***EE/CprE/SE 491 SEMESTER 2 WEEKLY REPORT 3***

***2/17/2025 – 2/23/2025***

***Group number: 11***

***Project title: Slowpitch Softball Pitch Detector***

***Client &/Advisor: Nick Fila***

***Team Members/Role:***

***Andrew Vick - Machine Learning Integration***

***Casey Gehling - Client Interaction***

***Sullivan Fair - Individual Component Development***

***Ethan Gruening - Team Organization***

***Josh Hyde - Research***

***Cameron Mesman - Testing***

o **Weekly Summary**

* Over the last week (2/14-2/20), the team fully integrated the OpenCV code into iOS. We do not have an implementation of the tracking modules for Android, but we are currently working on getting a framework built for it. The team also researched multi-threading so the YOLO model can run separately from the KCF tracking. This should make our app more efficient and improve the runtime. Lastly, we developed a testing plan to collect the relevant data to improve our project. From here, we plan on continuing to refine our tracking code, improve the user experience of our app, and perform field tests.
* This week (2/21 - 2/27), we began unit testing the Flutter computations and the computer vision (YOLO/OpenCV) for determining height. We began the week determining pairings to split up and work on the modules of the application before another comprehensive application test. Andrew and Sully, using the Ultralytics Flutter library, configured a Flutter program to allow a YOLO model to run on both iOS and Android’s machine learning-specific hardware to decrease latency for object detection. Casey and Ethan worked on unit testing the height calculation when given the ball’s coordinates. Removing a constraint, the setup process now requires one photo and simplifies the complexity of the height equations.

o **Past week accomplishments**

* **Andrew Vick:**
	+ Testing at Beyer Hall
		- Discovered that Yolo was not properly finding the ball, the coordinates it was finding did not line up with the coordinates system for Flutter’s camera preview
		- Yolo was also causing the video to be incredibly laggy. We discovered that we could improve performance by utilizing Apple’s ML framework CoreML for mobile devices. I started working on converting our app to use this framework, which has involved changing how we handle images and how yolo is implemented.
	+ I paired with Sully this week to get YOLO integrated into our app in a manner that would allow for real-time object detection. We accomplished this using an API created by Ultralysitcs specifically for use within a FLutter mobile application. The model also ran surprisingly well. In the past, when we were trying to run YOLO outside of the Flutter pipeline, we would be lucky to get 5 fps. By using the API, we were able to get 30fps without any noticeable lag or latency consistently.
	+ This week, Sully and I also looked into some different quality-of-life improvements we could make to the API so that it's a bit easier to use. One of the main issues we were having with it was when you orientate your phone from portrait to landscape. The video preview on the screen gets messed up, and half of the preview is off the screen. Sully and I were able to find the cause of this bug and fix it however, since the bug was within the API itself, we will need to create a GitHub issue for it and see if we can include our fix.
	+ TLDR
		- Got YOLO running efficiently on our app
		- Fixed some bugs within the API we were using to allow for seamless integration into the rest of our app.
* **Sullivan Fair:**
	+ Testing at Beyer Hall
		- Setup was convenient and simple to set up, but may differ when we perform user tests
		- Found that YOLO was not properly finding the ball
			* Determined that there may be issues with how the image is being formatted between Flutter and OpenCV
		- The video feed was still very laggy
		- Decided to introduce CoreML, which provides an easy way to implement machine learning into our app on the iOS side
	+ I paired with Andrew and successfully integrated a YOLO model into our app. We accomplished this using the Ultralytics API for Flutter and YOLO. The YOLO model runs at a consistent 30fps without any noticeable lag. This means we can keep all our code on the Flutter side and forgo any external C++ code.
	+ I also collected some videos of pitches using to be used to train our own custom YOLO model on softballs. The current code utilizes an existing YOLOv8n model that detects much more than we need. Training our own model lets us have more control and will provide better and more consistent results.
	+ I began the process of training our own custom model using our Roboflow dataset created by Ethan. Currently, there are some issues the model continuing to define objects that are not softballs. I believe this issue stems from pre-trained data on the yolov8n model being carried over to the custom model.
	+ TLDR
		- Tested at Beyer Hall
		- Got YOLO to run on our app
		- Collected videos to train a custom YOLO model
		- Custom model training
* **Casey Gehling:**
	+ Testing at Beyer Hall
	+ Continued research on the accelerate framework for iOS and potential acceleration strategies for Android.
	+ Researched existing OpenCV solutions in an attempt to mitigate our current pipeline issues. Working towards introducing some optimizations into our functionality.
	+ Paired with Ethan on researching Flutter image passing as well as streamlining our setup and height calculation processes.
		- Found that Dart passes values by reference, negating the need to pass information with pointers on the front end. This pass by reference behavior is automatically accounted for in our backend.
		- Relaxed a constraint with our image setup process in that we require the user to align the home plate and pitcher's mound. This allows for a simpler setup in which we only need to acquire a reference height as well as a plate position.
		- Tested this new calculation process through a visual interface made available through the camera screen.
	+ Attempted to create a YOLO-8n model from our existing Roboflow image dataset paired with images from the COCO dataset, but ran into storage limitations on my own computer that prevented me from fully creating the model. If anything, this was an exercise in learning the process of creating a YOLO model.
* **Ethan Gruening**
	+ Testing at Beyer Hall
		- Testing the product, we saw that the setup guide took 1 minute to complete (not including reading time). This is a very efficient and simplistic setup.
		- Issues with the Android testing were caused by the OpenCV tracking modules not being included in the standard Android SDK release of OpenCV.
	+ **All changes made below were pushed to a new branch, ‘improve-latency’**
	+ Decreasing OpenCV Latency
		- Since OpenCV automatically parallelizes many of its operations, I set the number of threads to the number of CPUs available to OpenCV. This will allow for parallel executions and lower latency.
		- Flutter’s CameraController’s fpm is set to perform at 120 fps for supported devices. I reduced the number of frames to process per second, skipping two frames for every frame sent to the backend.
		- To reduce the lagging for OpenCV calls in the backend, I made a Flutter Isolate, a separate thread for UI management that calls and waits for the C++ functions. This allows the screen to update smoothly, and values will be refreshed when the backend detects the ball.
	+ Building OpenCV tracking module
		- Creating an environment in Windows Subsystem for Linux and installing CMake, ninja, Google in-line web commands, Android SDK, Android NDK, and the OpenCV release repo and OpenCV\_contrib repo. I then executed the building script for a new OpenCV SDK with the tracking modules included.
		- I replaced the old Android SDK with the new one.
	+ New Flutter Looks
		- New visuals have been added for camera placement and reference height instructions.
	+ Flutter Image Transfer
		- Pairing with Casey, we researched how Flutter handles the passing of an image through several different functions and classes. Found that Flutter dynamically passes values by reference, allowing low latency for image transfer between threads.
	+ Unit Testing (Flutter)
		- Working with Casey, we worked on a new branch to unit test our height calculations given the location of the ball.
		- Disabling the C++ function calls and giving sample coordinates, we found our height calculation was giving incorrect results. Reworking the geometric calculations, there began to be complex equations using point-slope, slope-intercept, and triangle similarity to accommodate for the home plate and pitcher mound not being horizontally aligned.
		- Removing a constraint, we require our users to level the pitcher's mound and home plate, removing many setup steps and simplifying the height calculations to a simple pixel-to-distance conversion.
		- The calculation has been modified, unneeded setup steps are removed, and the Flutter side is ready for additional unit testing within the **self-identifying-test-height** branch.
* **Josh Hyde**
	+ Helped with the testing at Beyer Hall and helped to discover the ease of setup for most of our current design, but also the model not wanting to work correctly for the most part
	+ I also worked on improving the orientation issues I had before with the camera showing different orientations, such as being upside down when it should be right side up. One problem I found with this is that the people who have tested our app on an iPhone have found that the orientations are correct. They aren’t upside down or in weird directions, which may mean that either my phone is different or all androids may cause orientation issues that we may have to look into. However, In the current branch I am working in, I have “fixed” the orientation issues, at least for Android.
	+ I also worked to improve the max and min-height bounding lines. I also realized the reason that the 3rd line was showing up and that is now gone, however, I have some reason to believe that the lines are still slightly off as I slightly misunderstood the exact value that I was receiving from the other class that someone else made.
	+ Developed accurate height detection lines
		- I’ve been working on trying to get max and min lines to visually show up on the screen to signify where the max and min lines are when using the app. This is beneficial for several reasons, allowing for better testing and usability for our users. I finally got it to be about as accurate as it could be. However, it is based on the user's inputs on the previous screen that will allow them to click the home plate, the pitcher's mound, and the reference height above them. This could cause some issues as it is more based on the user’s own inputs and accuracy, which might be prone to errors.
		- Because the painter that was used to paint the screen with the lines uses a different coordinate system, I had to apply a scale factor of the painter to the phone for it to properly line up. This could be prone to future errors if the scale factor is different for every phone, but it seems to work properly for now.
	+ Worked on Android orientation issues
		- Although I could not fully come up with a solution, I spent a lot of time trying to think of different ways to fix the problem of my Android phone having the wrong orientation when on a camera screen. One temporary solution I found works. However, it compresses the image slightly, which in the long run isn’t very effective, so I have been seeking alternative methods that all seem to have their own errors.
	+ Grid System
		- The idea of a grid system was thought of, which would implement a grid coordinate system to tell the heights better and for testing purposes. Very minimal work was done as Cameron and I’s schedules didn’t really line up this week, so we will be doing more implementation of this next week.
* **Cameron Mesman**
	+ This week I researched ways to use the android ML hardware for better performance running our app. I did find some good info, however Andrew and Sully’s findings with the Ultralytics API might remove the need for this. If we can get good enough performance with just the API, the need for a specific model that uses the android ML hardware is nullified.
	+ Also started pairing with Josh on a grid system for the camera that will help us define screen dimensions. We got an initial grid on the screen, but will continue to develop the implementation next week.
* **Pending issues**
	+ YOLO through the Ultralytics functions still need to be integrated into Flutter’s tracking thread. A finer-tuned model can also help YOLO’s accuracy in determining only softballs within the frame.
	+ The height calculations when self-identifying the ball still needs to be unit tested, and there is currently an error with iOS setup.
	+ Grid layout still needs to implemented.

o **Individual contributions**

| **NAME**  | **Individual Contributions** *(Quick list of contributions. This should be short.)* | **Hours this** **week** | **HOURS** **cumulative** |
| --- | --- | --- | --- |
| Andrew Vick  | Testing, research and implementation of CoreML, Flutter/YOLO integration, bug fixing | 8 | 82 |
| Casey Gehling | Testing, iOS and Android acceleration research , Height calculation, unit testing | 7 | 82 |
| Sullivan Fair | Testing, research into CoreML, Flutter/YOLO integration, Filmed videos for model training, Model training | 7 | 90 |
| Josh Hyde | Height max and min lines, android orientation issues, grid system | 6 | 83 |
| Ethan Gruening | Lowering latency, adding setup images, building SDK, testing, Rewrote the height calculation and researching Flutter’s argument handling. | 7 | 109 |
| Cameron Mesman | Android ML hardware, camera grid system | 5 | 80 |

**Plans for the upcoming week**

* Andrew Vick
* Casey Gehling
	+ Continue testing
	+ Bug fixing
	+ UI changes
* Ethan Gruening
* Josh hyde
	+ Grid implementation
	+ Review pitches screen fixes
* Sullivan Fair
	+ Continue to refine the custom model
	+ Once the model is complete, figure out how to handle the returned data
* Cameron Mesman
	+ Grid implementation with Josh