a capstone project, but it is an approach nonetheless.
- **OpenCV SfM Module:** OpenCV is a popular computer vision library that was first released in 2000 by Intel. Because it is also an important component of the computer vision challenge, it will not be further elaborated on. It does have an optional SfM module that would be useful for the SfM calculations.
- **OpenSfM:** Released in 2013, OpenSfM is a popular Python library used for constructing 3D environments with options for external integrations and JavaScript viewers.
- **pyTheiaSfM:** The most recently released library we were able to find, pyTheiaSfM is a Python translation of another project, Theia. It was released in 2015 and serves as a general-purpose SfM library, but it is not intended to be as in-depth as the original Theia library.

3.4.3 | Analysis

In-House

- **Documentation / Community:** As we would be creating this library ourselves, the only documentation or community would be our own, assuming the project doesn't gain traction rapidly.

- **Interoperability:** The library would ideally be perfectly interoperable with our other frameworks. This would be facilitated by us, but it would be no easy task.
- **Performance:** We are not professionals (yet), and thus, our library would likely be unoptimized and potentially slow.

OpenCV SFM Module

- **Documentation / Community:** While there is some documentation for SfM, it seems buried and primarily focused on C++. The code is likely transferable, but without further information, the documentation is minimal. The community around OpenCV is quite large, with anything from forums to Slack channels.
- **Interoperability:** OpenCV is supported on a number of operating systems (Linux, macOS, Windows, Android, and IOS) and in multiple languages (Python, C++, Java, and JavaScript).
- **Performance:** Performance seems to be a primary focus of OpenCV, with them stating that the library is highly optimized. GPU acceleration is also a valuable asset.

OpenSfM

- **Documentation / Community:** While the existing documentation seems extensive, there appears to be a minimal community surrounding the project. There was a noticeable lack of dedicated subreddits, forums, etc.
- **Interoperability:** OpenSfM is only supported on Python, but it is built on OpenCV. As Python will likely be our language of choice, this is not necessarily an issue.
- **Performance:** OpenCV itself is already optimized, but OpenSfM also incorporates Ceres Solver, a C++ library focused on the optimization of large problems.

pyTheiaSfM

- **Documentation / Community:** There seems to be no real documentation for pyTheiaSfM itself, but extensive documentation for its source material. The source material is in C++, so the documentation is unlikely to be one-to-one.
- **Interoperability:** pyTheiaSfM is a standalone project, so there is minimal interoperability between our other systems. Python is useful, as it is in the same language, but that is not enough to fully integrate with our other modules.
- **Performance:** Theia itself has remarkable performance, based on Eigen and Ceres Solver. Whether this performance translates to pyTheiaSfM is unclear.

3.4.4 | Chosen Approach

We will now assign a point value out of 5 for each of the frameworks from the previous section and calculate the average score (also known as the Average) for each characteristic the team is looking for in each framework. The Average that is the highest will determine which of the frameworks we want to use for our approach.

Table 4 | Structure from Motion Library Scores

Library	Documentation / Community	Interoperability	Performance	Average
In-House	0	0	0	0
OpenCV SFM Module	3	5	4	4
OpenSfM	3	4	5	4
pyTheiaSfM	1	2	3	2

As shown above, there is some contention about our final choice for SfM. An in-house approach is unlikely to work and requires far too much work. pyTheiaSfM is also unlikely to work for our purposes. Finally, OpenCV SFM and OpenSfM have similar scores. OpenSfM is basically just an optimized version of the OpenCV SFM module but doesn't have the same community or support that base OpenCV does.

3.4.5 | Feasibility

In order to prove the feasibility of the SfM implementation, we plan to implement a series of test processes and images that would mimic those found in a HydroCams installation. We will have to implement functions from both libraries and determine which works better with our other options and which is more performant.

3.5 | Mobile Application Applying the Automated System

The mobile application is the final step in the whole project, when the automated system is completed and can be developed into an app. Developing an application will make it easier to collect data in the field. The current process involves hand-writing the data and reporting it back to the office. The app would save time and improve accuracy by allowing users to take pictures in the app and apply the features the workbench offers.

Those features are being able to take a picture and processing that picture. In the processing phase, the image will be scanned for gauging points and show the calibration information. That calibration information will include the distance between any found gauging points. If the scan is incorrect, the user will be able to modify and correct the detection.

3.5.1 | Ideal Solution

- **Community / Documentation:** We want to consider the community support and what is available. This will give the team a reliable resource if we face something that seems too difficult to conquer.
- **Ease of Use:** We want to make sure that the team is capable of using the framework without too much of a challenge. Additionally, the team wants to be able to minimize the development process, maintain quality software, and have an easy time maintaining the application.
- **Performance:** We do not want users to struggle with taking photos or making the edits that need to be made on the gauging points. So, the application must run smoothly without hassle when it is being used. Especially during the development process, we do not want our team to struggle with activities like testing features.

3.5.2 | Alternatives

- **Flutter:** Flutter is a framework that the team came across due to word of mouth and wanting to further research it. Especially because of how well known they are for working on both Android and IOS. Flutter was launched on December 4th, 2018, by Google. The goal of Flutter is to be open-source and free. In addition, Flutter has the ability to use already existing code and apply it to both Android and IOS. This framework is supposed to be an easier way to get into app development because it uses only one codebase and gives guidelines to basic functions.
- **Ionic:** Ionic is a framework that has been recommended to the team by Dr. Doerry, who suggested the team look into it since he knew it was a popular framework that is used for application development. He has used it for other projects that needed mobile apps and found it to be very adaptable. Ionic was founded in 2012 by developer and designer pair Max Lynch and Ben Sperry. This framework is well-known for cross-platform application development and can use other frameworks within it. Lastly, Ionic has the feature of having a native-like style or a hybrid style, depending on the coder's choice.
- **React Native:** React Native was another framework that Dr. Doerry recommended the team look into since it is another very popular framework. React Native was launched in 2018 by Facebook. It is well-known for its reusability which is a useful feature. One notable feature is when we define a component, we can use it multiple times because this framework allows that component to be shared between projects or applications. The benefit of this is that we can reduce redundancy in our code. Additionally, it has the support of using different libraries that allow the developer to have more choice in their style of coding.

- **Svelte Native:** Svelte Native was a framework that team member Noah found to be resourceful. Svelte Native was launched in 2019 by Harttle Land. So, this is the most recently released framework out of our options. This one has a unique approach to testing app development. When the app is running, it pops out a phone outline to still function as an app similar to a video link where it shows up on its own.

3.5.3 | Analysis

Flutter:

- **Community / Documentation**: The Flutter community provides different outlets users can use to find more information about Flutter. On the website alone, they provide a 'Cookbook' that shows simple builds that can be made through Flutter. There is also a feature about learning what is currently available through the 'What's new' tab to stay current with the framework. Lastly, it has a variety of social media outlets, including 'X' (also known as Twitter), YouTube, Medium, and GitHub, if the team wants to dive in deeper with support.
- **Ease of Use**: Although Flutter seems to be an easy framework to get into app development, it does have a downside. Even if it is cross-platform, there are times when the team will have to pay attention to the differences between platforms. If we use a current library like an image picker, we will have to make sure it is still functional for Android and IOS since the library does not adapt to both scenarios.
- **Performance**: For the most part, we found Flutter's performance to be decent. At times, it does not run the best and can take a bit to load. However, it does work well when applying widgets, which are base templates that the website provides. Otherwise, it still provides the functionality we are looking for, being cross-platform and running decently for the most part.

Ionic:

- **Community / Documentation**: The community is current with its resources. They have a solid outreach system where anyone can reach out on 'X', Discord, or GitHub. In addition, they still have the traditional system of reading through blogs and forums on the Ionic website.
- **Ease of Use**: Ionic is not difficult to use for app development. The website alone provides the fundamentals on how to create an app and provides an easy process to form an app that works on Android and iOS and can also form a website. In addition, there are plenty of resources like videos and tutorials to make it easier to understand the creation of the app. Lastly, it seems to not have a difficult time adapting to different platforms and is truly under a single codebase.

- **Performance**: When creating a template app, not only was the creation process easy, but had great performance. The app was run through a terminal and provided a link to show the current output without a long wait time. There seems to be no current struggle with the performance of the app, and is very reliable.

React Native:

- **Community / Documentation**: React Native does have community options through the website, but they are not as vast as the other frameworks. This framework still has the fundamentals for developer implementation listed on the website. The community is still up-to-date about the framework and does have GitHub for additional sources.
- **Ease of Use**: React Native makes it easy to customize features to the developer's liking, like using a component in another place throughout the project. Adapting other libraries that are not normally used throughout the framework. Lastly, it provides guidance on how to apply API in small ways, which is important if we want to adapt our back end to the mobile application.
- **Performance**: This framework has great performance. When it comes to testing the code during development, it has 'Hot Reloading', which is a helpful tool. This tool allows the developers to see the changes made to the code in real time, potentially reducing test time.

Svelte Native:

- **Community / Documentation**: This community does not have as many resources, but does have some on GitHub. As for documentation, they provide guidance on how to start the app and a simple implementation that can be used to get the app started.
- **Ease of Use**: The most interesting feature that makes it easier for the development process is the integrated tools it already provides. The Svelte command line interface (CLI) is included within it, which allows features like the 'Hot Reloading' which has a clean interface when the mobile app is being tested out. Lastly, it allows for reusable components allowing the components to be spread across files instead of piling into one big file.
- **Performance**: Similar to React Native, Svelte Native provides 'Hot Reloading' which can reduce testing time for the development process. It can reduce testing time for the development process because it displays the code in real time while the other frameworks did not. This is beneficial because we want to have enough development time, and every second counts. In addition to it, the compiler has the option to implement dead-code elimination, where it does not compile unused

code, saving some space and complexity. Overall, Svelte Native has smooth performance, a clean framework, and it is still on the newer side.

3.5.4 | Chosen Approach

We will now assign a point value out of 5 for each of the frameworks from the previous section and calculate the average score (also known as the Average) for each characteristic the team is looking for in each framework. The Average that is the highest will determine which of the frameworks we want to use for our approach.

Table 5 | Mobile Application Scores

Mobile Application Framework	Community / Documentation	Ease of Use	Performance	Average
Flutter	4	2.5	3	3.16
Ionic	4	4	3	3.66
React Native	2.5	4	4	3.5
Svelte Native	2.5	3	5	3.5

After seeing the scores from Table 5, we have selected Ionic as our mobile framework. This framework has the simplest camera implementation to our knowledge, since it has a plugin feature capable of maintaining the cross-platform compatibility that we are looking for. Unfortunately, Ionic does not have ‘Hot Reloading’, where we would have to compile the code and open it each time we update our code. However, it was found to be successful in maintaining one coding base.

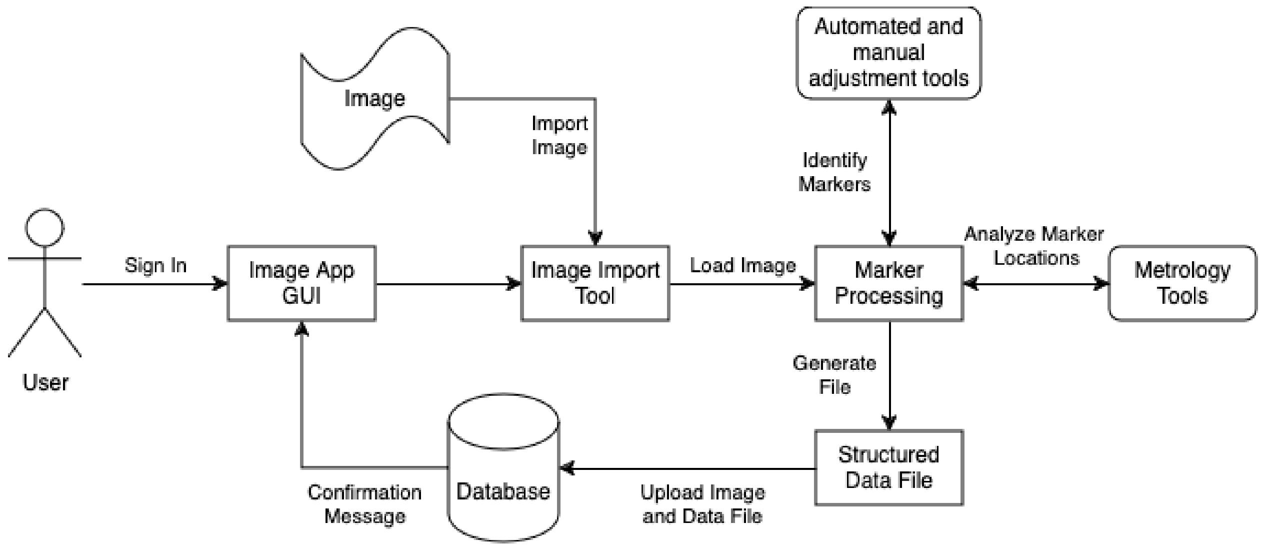
3.5.5 | Feasibility

For further investigation on Ionic, we will be looking into making prototypes to get a better understanding of the functionality of the framework. We will be testing the functionality of both the Android and IOS apps to ensure cross-platform consistency.

4 Technology Integration

Based on the analysis of expected technical challenges and solutions provided in the previous sections, this section will demonstrate how we intend to integrate all of them into a working model. Figure 1 depicts a basic graphical representation of the architecture we plan to implement.

Figure 1 | Project Workflow



A user will first launch the application and be prompted to import an image of the site where the markers have been placed. Once the image is uploaded, a marker processing toolset will become available. Within the marker processing toolset, options for automated and manual adjustment tools for locating, marking, and editing gauge points can be selected and run. Multiple iterations can be run to fine-tune the image. Additional toolsets, such as a histogram, exposure, contrast, etc., will also be available if other modifications are required. The user will also be able to use the metrology toolset for additional analysis if needed. Once the image has been processed, a structured data file will be produced. This structured data file will contain information about the markers in the image, such as a label, pixel location, and size. These will be stored in a database along with the original and annotated images for future reference and review.

5 Conclusion

In 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 to not only their frequency, but the immense inefficiencies and difficulties involved in modern flood detection systems. The HydroCams project aims to revolutionize traditional flood detection systems with innovative, modern technology, reducing the time and resources required compared to traditional approaches. 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 paper, we have explored the technical challenges we expect to encounter, along with the feasibility of our proposed solutions. In order to ensure we can implement all functionality requested by the client, we have suggested multiple viable solutions, along with a number of contingencies. Those solutions include: Svelte and Django for development of our web application, OpenCV for computer vision computations, OpenCV SFM Module or OpenSfM for the Structure from Motion library, and Ionic for our mobile application framework. We chose these tools based on both our analysis of their technical capabilities and their potential to deliver an efficient, powerful, and user-friendly system.

In order to address the challenges unearthed through our analysis, we intend to implement our proposed solutions, aiming not only to achieve, but exceed the expectations of ourselves and our client. The HydroCams project promises to offer quick and accurate flood monitoring at a fraction of the current cost and complexity, which would serve as a significant leap forward in disaster preparedness. We are confident that HydroCams will become a pivotal tool in safeguarding communities, saving lives and resources along the way.