COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Swim Tech LLC

Ethan Richards
Mason Shandy
Morris Gray
Grant Bohlmann

Revised December 9, 2023

CSCI 370 Fall 2023

Advised by Adjunct Prof. Lorenzo Gallegos

Table of Contents

I. Introduction

Project Description: The purpose of this project is to create a tool for swim coaches to assist in training swimmers using computer vision. We want to streamline the process of uploading a video to a web application and returning it to the coach who can then show it to the swimmer.

Client Information: Swim Tech LLC is a swim education company that wants to use software to assist swim coaches. They host swim teams in Colorado and use a technique-based approach to teaching swimming. Swim Tech wants this software to help swimmers visualize their resistance against water, which can then assist swimmers in improving their stroke form. Swimmers are typically entering or in competitive swimming during their teenage years, so improving form is crucial during this developmental period. Users of the software will be swim coaches who work with Swim Tech LLC, and to some extent, the swimmers who are being shown themselves swimming and the visualizations of their swim strokes.

Previous Software Revisions: The basis of existing code is an existing GitHub repository using code in C++ and libraries from OpenCV with a computer vision model. This was developed by a previous Mines Field Session team in Fall 2020. However, there is a need for continued effort because the process for using the previous software is extremely complicated and slow. It is also still relatively basic in terms of what it can provide, so improvements in visualization and fluid modeling could help swimmers. Our client also wanted a new web app, which was not a part of the original software.

Data Sources: Strictly speaking, there is no data in our project in terms of traditional numbered data, but we will rigorously analyze videos which can produce resistance/fluid model data about the swimmer and the efficiency of their form as well as resistance through the water. This data will be used on our computer vision machine learning model to better train the model to recognize form.

Software Maintenance: Future field session teams may maintain the software, as well as our client himself as he has relevant technical experience but limited time, which is why we're making the bulk of the product. Our client has also said that there is opportunity for future work and internships which will involve improving and maintaining the final software.

II. Functional Requirements

The functional requirements of this project include two pieces of software: a web application and a computer vision model. The computer vision model is an image/object recognition model which recognizes the drag of the water flow against the swimmer's body. The web application serves as the central hub for Swim Tech coaches to utilize the model via uploading and managing their videos. We also show deliverables for the visualization model to limit the scope of the project as the visualization model *and* web application are large pieces of software that may take more than three months to fully finish.

Web Application Requirements:

- Ability to upload raw videos, process them, and return/download video overlays (via web app or email).
- Base level authentication (email, username and password).
- Return overlay to the account that uploaded the initial video.
- Browser-based web application that coaches can use to show swimmers the visualization model video.
- Lessen the amount of time it takes for the video to process and then be returned to the end user.
- The web app must be responsive for coaches using tablets or phones to view the content.

Visualization Model Requirements:

- Convert current C++ code to Python for our Django backend, while still using OpenCV.
- Automate head and hip detection (currently needs to be manually selected by coach).
- Visualize fluid resistance on the video overlay, specifically the swimmer's arm angle and head position.
- Metrics/POIs displayed on overlay.

Visualization Deliverables:

- Torso angle + head and hip detection (current system); should be easy and take ideally 1 month.
- Specific stroke improvements (at least 1 per stroke); should be medium difficulty and take 2 months or to the end of the project.
- Limb-to-hip or limb-to-surface distance overlay; should be medium difficulty and take ideally 1 month of the project.

III. Non-Functional Requirements

The non-functional requirements of this project include expenses/budget limitations, performance limitations, and security and reliability requirements.

Expenses: Our client is giving us a $200-$300 processing and total server cost for the DigitalOcean Droplet, Database, and Spaces (S3-like store). Therefore, using open-source such as OpenCV is encouraged, as well as other free datasets for training our model.

Performance: Quick video processing with a return time preferably within 24 hours. Doing the visualization in real time seems unlikely given performance limitations of a cheap DigitalOcean server and because of the limitations of the model.

Security: User accounts should have a basic username and password, so coaches only view their videos (and not the entire database). User accounts should also have permission levels of what they can and cannot view. All data should be safely sent across HTTPS and encrypted properly in the database.

IV. Risks

Technical Risks:

1. Learning curve of OpenCV
   Likelihood: Very Likely
   Impact: Moderate/Major
   For all of us to learn OpenCV, we will have to dedicate a lot of time towards the project. This may have knock-on effects through the project since more time is required to research/learn; the level of quality in the visualization may also vary depending on how much we have time to learn given our time constraints.

2. Processing power/processing time vs. cost
   Likelihood: Likely
   Impact: Moderate
   The coach may have to wait longer to receive the processed video if our servers can't handle it.

3. Migrating the C++ code to Python (losing functionality)
   Likelihood: Very likely
   Impact: Major impact
   There may be some functionality lost or some features that don't transfer over to Python in the migration of the original C++ codebase to Python.. In this case, we may just have to write the code from scratch in Python, which also takes up a lot of our time..

Non-Technical Risks:

One risk in our project is the filming of student swimmers as they swim. There are consent waivers put in place to ensure the swimmer is okay with this; however, it is still very unlikely that it causes anybody any risk. If this risk was realized, the impact would likely be minor anyways as faces are not easily recognizable underwater. If a student does not want their data saved then they will be able to delete any old video, and they will no longer be filmed for the modeling process.

Another risk to consider is the injury caused from a bad swim stroke. If our program recommends an incorrect swim stroke, this could lead the swimmer to injury after incorrect repetition. This is very unlikely, as our code would have to fail and the swim coach would have to not notice the incorrect form. The impact could be moderate if it is a serious injury. If this risk becomes more likely, then we will make sure our code does not suggest anything, and allow the coach to make these changes.

V. Definition of Done

Minimal Feature Set: Be able to upload the video to the web app, get it processed by the visualization model, and be returned an edited video file of the swimmer swimming with resistance drag overlays.

Client Acceptance: The client wants to test out the software with existing videos and personally analyze the form of the swimmer comparatively to how our program is identifying any pitfalls in the swimmer's form. Once the software is accepted, the client may also want to test it with coaches in a public beta.

Product Delivery: The product will be delivered once the client is satisfied with the video upload process as well as the resistance visualizations that we are producing. The product will be deployed once done and then integrated into the existing SwimTech workflow. Coaches will be able to use this application immediately with their swimmers and provide any feedback/bug reports to us as needed.

VI. System Architecture

The system architecture that we are implementing include three main tenants: the processing backend, the web app backend, and the web app frontend. The backend and the frontend communicate over HTTPS with basic requests and responses. The backend and frontend are also decoupled to be truly RESTful, with API endpoints on the backend designed with REST in mind. The frontend uses Vue3 and Vuetify. The backend runs on Django with DjangoRestFramework (DRF) and a PostgreSQL database. The backend communicates with our processing backend which is also written in Python, and communicates with Roboflow and OpenCV for the visualization model, as well as S3 for storing processed videos. This architecture is shown in Figure 1.

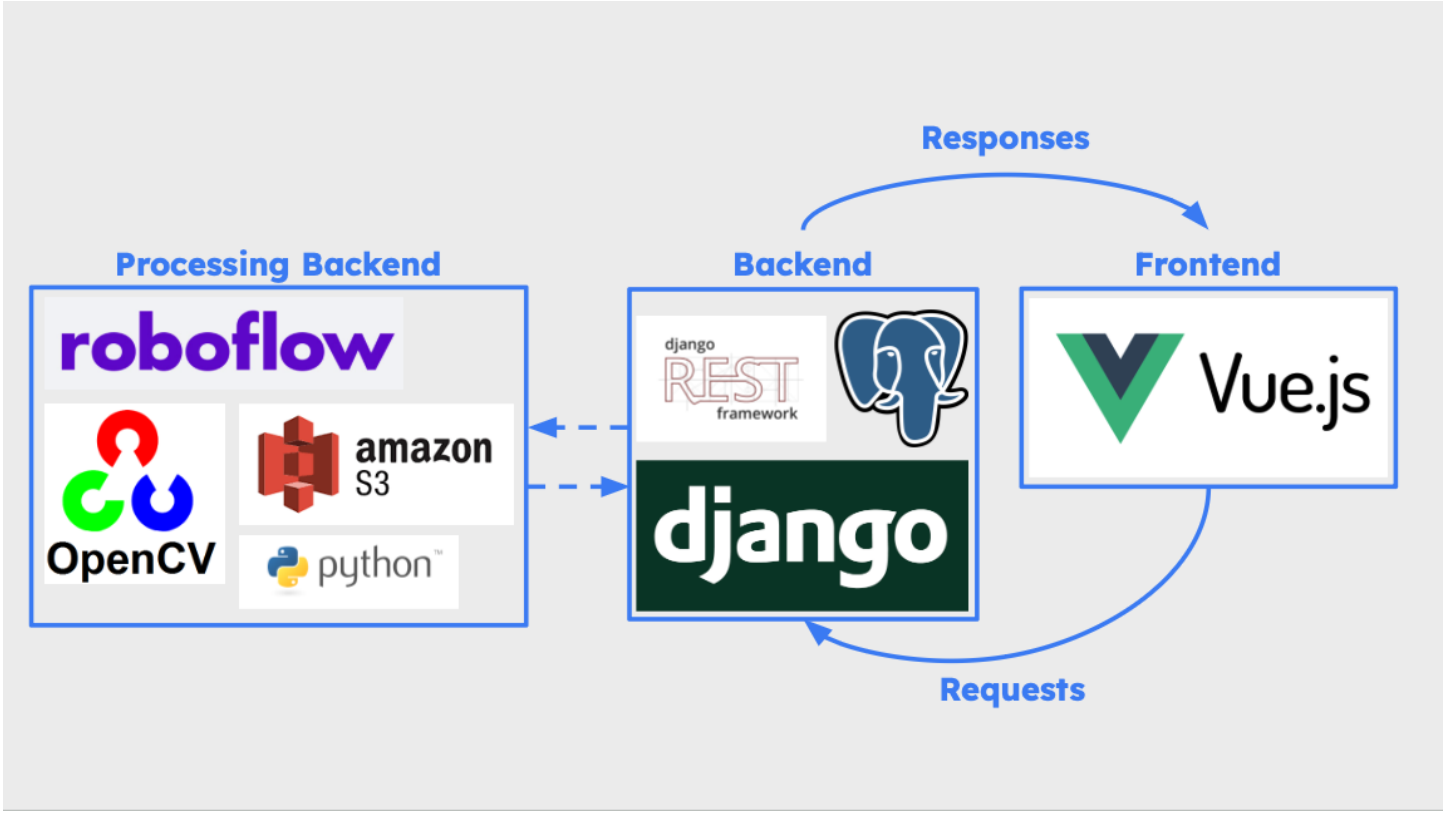

Figure 1, System Architecture

In Figure 2, we see the mobile dimension website mockups. We display the following screens: login, logout, sign up, upload video, all videos, and viewing a certain video. All of these properly scale to mobile dimensions with builtin responsiveness in Vue and Vuetify, our frontend frameworks.

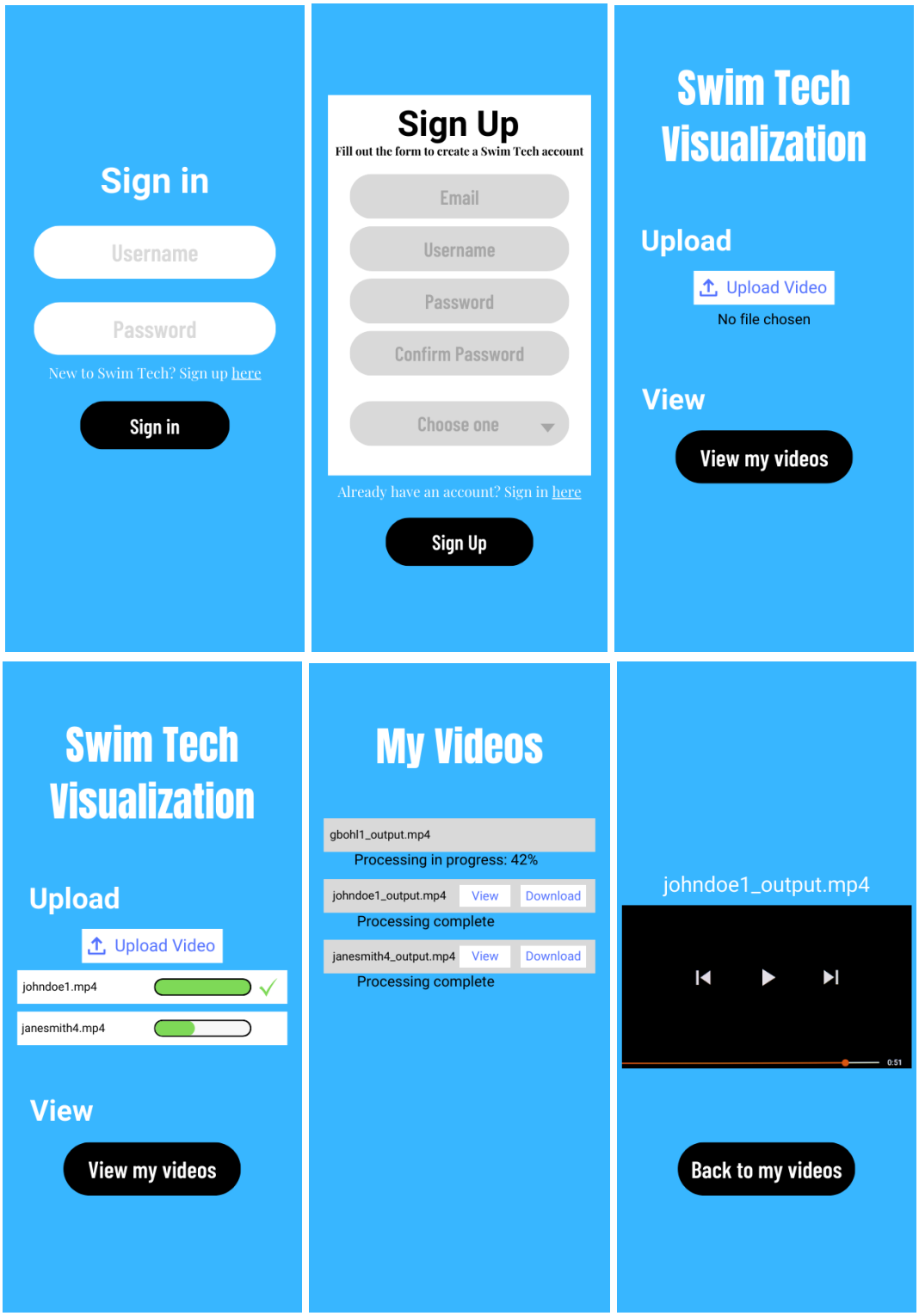

Figure 2, Mobile Website Mockups

In Figure 3, we see the desktop dimension website mockups. We display the following screens: login, logout, sign up, upload video, all videos, and viewing a certain video. These content layouts will scale appropriately for tablet size screens as well.

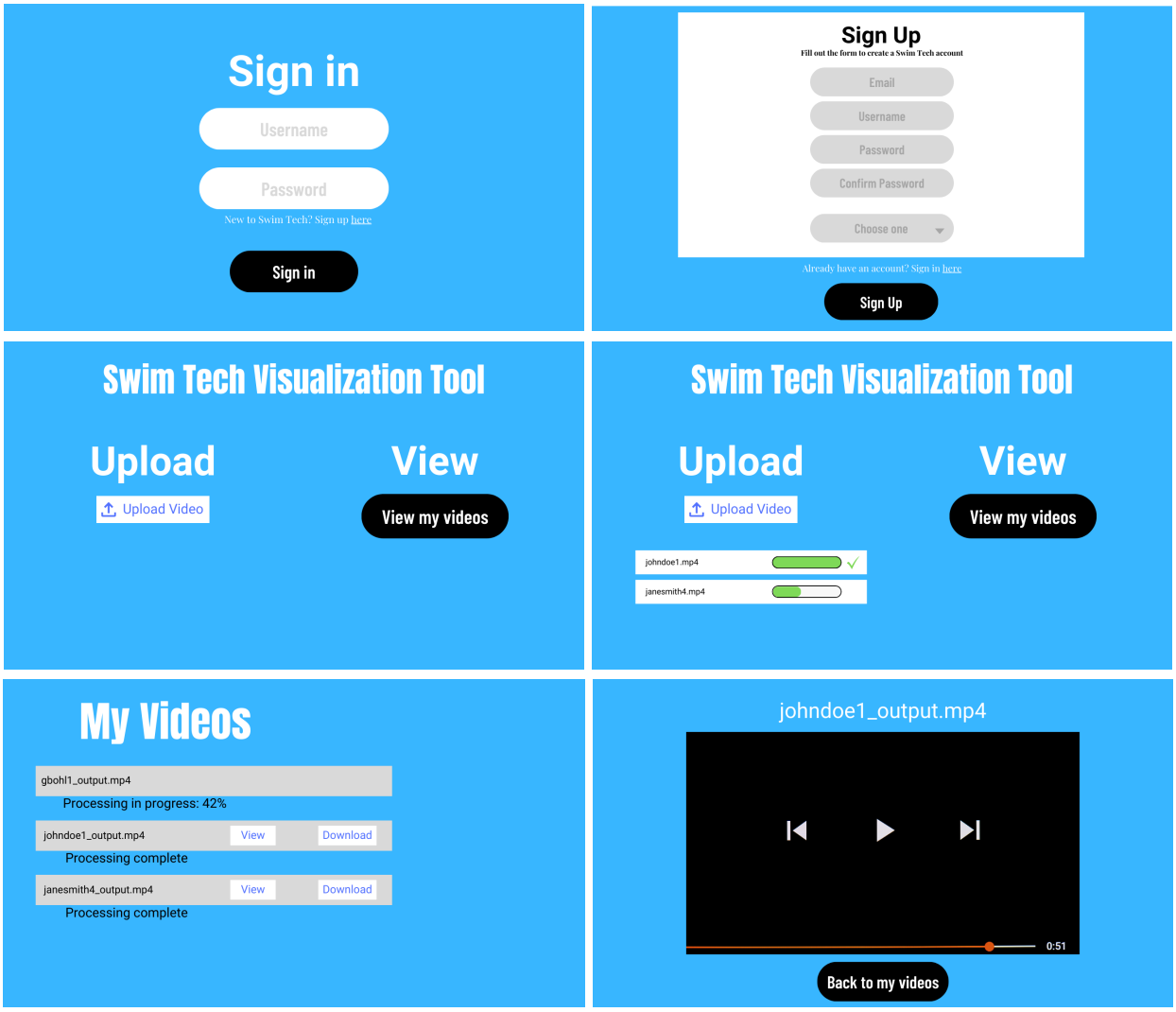

Figure 3, Desktop Website Mockups

Figure 4 shows our database ERD. We explain the database schema further in section VII, but as per the relationships between the schemas; videos have relationships to users (such as the coach who uploaded the video or the swimmer in the video), and users have a single role.

[REDACTED]

[REDACTED]

VII. Technical Design

The technical design and implementation of our product includes all that was mentioned in Section VI, as well as further relevant info listed in this section. Our design also introduced a novel opportunity to use threading to more effectively handle video processing, which will also be mentioned in this section.

We are also using session authentication for our authentication model because we want user sessions to be able to persist and be canceled if needed (as compared to JWTs, which have a static expiry date). DjangoRestFramework provides a fair implementation of this system to begin with, so we are using that. We are also extensively testing how session authentication (with cookies) works with the decoupled nature of our frontend and backend flow. Further, we are using Pinia as a state store on the frontend which can store user state which will be validated within the backend upon each request. Authorization with role permission checks are done upon each route too.

Figure 5 shows one of our novel implementations: the threaded processing model. The intended flow of the application is for a user to upload a video over HTTP, which is already a hefty task due to the limitations of HTTP. Then, the video will process, upload to S3, and then be able to be viewed by the user. However, the major problem with this approach is that HTTP quite simply cannot handle large file uploads and will time out, leaving a blank or forever loading screen for the user. At best, the request would take a long time.

Therefore, we implemented a threaded model for our video processing. When the video is uploaded to S3 from the frontend, the backend is made aware of this, and downloads the video from S3 once available. A response is sent immediately, while the backend begins processing anyways. This threading also lessens the burden of memory on the web server's main thread, and with more computing resources, threaded processing may also speed up processing; instead of having one video processing at a time, we can have multiple in parallel.

Threading is also used for re-encoding and uploading the video. We are re-encoding the video with ffmpeg, which requires threading to be used or else it will crash the web server's main thread. Once both the processing and re-encoding is done, the user will be able to see the processed video alongside the original video on the frontend.

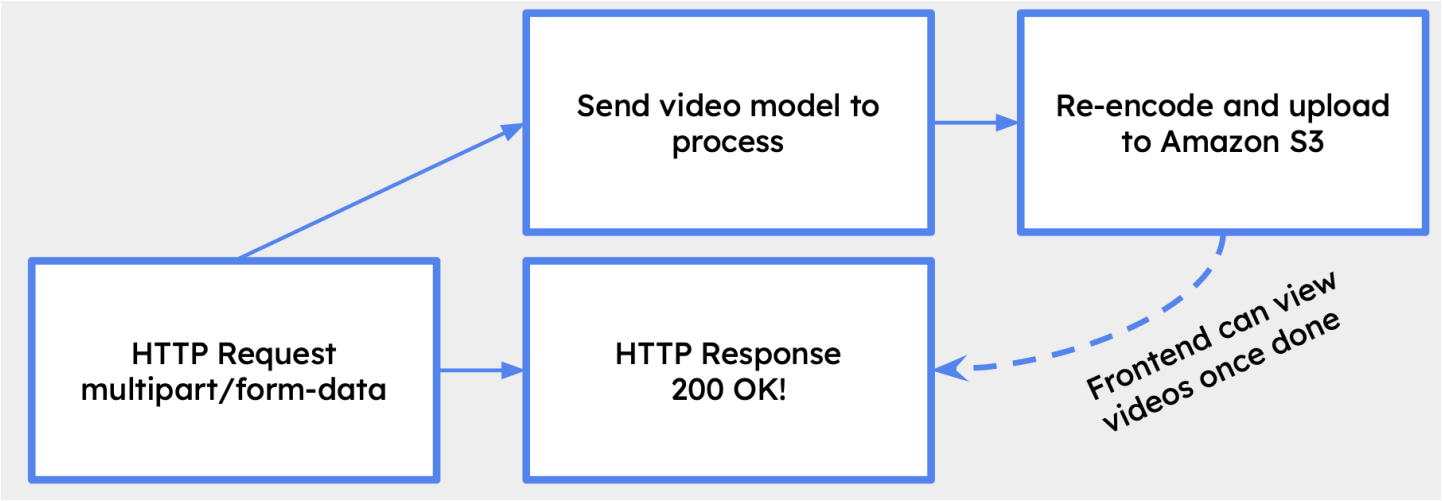

Figure 5, Threaded Processing Model

Our database consists of three main models, as seen in Figure 6: Users, Videos, and Roles. Users have standard information, such as a name, email, password, and ID. Videos include references to the swimmer in the video, the coach who coached that swimmer, feedback from the coach, and the date the video was uploaded. Roles a name and different booleans for certain actions that a role can be authorized to do.

[REDACTED]

In terms of our visualization model, we are using transfer learning with a YOLO v5 machine learning/object recognition model. The model was originally trained on the Microsoft COCO dataset (which contains thousands of annotated images) and then we leverage transfer learning to train the model on some of our own annotated swimmer images. The annotations were done in Roboflow and extended the capabilities of the original Microsoft dataset. When the model is run on any given video, it goes through frame-by-frame and generates predictions for each frame (again due to processing and model limits). The model can't make predictions for all frames, so we fill in the gaps using a histogram of oriented gradients (HOG) along with the distance formula. The HOG takes multiple oriented gradients for a given image and stores them numerically in a HOG descriptor; we then use these numerical HOG descriptors on the area around the last predicted frame, and we use the distance formula to determine which subframe contained the most similar descriptor to the previous prediction.

Some visualization challenges we face are the training data quality and having the head out of water. The training data quality is just by the nature of cameras and stabilization underwater, but lots of bubbles and turbulence/chaos underwater really begins to cause problems for our model (see Figure 7). Our client is aware of this and is working to improve videos taken in terms of quality. The swimmer's head moving in and out of the water is another issue that we are finding with our model, but the HOG descriptors are mostly helping here, at the cost of the video looking a bit choppy, but overall it's working well.

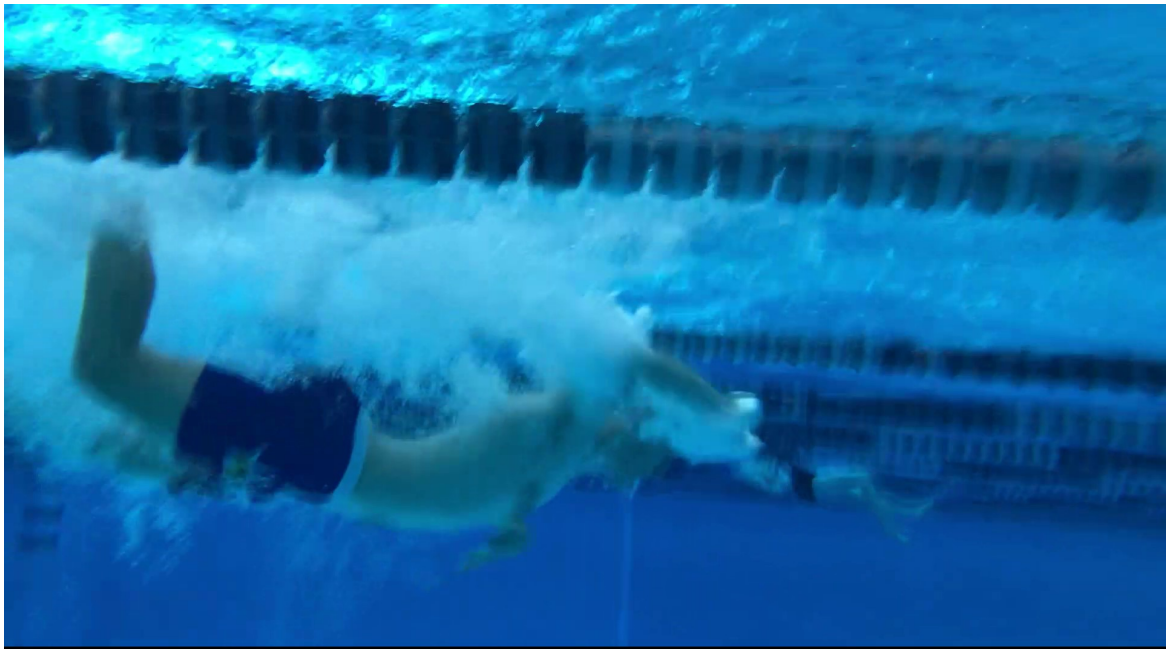

Figure 7, a chaotic image for our model to decipher

Altogether, we can leverage this trained model on Roboflow via API requests from our processing backend to process videos. Once we process the videos in the model, we add overlays with OpenCV and rewrite the frames of the video. The overlays include head and hip detection which adds red squares around the swimmer's head and hips. The overlay also includes the angle of attack, which is the angle at which the swimmer is cutting through the water, denoted by a red line in our overlay. Finally, we add blue curved arrows with lines going around the angle of attack and swimmer to represent the resistance drag flow. Once all of the frames have overlays, we stitch the video back together, re-encode it with ffmpeg, and save it as an image file which is then uploaded to S3 as mentioned.

VIII. Software Test and Quality

Backend Testing: We are testing our database to ensure that the database schema and models were created correctly (i.e., primary key or other constraints are working). We also have unit tests for the backend to ensure the proper flow of logic with certain models, and we have a CI pipeline for running tests on commit. For all unit tests, we will be aiming for ~40% code coverage. Of course, if we have time, we can shoot for higher, but we want to make sure that we complete the majority of the actual application instead of worrying about code coverage with unit tests.

UI Testing: We plan to test the frontend user interface semi-automatically (with actual manual testing) and hopefully automatically (with webdrivers) to ensure that page elements are correctly shown and queried from the backend.

Frontend Logic Testing: Any frontend logic that is testable should be tested on top of automated/semi-automated user interface testing.

Code Reviews: We're taking a pretty generous approach to code reviews, but for crucial features and leading up to our production release, we want at least one other person on the team looking at commits and reviewing code in general.

After the initial implementation of the product, we are beginning to also use branches and pull requests to help alleviate merge conflicts and allow for easier code review.

User Acceptance Testing

- **Training**: The client will be trained to use our product for a day in-person and made aware of any possible implications or difficulties.
- **End user**: We will ask the client to go through the entire video uploading process and watch them from start to finish, ensuring that the product does exactly what they want. It would also be beneficial to talk to one of the swimmers (or watch one of the coaches talk to them) to ensure our automated feedback/detection system is actually beneficial.

Security Testing

- Within the continuous integration, we can run secrets detection as well as other security tools to verify that things in the repository are safe.
- We are still testing our authentication to ensure that the flow works correctly and no user information is at risk of being leaked.

Static Program Analysis: We'll be using a flake8 linter to maintain code standards on our Python + Django backend.

Test Plans

Requirement #1: Web backend

- Test that the backend database queries are returning the correct thing as well as sending the correct data to the frontend via the Django model system.
- Can use Django's builtin test suite.
- Should pass for all runs of the test suite.

Requirement #2: Web frontend

- Test that the user interface elements are being queried from the backend and rendered correctly, and also make sure frontend logic is working properly.
- Use vitest, Selenium web driver and related to automatically/semi-automatically test. Also manually test this.
- The website should properly render a set of test videos on screen as well as properly handle the user authentication and authorization flow.

Requirement #3: Visualization

- Test that the visualization model is producing the correct angles (with some tolerance) for a specific testing video.
- Use pytest or a similar Python library to test.
- Should pass (with tolerance) for all runs of the test suite.

IX. Project Ethical Considerations

Storing user videos of minors (even if they signed a waiver): We want to be respectful and considerate of our users' privacy, even if they're signing waivers to let us use their data.

Encrypting user data (passwords, videos) in our database: We want to ensure that our session authentication is robust, as well as our authorization for each action on each route.

Computer vision and machine learning bias: Current model has only been trained and tested on white females; we are trying to get a hold of more diverse training data to improve our product. Other races or males could have much worse performance with the current model.

X. Project Completion Status

The goal of this project was to create a full stack web application and a computer vision video processing backend, and we reached these goals. The full stack web application should use Django and PostgreSQL on the backend, and Vue3 on the frontend. The computer vision video processing backend was also written in Python using OpenCV, and integrated into our Django webapp. The processing script fully processes the video and saves a copy of the file with visual resistance feedback written to it. On the backend, we implemented database models which store Videos, Users, and Roles. [REDACTED] The client will be able to edit permission levels in the future and swimmers, coaches, and admins will all see different things in the frontend. The frontend includes pages to view all videos, upload a video, administrate users, and view videos per user. Altogether, the coach upload flow is implemented; the coach takes a video, uploads it, it's processed, and then returned for both coaches and swimmers to analyze. The final product will be deployed to DigitalOcean servers for a production environment. In terms of performance testing, it seems to work well (at least better than the existing software). We are still testing but are confident that the processing is working well. The web app also works across the following browsers which we have tested: Safari, Edge, Firefox, and Chrome. We are still planning tests with the client and their coaches and swimmers, but the client has been pleased from a usability standpoint so far.

The following are features listed out of scope with consensus of the whole team and client:

- This project will not be integrated into any of the current Swim Tech Web applications. There is an existing Django and Vue website, but we were specifically asked to simply make the visualization API.
- [REDACTED]
- Stroke processing improvements: The future of this applies in more advanced visualizations and stroke-specific feedback. We already implemented some functionality on the frontend for this, but it needs a lot more work over time.

XI. Future Work

The future work on this product will include focuses on all of the features which were listed out of scope. In general, the knowledge and skills required for this include strong web development skills: Vue, PostgreSQL, HTML, CSS, JS, HTTP, Python, OpenCV and Django. In general, the time required to implement more features should take about 6 months to a year. In specific, the stroke processing improvements (mentioned in section XI) will take the longest as they need further tweaking and model training which takes time and knowledge. The future developer(s) of this application must understand OpenCV thoroughly as well as domain knowledge of swimming. Talks have already begun with the client to extend the work and potentially offer paid work for the completion of out-of-scope tasks.

XII. Lessons Learned

Video processing is tough! Not just with OpenCV, but with the whole flow of saving videos and encoding/decoding. As mentioned in section VII, we use threading to deal with this issue, but that brings its own issues. We now have to manage threads while dealing with an entire web app backend running at the same time.

Keeping everybody on track with skills that they haven't learned yet is difficult. We were all in different places in terms of programming experience, and keeping the project moving forward while people were learning was difficult. We all had to balance learning new things with producing actual code output. Hence, we all were forced to learn how to better manage our time and learning through a project.

Communication is key. Even though we had a technical client, there were a lot of clarifying questions we had to ask and even some misunderstandings that could have been avoided. We all learned how to ask better questions as well as interpret what the stakeholder is asking for.

Authentication is really tough. We implemented session-based authentication using Django Rest Framework (DRF), but due to the decoupled nature of our web backend and frontend, it became much harder than it should've been, and we still had to implement a lot from scratch. As a result of the struggle, we learned a lot more about the nature of HTTP and how the web works as well as authentication tokens and different methods of authentication (token, jwt, etc).

Vue3's composition API was difficult to use at first and an interesting lesson learned in terms of structuring frontend applications. We were originally using the options API, but by request of our client (who knew technical details), we switched to the composition API. This was definitely an interesting paradigm shift for those of us who knew React or other frontend client side frameworks. Overall, we all learned a lot about how to structure content, scripts, and styles in frontend applications.

XIII. Acknowledgments

Thank you to our client, Evan, who was incredibly respectful and involved with us. Evan's technical ability and insights allowed us to achieve even more and work more quickly. He gave us insightful career and life advice in 1-on-1s and even did resume reviews for us. We couldn't have asked for a better client, truly.

Thank you to Dr. Nils Tilton who gave valuable advice on fluid modeling and computational fluid dynamics, which dictated the majority of our future work with the modeling/visualization aspect of the project.

Thank you to our technical advisor Lorenzo, who gave valuable advice on Rest APIs and the general product architecture of our project as well as taught us about web architecture and CI/CD.

XIV. Team Profile

Ethan Richards
Junior in Computer Science
Hometown: Centennial, Colorado
Work Experience: Experienced in full stack web development and a variety of programming languages and skills.
Extracurriculars: President of Mines ACM & HSPC Chair

I'm excited to work on Swim Tech to create a useful product (with a clever technological solution) that can be used to better help and teach people swimming!

Grant Bohlmann
Senior in Computer Science
Hometown: Houston, Texas
Experience: Experience with computer vision, graphics, and software development.
Extracurriculars: Club Volleyball

I'm looking forward to gaining some software engineering work experience working on a large project with a great team.

Morris Gray
Senior in Computer Science
Hometown: Lafayette, California
Experience: Experience with machine learning, computer vision, and graphics
Extracurriculars: Desk Assistant

I'm excited to gain experience working on a real, big project as well as working with a small team.

Mason Shandy
Junior in Computer Science
Hometown: Colorado Springs, Colorado
Work Experience: Published author of research paper, experience with backend web development
Extracurriculars: Fellowship of Christian Athletes Leader, club soccer

I am excited to learn and experience what working with a real project and team is like.

Appendix A – Key Terms

Term	Definition
OpenCV	*“Open Computer Vision,” a computer vision library*
Django	*A backend Python web framework*
Vue/Vue3	*A frontend JavaScript web framework*