Slowpitch Softball Pitch Detector

Design Document

sdmay25-11

Nick Fila

Dr. Jones

Andrew Vick (Machine Learning Integration)

Casey Gehling (Client Interaction)

Sullivan Fair (Individual Component Development)

Ethan Gruening (Team Organization)

Josh Hyde (Research)

Cameron Mesman (Testing)

Team Email

https://sdmay25-11.sd.ece.iastate.edu/

Executive Summary

Our senior design project focuses on developing a system to assist umpires in detecting illegal pitches in slow-pitch softball games. The problem arises from the challenge of accurately determining whether a pitch falls outside the legal bounds of 10 to 12 feet in height, leading to inconsistent calls and disputes. Our solution aims to eliminate this ambiguity, allowing umpires to focus on other aspects of the game while ensuring fairness and accuracy.

To address this issue, we are designing a mobile application that leverages computer vision and machine learning to analyze softball pitches in real-time. This approach removes the need for leagues to invest in expensive or specialized hardware, making our solution cost-effective and accessible to a wide range of users. The app uses YOLO (You Only Look Once) for object detection and tracking, enabling it to identify the softball and monitor its trajectory throughout the pitch.

The key design requirements include accuracy in determining pitch legality, minimal interference with the flow of the game, and affordability for users. Feedback from current umpires has validated these priorities, and our design aligns with their expectations.

Thus far, we have implemented the YOLO model to run within our Flutter application using a plugin developed by the Ultralytics team. Due to the plugins limited functionality though we had to alter it to better fit our needs. Additionally, we have developed calibration functionality that allows the user to define key reference points—such as the pitcher's mound and home plate—and input the camera's height. This calibration maps the physical space into a frame of reference, enabling precise height calculations for the pitch and determining whether it is legal.

Our design meets the defined requirements by ensuring the app runs smoothly on readily available iOS devices, avoiding the need for additional hardware. Accuracy is achieved through rigorous training of a YOLO model and strict setup instructions. The next steps include adding additional features to our app outside of the main functionality, like a past pitches screen.

By enabling umpires to confidently identify illegal pitches, our product aims to improve the quality and fairness of games while maintaining accessibility and usability for all levels of softball leagues.

Learning Summary

Summary of requirements:

- Object detection system to locate a softball at its maximum height during a pitch.
- Detect an illegal pitch when a softball's maximum height is higher than the specific maximum height or below the specific minimum height.
- Create an audible sound indicating an illegal pitch.
- The device cannot be physically obstructive to the game.
- The device must accommodate different fields, lighting, and balls.
- The device cannot be visually distracting to the game.
- The device must have a guided and simple setup.
- The device must have user-adjustable settings for the maximum and minimum height for pitches.

New skills/knowledge acquired that was not taught in courses:

- Computer Vision
- Machine learning applications
- Flutter framework
- iOS development

Table of Contents

1. Introduction

1.1. Problem Statement

In the game of softball, one crucial rule states that a pitch is only legal if the ball reaches its peak height within a minimum and maximum height bound during its flight to the batter. Currently, umpires must estimate this height by eye for every single pitch: a challenging task that relies heavily on personal judgment and can vary from one umpire to another. This reliance on subjective estimation can lead to inconsistent calls, potentially altering the outcome of games and affecting the fairness and enjoyment of the sport. Players might strike out on pitches that should have been deemed illegal, and fans may witness games swayed by questionable decisions.

These issues highlight a broader problem: the lack of accessible technology to assist in making precise, real-time measurements during games. To address this, we are developing an affordable and user-friendly device that tracks each pitch using a single camera, accurately measures the ball's height using machine learning, and alerts officials when a pitch falls outside the legal bounds. By eliminating the guesswork from pitch height estimation, we aim to enhance the accuracy of calls, uphold the integrity of the game, and improve the experience for players, umpires, and fans alike.

1.2. Intended Users

Our product is designed to benefit several key groups within the softball community, including umpires, players, fans, and coaches.

1. **Umpires**
    a. **Description**: Umpires are the officials responsible for enforcing the rules of softball during games. They must make quick, accurate decisions under the pressure of real-time play, often without technological assistance.
    b. **Needs**: Umpires need a reliable method to accurately determine if a pitch meets the legal height requirements, reducing the reliance on subjective judgment and minimizing errors that could impact the game's outcome.
    c. **Benefits**: Our device provides umpires with precise, real-time measurements of each pitch's height. This technology aids them in making consistent and accurate calls, reducing stress and enhancing their confidence in officiating. By supporting umpires with accurate data, we help maintain the game's fairness and integrity, directly addressing the issues outlined in our problem statement.
2. **Players**
    a. **Description**: All players in a softball game are directly affected by the calls made by umpires and strive for a fair and competitive environment to showcase their skills.
    b. **Needs**: Players need assurance that the rules are being applied consistently so that their performance is judged fairly. Batters, in particular, need protection from illegal pitches that could unfairly result in strikeouts.
    c. **Benefits**: With our device ensuring accurate detection of illegal pitches, players can trust that the game is being officiated fairly. Batters are less likely to be unfairly penalized, and pitchers receive clear feedback on the legality of their pitches. This fosters a fair playing field and allows players to focus on their performance, enhancing the game's overall quality in line with our problem-solving goals.

3. **Fans**
    a. **Description**: Fans are the spectators who enjoy watching softball games, whether in person at the stadium or through broadcasts. They are passionate about the sport and value exciting, fair competition.
    b. **Needs**: Fans desire an enjoyable viewing experience where player skill rather than officiating errors determine the game's outcome. They appreciate transparency and fairness in how the game is played and called.
    c. **Benefits**: By improving the accuracy of pitch legality calls, our device enhances the fairness and excitement of the game, leading to a more satisfying experience for fans. They can enjoy the sport knowing that technology is helping to uphold its integrity, which aligns with our aim to improve the overall enjoyment of softball.
4. **Coaches and Teams**
    a. **Description**: Coaches and team staff are responsible for training players and developing game strategies. They work closely with players to improve skills and ensure compliance with the rules.
    b. **Needs**: Coaches need effective tools to train pitchers on delivering legal pitches and to adjust strategies based on accurate information about gameplay.
    c. **Benefits**: Our device can be used during practices to provide immediate feedback to pitchers on the height of their pitches, aiding in skill development and rule compliance. During games, accurate officiating supported by our product allows coaches to focus on strategy without worrying about inconsistent calls. This directly enhances player performance and upholds fair competition, as highlighted in our problem statement.

By serving these users, our product addresses the critical issue of subjective pitch height estimation in softball. It promotes fair play, enhances the accuracy of officiating, and improves the overall experience for everyone involved in the sport. Our solution connects directly to our overarching goal of removing guesswork from the game, ensuring that softball is enjoyable, fair, and competitive for all participants

2. Requirements, Constraints, And Standards

2.1. Requirements & Constraints

Functional Requirements

- Object detection system to locate a softball at its maximum height during a pitch.
- Detect an illegal pitch when a softball's maximum height is higher than the specific maximum height or below the specific minimum height. (constraint)
- Create an audible sound indicating an illegal pitch

Physical Requirements

- The device cannot be physically obstructive to the game.
- The device must be portable to set up on a softball field.

Environmental Requirements

- The device must accommodate different fields, lighting, and balls.
- The device cannot be visually or audibly distracting to the game.

UI Requirements

- The device must have a guided and simple setup.
- The device must have user-adjustable settings for the maximum and minimum height for pitches. (constraint)

Constraints

- Maximum height (in feet) the softball can reach on a serve.
- Minimum height (in feet) the softball must reach on a serve.
- The device cannot be physically obstructive to the game.

2.2. Engineering Standards

The Importance of Engineering Standards (Q1)

Engineering standards are important because people interact with engineering products every day. If there were no engineering standards, that would not only compromise the desired quality of engineering products but also affect things like the safety of everyday people. For example, people interact with civil engineering like bridges and buildings, and non-physical structures like apps and programs. A lack of requirements for these products could result in catastrophic failures of civil structures or a breach of confidential personal data from an insecure application. Engineering standards help ensure that these types of issues are avoided so people can continue to use these products to improve their lives and the lives of others

Published Engineering Standards (Q2)

- **IEEE 1857.9-2022** provides standards for video encoding/decoding and analyzing methods. These standards apply to all forms of video manipulation, spanning from areas such as network video transmission over services such as UDP to computer vision topics such as object detection.
- **IEEE P3110** provides the standards for the necessary API requirements in a computer vision implementation. These API abstractions interface with various machine learning algorithms and are used to develop computer vision solutions such as OpenCV.
- **IEEE 1008-1987** provides guidelines for proper unit testing with a codebase. Specifically, guidelines exist for proper categories of testing (unit, integration, etc.) and code coverage reporting, among other testing-related requirements.

Engineering Standards Relevancy (Q3)

After reviewing the three standards, each has varying relevance to our project. Since our primary focus is on computer vision, IEEE P3110 is highly applicable. This standard provides guidelines for API requirements in computer vision and will help us ensure our project adheres to best practices when working with machine learning models like YOLO. It offers a strong foundation for structuring our computer vision solution. IEEE 1008-1987 also holds some relevance, as it provides useful guidelines for software testing. While our project's smaller scale, incorporating structured unit testing can still improve code reliability. On the other hand, IEEE 1857.9-2022 is less applicable since it focuses more on video encoding and compression, which isn't central to our object detection work.

Other Applicable Standards (Q4)

Some of the standards we found that differed from those provided that might be at least somewhat applicable to our project are below. IEEE Standard Digital Interface for Programmable Instrumentation. This standard applies because we use equipment and information that takes in different measurements. We will use different types of instrumentation that may be programmable to detect the softball. IEEE Standard for Application Programming Interfaces (APIs) for Deep Learning (DL) Inference Engines. This engineering standard is applicable because we are programming a solution to consistently and accurately find the height of a softball, including deep learning and AI-based detection algorithms. IEEE Standard Letter Symbols for Units of Measurement (SI Customary Inch-Pound Units, and Certain Other Units). This standard is

applicable because it ensures that we incorporate the appropriate symbols for our measurement and that it is understood clearly using accurate measurements and symbols.

Modifying Our Project to Incorporate Engineering Standards (Q5)

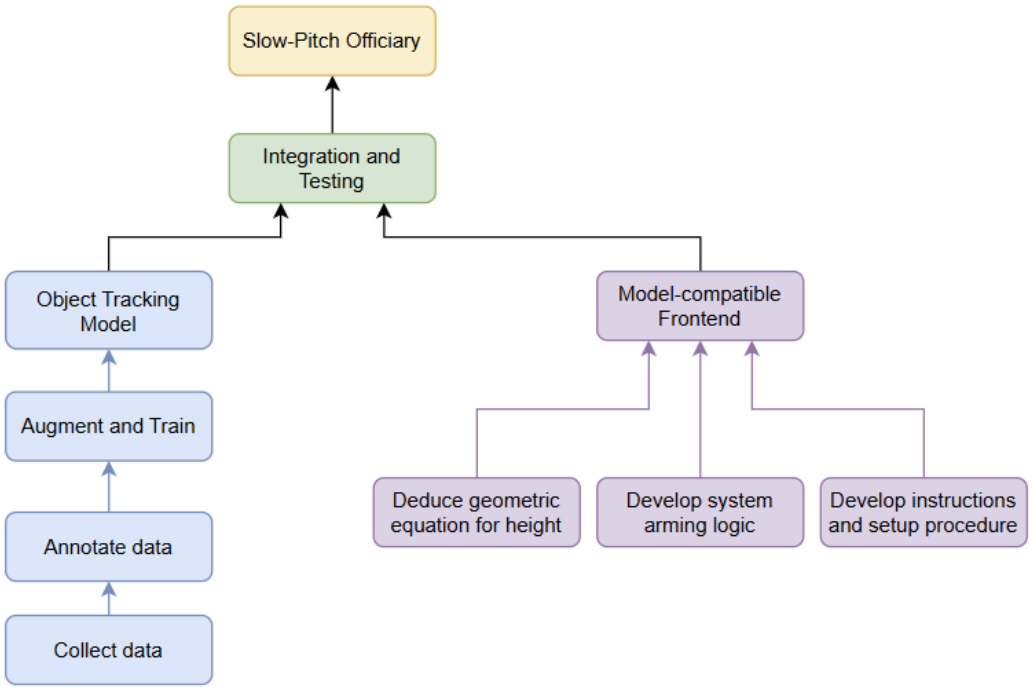
Since our design is not finalized as we continue to test and prototype different designs, we don't need to make any modifications now. However, several of these standards must be considered as we proceed with our design. Firstly, as we develop the software component of the project, we will need to keep in mind IEEE 1008-1978 regarding unit testing. We need to ensure that we are writing the software in a way that can be easily tested. We will also need to consider IEEE P3110 which covers standards with computer vision. Since many of us have never built an official project using computer vision, it is not natural to consider these standards. Moving forward, standards like this one must be considered during design

3 Project Plan

3.1 Project Management/Tracking Procedures

We decided to adopt an AGILE methodology to manage our project. There are complex components we need to manage with our app. These components include our height detection script utilizing a machine learning approach and a mobile app utilizing Flutter and the Ultralytics Flutter plugin. To make development more manageable, we can divide the work into AGILE sprints with smaller goals, so we are not trying to tackle the entire project at once. This will improve our understanding of individual project components and help make a full implementation easier. To keep track of our progress, we are utilizing Git issues coupled with personal branches for individual development. Having an issues board will help us manage what tasks must be done. Using personal branches will help isolate development so we can develop individual components more efficiently without accidentally breaking the project's main branch or having more than one person alter the same file in different ways.

3.2 Task Decomposition

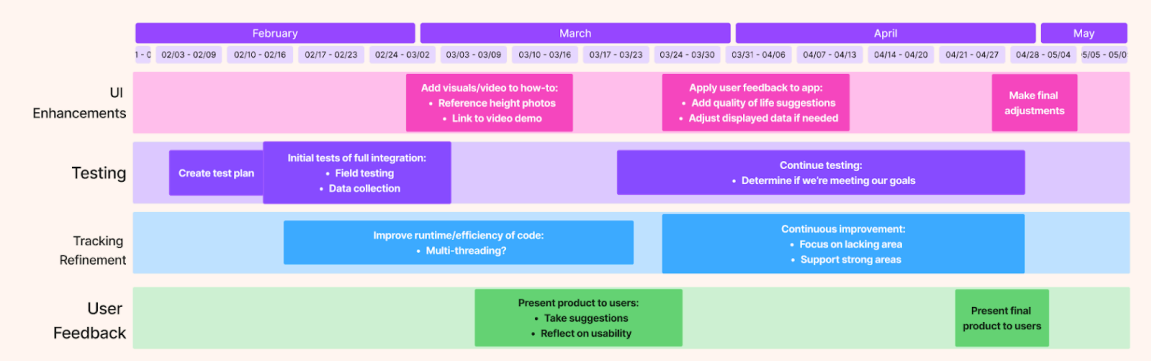


These tasks are intentionally left broad since we have not made final decisions on the entire design. Based on the AGILE methodology, many of these tasks will be adjusted as development continues and we receive feedback from the client. Any testing and prototyping would also affect our final implementation of these tasks. With this task decomposition, we split the project into two major tasks: detecting illegal pitches and developing the mobile application. These two tasks are the most significant challenges and are further broken into subtasks. In creating a model-compatible frontend we needed height calculation, pitch detection, and setup techniques. In developing an object tracking model we must collect, annotate, and augment data for training. Integration of the model and frontend would be completed for a fully functional application.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

In an AGILE development process, these milestones can be refined with successive iterations/sprints (perhaps a subset of your requirements applicable to those sprint).

3.4 Project Timeline/Schedule



Above is our proposed AGILE project timeline, featuring a clearly outlined sequential list of milestones. Our project has been separated into four parallel-running distinct categories: UI enhancements, testing, (object) tracking refinemenst, and collecting user feedback . Following an AGILE methodology, we have separated our project timeline into roughly one-week sprints where certain tasks are scheduled for completion. Each bar on the above Gantt chart represents a deliverable within an AGILE timeframe set to expire at the bar's end. Holistically, the above Gantt chart can be summarized in sprints as follows:

Timeframe/Deliverables

02/03 - 02/09 (Sprint 1)

- Create test plan
- Develop prototype plans

02/10- 02/16 (Sprint 2)

- Initial tests

02/17 - 02/23 (Sprint 3)

- Improve runtime
- Improve functionality

02/23 -03/02 (Sprint 4)

- Add Visuals
- Add Videos
- Start trying to combine app with model

03/03 -03/09 (Sprint 5)

- Test product with users
- Keep testing

03/10 -03/16 (Sprint 6)

- Continue Testing

- Fix bugs and improve optimality

03/17 - 03/23 (Sprint 7)

- Continue Testing
- Make sure tests are going smoothly

03/24 - 03/30 (Sprint 8)

- Apply feedback to app

03/31 - 04/06 (Sprint 9)

- Continued improvement
- Fix issues and or areas that need improvement

04/07 - 04/13 (Sprint 10)

- Test more with added improvements
- Try and find and then fix any issues

04/14 - 04/20 (Sprint 11)

- Add final functionalities of the app that will be used

04/21 - 04/27 (Sprint 12)

- Present final product to users for testing
- Make final adjustments and improvements of the app/code

04/28 - 05/04 (Sprint 13)

- Finish the full working app
- Complete all final deliverables

3.5 Risks and Risk Management/Mitigation

Sprint 1:

- Risks:
    - Our current implementation of the opencv C++ backend with flutter isn't ideal, as the implementation and usage of both of them currently are separate and need to be implemented together if we want to make any sort of progress.
- Mitigation:
    - Try and merge our flutter project app development with our new C++ backend of the opencv implementation

Sprint 2:

- Risks:

        - The main issue is that running yolo on the phone without utilizing Apple's ML framework causes significant frame hangs and stutters
        - We are also having difficulty translating the frame coordinates for the raw image data to the coordinate system flutter is using.
- Mitigation:
    - Potentially implement the Ultralytics plugin model that can better run the yolo model much faster and with less framerate loss. Could also run faster than before as well.

Sprint 3:

- Risks:
    - 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 be implemented.
- Mitigation:
    - Implement the grid layout as quickly as possible, allow the integration of YOLO through Ultralytics in flutter's tracking thread. Test and ensure the accuracy of the height calculation

Sprint 4:

- Risks:
    - Ultralytics still has some issues when implementing this and the overall implementation of all of these parts still needs to be studied and merged together.
- Mitigation:
    - Continue debugging the issues with Ultralytics YOLO and make sure that we start merging the other aspects together.

Sprint 5:

- Risks:
    - Currently, all of our components are separate
    - We need to get a central screen that meshes everything we have worked on to beg to not have a central screen yet that we can test our current info and analysis on yet.
    - Still haven't really tested anything yet with actual on field tests
- Mitigation:
    - We need to get a central screen that meshes everything we have worked on to begin doing real live tests.

Sprint 6:

- Risks:

        - Current roboflow model that we implemented from last semester isn't entirely accurate, however it was tested on different angles and types of videos that aren't very similar to what we are going to test on.
- Mitigation:
    - Implement a new roboflow model that is trained mainly if not completely on the actual camera angle and general conditions that we will be using the app on, so a side camera angle on a field with the ball being thrown in the air across the screen.

Sprint 7:

- Risks:
    - The Roboflow annotations still need to be completed. This takes time, and a new model can be built once finished.
    - The Ultralytics plugin requires portrait orientation, while our application requires landscape mode. This causes an error in the sizing and orientation of the camera preview.
    - The Ultralytics plugin currently does not update the ball coordinates variable.
- Mitigation:
    - Try to annotate the new roboflow model as quickly as possible, and try and fix the orientation issues of Ultralytics forcing portrait, by potentially changing the rest of the app to portrait orientation. Try and fix the issue of plugin not updating the coordinates or not

Sprint 8:

- Risks:
    - The Roboflow annotations are still being worked on
    - The Ultralytics plugin requires a TensorFlow Lite model for Android.
    - The Ultralytics plugin currently does not update the ball coordinates variable.
- Mitigation:
    - Finish annotating the roboflow new model images, as well as try and implement the new tensorflow Lite model for the ability to run our roboflow model on android as well.

Sprint 9:

- Risks:
    - The orientation fixes for IOS changed the Android orientation to be off now, so we might need more platform specific orientation changes.
    - Further testing and enhancing of our current model
    - Further work and implementation of pitching box and detecting when a pitch is being thrown and not just any ball in frame.
- Mitigation:
    - Potentially fix platform specific orientation changes or switch to only one platform. Allow the pitching box to have an arming system that will arm when ball is detected for a certain amount of time within the pitching box.

Sprint 10:

- Risks:
    - Our conversion of YOLO coordinates to standard image size is not functioning. We need a new way of mapping our coordinates used in height calculations.
    - Full functionality is present within individual components and need to be integrated into the full system.
- Mitigation:
    - Switch coordinate system and calculations to only one form of coordinates, preferably just use the YOLO coordinate system for everything to standardize it. Start working on full integration and test to make sure everything works together well.

Sprint 11:

- Risks:
    - YOLO red ball marker
        - When the Yolo model is executed using the iOS CoreML framework, the ball's x and y coordinates are automatically captured as values between [0, 1]. Currently, the Ultralytics plugin converts these coordinates back to pixels based on an assumed aspect ratio.
    - UI Fixes
        - Overflowing widgets
        - Instructions page is outdated
    - Illegal logic listener
        - When 'tracking' is triggered to True, listener must collect and analyze the ball coordinates throughout the pitch until a max height is reached.
        - Sound should be played with the max height found is out of bounds.
- Mitigation:
    - We could pass the x and y values directly to our screen, which calculates height, and then scale the ball's position to fit our screen dimensions.
    - Update instructions page, try and fix overflow.
    - Implement the illegal sound audio to be played at the correct time.

Sprint 12:

- Risks:
    - The presentation and design document are not yet complete.
    - Do the finishing touches to our project, specifically small UI changes and making sure that the app runs smoothly and as accurately as possible.
- Mitigation:
    - Finish up working on the presentation and design document as early as possible.
    - Fix and update the final touches on our project.

3.6 Personnel Effort Requirements

Team Member	Task Descriptions	Hours Worked
Sullivan Fair	YOLO Model development, Helped integrate Ultralytics-Flutter plugin, Gathered pitching data, Creating arming system for tracking, Refined Flutter screens, Performed user testin and gathered feedback	129
Casey Gehling	Flutter screen development, C++/Flutter integration research, codebase hygiene maintenance, field research, video caching implementation, client interaction/team interaction	107
Ethan Gruening	Prototype height calculation techniques, collect pitching data and videos, continuously developed and improved all Flutter screens (instructions, setup, tracking), field test for development, conducted user and model testing, weekly update presentations, and developed team website.	170
Josh Hyde	Prototype object detection, research on different models and potential solutions, testing videos, Flutter screen work, illegal height lines, grid lines, android implementation work,	124
Cameron Mesman	Flutter screen mock-ups, integrating C++ code into flutter, integrating illegal pitch audio into flutter code, field testing, unit testing	114
Andrew Vick	Prototype object detection code, translate python scripts into C++, prototype detection and tracking code running on iPhone, Integrate tracking code and object detection	120

3.7 Other Resource Requirements

Developing a mobile application, our team already owns cell phones to test our Flutter applications on iOS operating systems. The primary resources our team has and will continue utilizing are Iowa State's recreational softball fields. Field availability is crucial for the research and testing for proper camera setup, object detection, and in-game functionality. Field reservations will be made as needed for future testing.

4 Design

4.1 Design Context

4.1.1 Broader Context

Area	Description	Examples
Public health, safety, and welfare	Our project is a bystanding officiary to an existing softball game, an environment with a high risk for player injury. Our design must be non-intrusive and not be in the way of gameplay. There should be little to no contact between the game activity and our physical device to ensure public safety.	Considering public safety, our setup is a mobile device placed on a mount along the fence. This ensures minimal physical interaction between the players and the device, limiting additional risk for the players.
Global, cultural, and social	Slow-pitch softball is a variation of a typical softball game for more novice players. Local leagues and amateur slow-pitch players are our target communities and it is our mission to ensure this device is catered to their financial and accessible needs.	As engineers, our social and ethical responsibility is to create a design to accommodate local leagues. Our design as a mobile application opens our user base to all people with smartphones. This addresses the low-cost and portable solution that local leagues would require.
Environmental	Since our project is very niche, there is little to no environmental impact when using our product. However, it is important to recognize the environmental impact of materials and power consumption of the operating devices.	Our product can be downloaded onto a user's smartphone to limit additional electrical components. Additionally, a power supply for the smartphone will need to be provided and may require a battery charger for prolonged extended use.
Economic	Catering to local slow-pitch leagues, we must consider how our product will impact the league's funds for both umpire training and operation.	Our product will be affordable for all users within a slow-pitch league since it requires an existing smartphone. A simple setup ensures that there will be little time training umpires to use the application or set it up before a game.

4.1.2 Prior Work/Solutions

Include relevant background/literature review for the project (cite at least 3 references for literature review in IEEE Format. See link: https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf)

Beginning our discussions on our project design, we researched existing products and solutions for cameras that can track balls within a sports setting. Analyzing the existing products, their user reviews, and their targeted impact helped our team decide what approach would best suit our project.

The first product we researched was the Specialized Imaging Tracker 2, a high-speed fast projectile measurement camera. This camera has three-axis rotation for camera positioning and is programmable for customization. Many users liked the full remote operation, multiple operating modes, tight accuracy error margins, and no need for calibration. However, this is a research-grade camera with a high cost and is very large. Considering our use case, local slow-pitch leagues would appreciate a camera with small error bounds and little to no calibration. Additionally, it's essential for our users to have an affordable and portable option for recreational use. The Specialized Imaging Tracker 2 may be too advanced for the operating scenario and out of the price range for local leagues.

A similar product used in professional sports games is the Veo 3 Sports Cam. This is a 360-degree camera that not only tracks sports balls but also tracks players, displays heat maps, match stats, and tags game actions. This device is known for its object detection accuracy and long battery life. Users can also live stream their sports games through this camera. These features would benefit local softball leagues, though they are not needed for operating as a slow-pitch officiary. The downside to this product is it costs $2,400 and requires a subscription to operate. Used in a professional sense, the Veo 3 Sports Cam is a successful product. Its features may not be needed for the primary operations of a slow-pitch officiary tool; however, the high standard for preserving the integrity of the game with an accurate system gives users the confidence to use the product in game settings.

The Pocket Radar Smart Coach is another product specializing in ball detection in a softball setting. This product is a small device, about the size of a smartphone, and records a clip of a softball pitch and its speed. The device is compatible with a mobile app that saves the video and speed of pitches. With a strong user interface, there are many positive user reviews. Users typically buy this product for the connection to their mobile device, the ability to share and review their pitches, and a cost-effective solution with a portable design. The Pocket Radar targets users who recreationally play and practice softball and baseball. Their isolation of extra sensors and cameras into a handheld device makes it accessible. A drawback to creating a small device is there is low battery life that limits users to a small usage time. The lower-quality camera has a more significant error bound than professional cameras, with a 1 mph error. Although this device has a small battery life and a more significant error bound, it is more tailored for local and recreational users, receiving positive feedback for their specific use cases.

4.1.3 Technical Complexity

Our slow-pitch softball application has several components that integrate together mathematical, engineering, and social challenges. Our engineering challenges can be split into two main modules: the frontend user experience and the height detection. The project's technical complexity would be a large project to design, prototype, and test. Considering this needs to be developed to be

affordable and portable for local leagues, this increases the complexity as more readily available technologies should be used for a wider user base with lower costs.

The height detection component of the project will require using a camera feed and machine-learning object detection techniques and existing libraries in tandem. Additionally, object detection alone will not be enough to determine height; a technique must be found to coordinate the ball's location to a specific height through geometric calculations or calibrated values. With the limited existing products, our team must find our own solution to determining height using engineering principles and practices. Once accomplished, there can be further development to increase the fps or computation time for frame analysis. As an open-ended component of the project, the technical design, prototyping, and testing is a true example of an engineering problem within the real world.

The height detection backend component is the application's "computational brain." Gathering calibration information, collecting camera frames, and setting maximum and minimum heights will need to be incorporated into the system's frontend display. A user would interact with the program in several ways to start, end, and view the officiary logs. It is essential to develop an accessible and guiding frontend for umpires and players to experience a seamless setup and collection process. The compatibility of the frontend and the backend is also very important. This module is a technically advanced project, not only developing a user interface but also stitching it with the backend's operations and reporting its output.

In total, the technical components of this project provide complex engineering problems that match the industry standards as a real-world software project.

4.2 Design Exploration

4.2.1 Design Decisions

A primary design decision was what system we would use for the camera collection and user interface. This decision will define what our users will need to purchase, calibrate, and gain familiarity with when running our application. From a technical standpoint, using multiple cameras would gain more data points to pinpoint the exact height of the ball. However, developing this product specifically for the ease and affordability of local slow-pitch leagues, we decided to implement this application on a singular mobile device. This design decision increases the project's technical complexity but better suits our user base. The application was chosen to be developed using Google's app development platform, Flutter, which can be compiled to run on both Android and iOS, however our development is only focussed and works with IOS. Allowing anyone with a mobile device to install our application invites many users to interact with the widget-based accessible program for IOS. This was our most critical design decision, led by our social and ethical responsibility and our user needs.

Another design decision was the technologies to use in the object detection of the softball within the camera frame. This decision was based on the compatibility with Flutter, the speed of the library's methods, and the accuracy. After careful consideration and research, we decided to utilize the Ultralytics Flutter plugin, which allows us to efficiently run a custom-trained YOLO model for object detection. The plugin provided the best performance compared to hybrid solutions. Using a custom YOLO model for our object detection allows us to have complete control over how we

want to refine the machine learning algorithm to best track softballs in out application. The plugin also offers an easy easy to switch which model we are running, allowing us to test models more efficiently. Overall, this decision balances performance with freedom of development, which complements the flexibility of our implementation.

The final major design decision was the height calculation technique used to take in the position of a softball within a camera frame and output a specific height. After researching, the vertical pixel distance is constant and can be converted to feet when calibrated with known heights and a horizon line (the line from the home plate to the pitcher's mound). This was found by placing a camera equidistant from the home plate and the pitcher's mound, selecting where both are located within the frame and where a known height exists along the horizon line. The image below shows the equal segments representing 2ft.

We now have a technique to analyze the (x,y) of the softball and determine if it lies within the maximum and minimum height bounds set by the user.

4.2.2 Ideation

The design decision we struggled with the most was finding and deciding on the height detection technique. As mentioned in section 4.2.1, we finalized the decision for finding the corresponding ground line below the pitch from home plate to pitcher's mound and identifying a known height to get the vertical pixel-to-feet ratio and turn the detected softball's (x,y) coordinates into a height value. There were other prototyped and researched height calculation techniques that we considered for the final design:

- Multi-Camera Detection
    - Set up three cameras in different locations throughout the field.
    - Run the object detection on all the cameras and determine the ball's distance from each camera with the camera's detected radius.
    - You can use each camera's height to triangulate the exact location of the ball, including height.
    - This violates our design choice for a single camera.
- Machine Learning
    - Given a machine learning model the distance to the softball, calculated by the pixel radius of the detected softball, and the (x,y) coordinates, a machine learning model will guess the height.

    - After many iterations of training the model, we found it inconsistent and widely inaccurate.
    - The search for a geometric calculation technique was needed for fast and accurate return values.
- Known Radius Geometry
    - With the assumption that the radius stays constant for all softballs, we can use the number of pixels of the detected softball to find its distance from the camera
    - Given its (x,y) coordinates, we can determine the angle from the camera and use
- Pitching Line Calibration
    - The home plate and pitcher's mound are identified within the camera frame
    - The pixel-to-distance ratio can be calculated with the known distance between the bases.
    - The pixel-to-distance ratio was inaccurate as the vertical ratio of pixels differs from the horizontal ratio.
    - The ratio found in this method will only determine horizontal distances, not vertical heights.
    - The search for a vertical distance conversion is needed to find the height of a given (x,y) point.
- Known Heights Calibration
    - The home plate, pitcher's mound, and known heights are identified within the camera frame
    - The vertical pixel-to-distance ratio can be calculated from the home plate to the pitcher's mound line and known height's lines.
    - The height can be calculated given the (x,y), getting the number of pixels from the pitching line, and translating it into feet.
    - Used in the final design.

4.2.3 Decision-Making and Trade-Off

Design Option	User Compatibility	Observed Accuracy/Performance	Mathematically Supported
Multi-Camera Detection	**3/10** It would require users to place and have access to multiple cameras running the application	**9/10** Research shows this method is fairly accurate in triangulating a point, but the camera connection will delay the computation.	**10/10** Triangulation is a technique used in satellites for GPS tracking for its accuracy.
Machine Learning	**10/10** No setup would be needed as the machine learning model will handle the field differences.	**4/10** The machine learning algorithm struggled to accurately depict heights when the camera was moved into a new environment.	**5/10** Machine learning is the analysis of data trends, not a direct hard-coded mathematical relation between ball

			position and height.
Known Radius Geometry	**9/10** The height of the camera would need to be inputted by the user.	**3/10** The radius of the ball becomes very small when the camera is placed far away, so the exact distance jumps when the radius increases by one pixel.	**4/10** The angle of the ball to the camera based on its position in the frame varies from camera to camera.
Pitching Line Calibration	**7/10** You would have to select 2 points on the camera feed as a part of the setup.	**8/10** The observed height looked fairly accurate when measuring a few trials in person, but reading larger heights prompted larger errors	**7/10** The horizontal pixel-to-distance conversion ratio is similar to vertical, but not quite the same.
Known Heights Calibration	**6/10** You would have to select 4 points on the camera feed as a part of the setup.	**9/10** The observed calculated heights seemed accurate, and the vertical distance distribution of pixels was visually equivalent.	**9/10** The vertical known heights are equidistant when viewed and can be used to calculate a strong vertical pixel-to-distance ratio
OpenCV	**3/10** Very slow and isn't very practical for an app that uses opencv to be usable for most users.	**5/10** Could detect the balle quite accurately, but only up close. Not as practical for an app to use	**7/10** Realistically, opencv is quite mathematically supported, it uses well-know strategies to be able to detect things
Ultralytics YOLO	**7/10** Easy to integrate into mobile apps, but takes a while to integrate with roboflow pytorch files.	**9/10** Real-time object detection is accurate and consistent across varied environments with good lighting.	**8/10** Uses convolutional neural networks, which are mathematically grounded but abstract

We chose the known heights calibration because of the mathematical accuracy it provides. The observed height still needs to be rigorously tested but has the most significant potential of the options. The multi-camera detection option was not considered because it did not align with our cost-effective user needs and requirements for fast computations.

4.3 Proposed Design

4.3.1 Overview

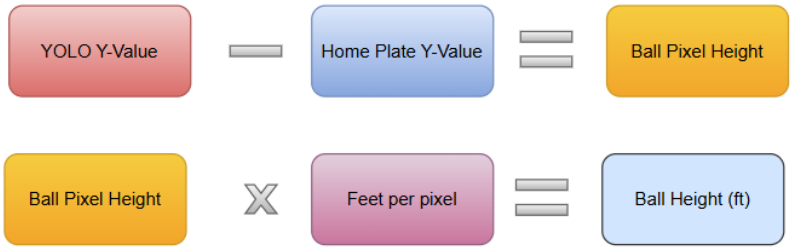
Starting with the Flutter app on a mobile device, users will have access to set up calibration values (min height, max height, and reference height) for the camera to fit their specific environment. When they enter the tracking screen, our app will utilize the Ultralytics-Flutter plugin to access their device's camera and begin feeding it into our YOLO model. With the plugin, we are able to execute the model in the Flutter environment. The YOLO model will track the ball based on a custom dataset that we created. The model will then return the coordinates of the ball which we can then use to calculate the height using our pixel conversions. In addition, we do not want our model to be tracking the height of the ball all the time as that would result in illegal calls being made when the ball is in play. So, we also developed an 'arming' system that involves drawing a bounding box around the pitcher's mound. When the ball is found in that box for two seconds, our tracking algorithm triggers and the app begins checking for an illegal pitch.

4.3.2 Detailed Design and Visual(s)

Calibration:

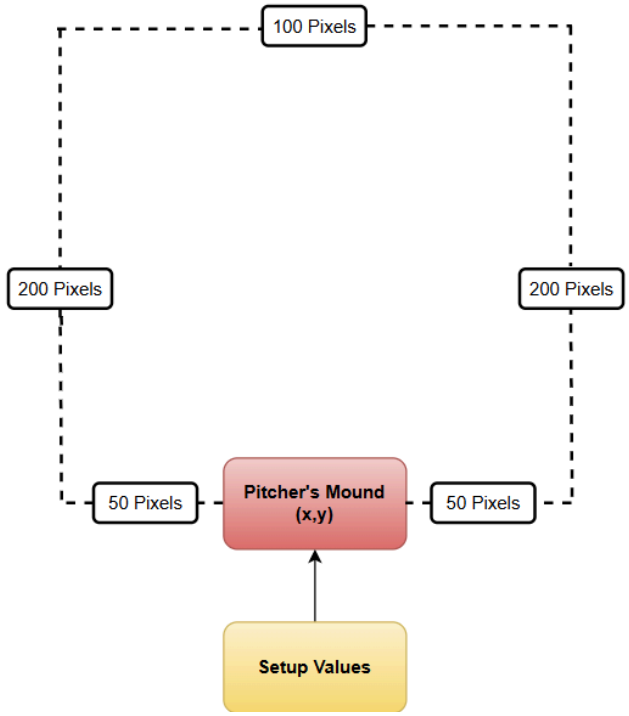
- Use YOLOv8 model to track the ball and collect the coordinates of the home plate, a reference height, and the pitchers mound.
- 

Calculate Height:

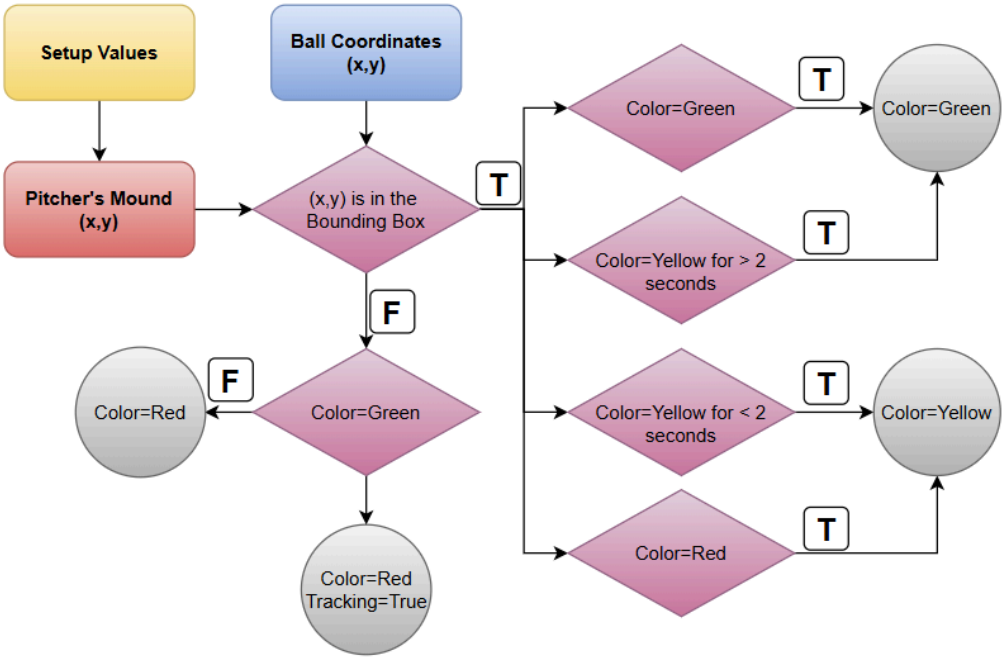
- Use YOLOv8 model to track the ball and collect the coordinates when pitched
- 

System Arming:

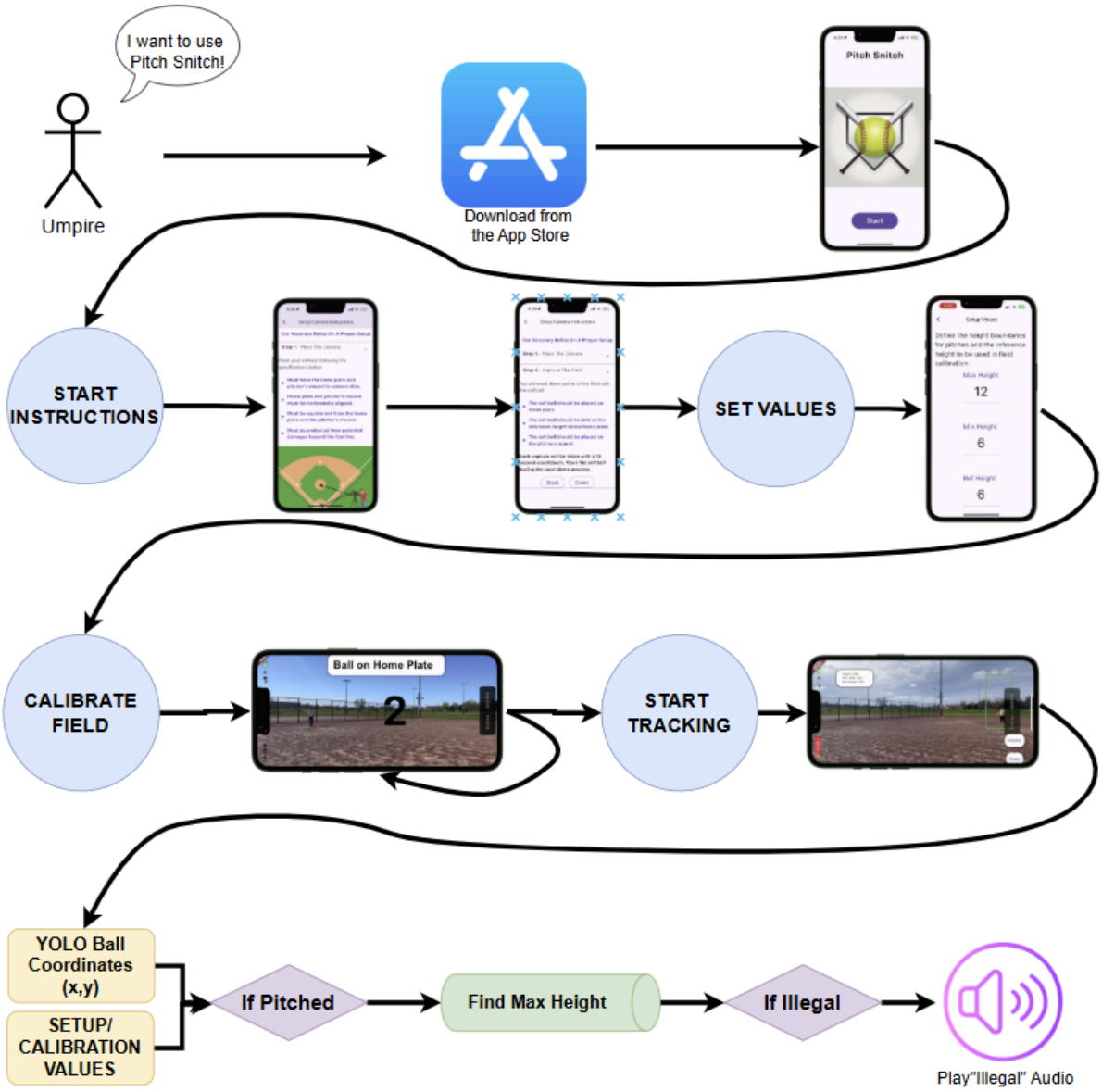
- Map the Pitcher's Bounding Box



- Box Status Indicators (Red, Yellow, Green)



4.3.3 Functionality



4.3.4 Areas of Concern and Development

Currently, we have a complete implementation of our app. Users are able to specify the known height in the app, and through our calibration process, are able to use the app to track the height of a softball. However, the main area of concern for our application is the robustness and accuracy of our machine learning model. In the model's current state, it is not sufficiently trained on all of the various lighting conditions that may be present during a softball game. To mitigate this issue, the model can be trained further on datasets with more lighting variety to increase the overall accuracy. In addition, we have been experimenting with different versions of YOLO models, mainly YOLOv8

and 11. We need to be able to find the model that provides the best combination of accuracy and performance, which is the end goal of making it best for our users.

4.4 Technology Considerations

Our project consists of multiple components that help integrate the two main components of our design: object detection and mobile app development. We use Google's app development tool Flutter, which can be compiled to run on an Apple iPhone, Android mobile device, or mobile emulators. However, due to compatibility issues with the Ultralytics plugin, we have prioritized the IOS platform for our application. Flutter uses Dart for an accessible widget-based design. Flutter's plugin functionality allows a smoother cross-platform development when utilizing device components such as the camera. Using Flutter, we can satisfy our technical challenges of developing an aesthetically pleasing and accessible user interface with an Ultralytics backend.

Integrating into Flutter's backend, we use the Ultralytics-Flutter plugin to track the softball in the camera frame via machine learning. There exist many sports ball tracking systems currently on the market. However, these systems typically guarantee accuracy with a multi-camera setup with expensive, high-quality cameras. Our design accommodates local league use with a camera on a mobile device. With a singular camera on the field providing one plane of vision, our application's accuracy relies on precisely detecting the softball within a camera frame. We do this using a custom trained YOLO model, which detects an object using machine learning. With our machine learning approach, we can use known heights provided during calibration to calculate the height using the ball coordinates returned from the YOLO model.

4.5 Design Analysis

Using our machine learning based design, we have been able to fully integrate our object detection and height calculation methods into our app. The Ultralytics plugin allowed for a seamless integration of our YOLO model into our Flutter environment. Currently, our app is able to create the pixel conversions we need to calculate height using provided known height values, our YOLO model then tracks the ball and returns the coordinates where it was found, and we use those coordinates in our height calculation to return the height of the ball. With this design, we have confirmed that it is possible to determine the height of the ball using a single camera.

From here, further refinements of our machine learning model are needed to account for the various conditions that can be present during a softball game. Currently, our model provides accurate results under ideal conditions, but we want to ensure that our users can utilize our app under all conditions. Specifically, our model occasionally detects objects that are not softballs, meaning that the model is either overtrained, or is not trained well enough. Through model refinements and further training, we expect that our app can be completely functional under all conditions. We also need to ensure the efficiency of our app. The YOLO model does require reasonable computation power, which has revealed issues when doing things like screen recording the phone while running our app, which often results in crashing. While this is a specific scenario, we aim to take these issues into account so our app can run as expected without reduce the performance of other aspects of the device.

5 Testing

5.1 Unit Testing

Our solution is primarily software based; this implies the necessity for proper unit testing for each separate piece of functionality within our code. Because our main solution functionality is divided among two separate technical areas, the Flutter frontend and our tracking backend, unit testing is treated differently between the two implementations.

Within our frontend, unit testing occurs with each individual "widget", or visual component of our user interface to ensure items appear correctly on screen from our mobile application. This testing ensures that the user experience of our app remains consistent and that there is a high ease of access for users. It also ensures that data is properly being accessed within our application between components to ensure accuracy. In terms of the tools used for unit testing within our frontend, Flutter comes bundled with real-time debugging tools, useful for diagnosing and troubleshooting problems that occur when running an app. These tools can be accessed via a locally hosted interface accessible upon starting the Flutter app.

For testing the backend, we focused on the performance of our machine learning model. To test this we started by reserving sections of our prepared dataset for testing and validation. When the model goes under training, it takes a subset of the training frames from our dataset, trains on those frames, then runs the model against a subset of validation frames. This process was completed over a series of 400 epochs. Once the model was done training, we ran it against the testing frames to verify its performance. Further testing was done once the model was integrated into our app. Initially, we added a shortcut to the model within our app that bypassed the calibration section. From there, we were able to run the model directly to verify that it was able to function within our app.

5.2 Interface Testing

The interfaces included in our design include the user interface built within Flutter along with the communication between the Ultralytics-Flutter plugin and the Flutter side. The graphical user interface was testing by running our app and ensuring that all of the widgets were behaving as expected. We also verified that certain function calls were being entered by adding print statements within the functions that were displayed via the Flutter debugging output. For the Ultralytics plugin, we were able to directly integrate the plugin within our app, so no external communication was necessary. The Ultralytics plugin was verified for correctness with the aforementioned model shortcut, along with print statements being added where necessary. As there was no external communication between Flutter and the plugin, just being able to run the model showed that our interfaces were behaving properly.

5.3 Integration Testing

The main focus of our integration testing was ensuring that we were able to use our model after performing the calibration. To verify this, we first displayed the calibration values after setup via the debugging output. This confirmed that our initial values were being initialized correctly. After calibration, the app moves to the actual tracking screen which utilizes the Ultralytics plugin. We then printed out the setup values in that screen to confirm that the calibration values were being passed to the tracking screen properly. Next, we ensured that the model itself ran properly after moving to the tracking screen by displaying a red dot if it detected the ball. With these steps, we

were able to ensure that the integration between Flutter and our machine learning model behaved as expected.

5.4 System Testing

We performed system testing by using our app in a realistic environment. To achieve this, we went out to the recreational softball fields, where we mounted a phone on a tripod and ran our app. From there, we verified that our app functioned normally. In addition to Flutter's reloading capabilities, we were able to make small corrections to the code where needed and reload our app while it was running to verify correctness. Based on the results we gathered from each testing session on the field, we were able to make modifications to our code to mitigate issues that arose during testing. After fixing any issues, we went back out to the field to test further.

During testing, we also verified our height calculation algorithm by holding the ball at a known height. With the known height, we could observe the height that the model returned and verify that what the model returned was correct. As we stored and displayed the setup values that are used in our height calculation, we were able to easily make adjustments based on what the model was outputting.

5.5 Regression Testing

The main focus of our regression testing was based on changes we made to our machine learning model. As Flutter is modular by design with the widget-based implementation, it was easy to make changes to the Flutter side without changing large sections of code. For the machine learning model, the Ultralytics plugin made it easy to specify what model we wanted to use. So, as we made further refinements to our model, we were able to easily plug-in the new model and verify its correctness by testing our app on the field.

5.6 Acceptance Testing

To verify the correctness of our system, we based it on our design requirements. With our requirements, we could easily tell if our app was providing accurate results. Specifically, we wanted our app to be easy to set up as well as the illegal call being faster than a normal umpire with an height accuracy being within ±4 inches.

To ensure this, we have members of the ISU softball team use our app to see how easy the setup and use of our app would be for someone who has never used it. This ensured that we had an unbiased opinion of our app along with accounting for any potential user errors that we missed. Finally, we were able to verify our height calculation by holding the ball at a known height.

5.7 Results

Based on our testing, our app functions very well based on our requirements. Our app is able to detect the height of the ball in a ±6 inches, which is not within our desired range; however, future refinements of our machine learning model would likely move the detected height within our desired range. Next, our app was able to detect and call an illegal pitch in .22 seconds, which is just above our desired goal of .2 seconds. Finally, for ease of use and setup, the ISU club softball members stated that the setup of our app was simple, and that they would trust a system like this to be used during actual softball games.

While our end metrics don't quite meet our previously set goals, we believe that with further training of our machine learning model, the performance of our app should fall within our desired goals. In the end, we feel that we created a successful implementation of the desired product.

6 Implementation

Object Detection:

For our final deliverable we settled on utilizing only a YOLO model for object detection. We were able to solely rely on the model because we were able to integrate the Ultralytics Flutter plugin into our app which allowed us to utilize the iPhone CoreML hardware. By taking advantage of the phone's dedicated hardware for ML tasks, model inference speeds were quick enough to rely on it for tracking and detection.

Pitch Detection:

As mentioned in 4.3.2, we first detect when a pitch is thrown before deducing if a pitch is illegal. We do this by creating a bounding box around the pitcher's mound and triggering our tracking only when a ball is thrown after the pitcher is holding the ball within the bounding box for 2 seconds. Additional logic for our pitch detection and system arming can be found in section 4.3.2.

Height Detection:

As mentioned in 4.3.2, we implemented our height calculation to use the coordinates of the ball, home plate, and a known reference height, using coordinates from the YOLO model, to convert pixel height to a height in feet.

We observed in the image below, that when we horizontally align the pitcher's mound and the home plate within a camera frame, that the vertical pixel-to-feet ratio is constant. Each green line represents a reference height, measured on both the pitcher's mound and home plate, and connected by the green line. The distance between each line is one foot, and the pixels between lines remain constant for each new height.

Using this knowledge, we can calculate the average pixel length from the known reference height and the home plate, calculate the pixel height of the ball, and convert the pixel height to a height in feet. When the maximum height of the pitch is found, we can then determine if the pitch is illegal using the maximum and minimum height values.

Screen Development:

Our application contains multiple screens that flow from an initial application landing page to the main camera screen as pictured below. These screens include:

- **Landing Screen**: acts as an initial screen which is presented to the user. This is the root of the application and is where the user starts navigation.
- **Instructions Screen:** instructs the user on the method of calibration that is essential for our applications functionality. This screen includes information as well a guided visuals on how to properly calibrate our application on the following screen.
- **Setup Values Screen**: the user inputs a reference height (usually the height of a player or a known-length object) as well as maximum and minimum ball thresholds.
- **Calibration Screen**: a series of timed camera screens that are used to obtain ball coordinates as well as reference height points for height calculation.
- **Camera Screen**: displays the currently tracked softball as well as other information essential for an end user to discern the height-tracking capabilities of our application.

The implementations of these screens as well as visual references to them can be viewed under appendix one below.

7 Ethics and Professional Responsibility

7.1 Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	Definition	IEEE Item	Team Interaction
Work Competence	Perform professional, high-quality work	"To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors..."	The team focused on this area by developing solutions and experimenting with modern technologies to ensure we create the best solution.
Financial Responsibility	Deliver reliable products that are reasonably priced	"To be honest and realistic in stating claims or estimates based on available data..."	Our design choices are based on usability, such as having the app be free and not requiring an additional camera, which upholds this area.
Communication Honesty	Report truthful work to shareholders	"To be honest and realistic in stating claims or estimates based on available data..."	We have upheld this area by meeting weekly with the client and frequently communicating our progress and design choices.
Health, Safety, Well-Being	Minimize shareholders' health and safety risks	"To hold paramount the safety, health, and welfare of the public..."	The team wanted to minimize injury risk during the setup of our product, which helped lead us to our app-based solution, which eliminated the need to set up multiple cameras.
Property Ownership	Respect the property of others	"To credit properly the contributions of others..."	We followed this area by acknowledging the individual contributions of the team and by collaborating on different areas of the project so multiple

			members could gain knowledge in multiple areas.
Sustainability	Protect environmental and natural resources	“To strive to comply with ethical design and sustainable development practices...”	We focused on this area by making our app cross-compatible and usable on multiple iterations of phones, meaning our users will not need to continue to purchase new devices to use our product.
Social Responsibility	Make products that benefit society	“To improve the understanding by individuals and society...”	Our team has a responsibility to produce accurate data and to better the experience of rec league slowpitch softball players and umpires, so we focused on making our solution easy to use and available to slowpitch leagues.

Best Area of Responsibility; Social Responsibility:

We have chosen to approach this standard by designing our project to prioritize affordability, portability, and easy accessibility for local communities to have access to simple officiating assistance. Although this further complicates our software tracking model, it simplifies the application for the community. This decision in our development was ethically driven with the community in mind rather than solely for efficient development.

Worst Area of Responsibility; Sustainability:

We haven’t been prioritizing the efficiency and runtime of our code up to this point as we’re trying to get prototypes working, which could result in more power consumption and more contribution to negative environmental impacts. Our team plans on working to not only write code but to refine it to be as efficient as possible. Thus reducing the amount of power needed to run our app.

7.2 Four Principles

	Beneficence	Nonmaleficence	Respect for Autonomy	Justice
Public health, safety, and welfare	Our product aims to improve the state of rec league softball games, which benefits the enjoyment of everyone involved.	We are avoiding using multiple cameras which reduces risk during setup and improves the usability of our product.	We allow users to store past pitches, so they can make their own decisions based on non-subjective data, improving the enjoyment of the game.	Our app will provide unbiased, real data to help officiate games, which will improve the fairness of the game.
Global, cultural, and social	Promotes fair play by providing an objective tool for measuring pitch height, which will reduce arguments between players and officials.	Keeping our solution focused on a single device allows a normal game of softball to play normally without disrupting the existing league culture.	Since the height range of our app will be customizable, different cultures and leagues can set up the app to work for their needs.	The height range customization of our app will allow multiple leagues with different rules to improve the equality of how the game is called.
Environmental	Focusing on developing our project on a phone reduces the need for multiple tracking cameras benefitting the environmental impact.	Avoiding multiple cameras allows our users to use an existing phone instead of having to pay and ship a new camera.	Our product allows users to choose to use their existing phone instead of making more environmental impacts by paying for more cameras.	Our app will be able to be utilizied by all leagues, even ones with limited resources. This keeps them form needing to invest in other cameras that may contribute to environmental damage.
Economic	Our solution is free which eliminates the need to purchase expensive tracking equipment.	Using one device to perform all of the tracking helps us keep the app free because we can forgo the cost of an additional camera.	A league can make their own choices on whether or not to use our app or not, which gives them control over any economicals adjustments it my require such as a league phone to use the app.	The free price point of our app allows a wide range of leagues with different financial backgrounds can utilize our app to improve the league.

Area We Focused On; Public health, safety, and welfare X Beneficence:

We are aiming to improve the enjoyment of the game. We plan to achieve this by ensuring simple setup for umpires, returning accurate data so the right calls are constantly being made, and being configurable for different height ranges so it can be used by a variety of leagues.

Area of Improvement; Environmental X Respect For Autonomy:

Based on the result of further testing, we may end up needing to restrict our app to only work on phones with a certain camera quality. Older phones with worse may not be able to provide the data needed for accurate tracking. This could result in leagues without immediate access to a high quality phone having to purchase a new device through shipping methods that contribute to environmental damage.

We plan on overcoming this negative aspect by ensuring our app can run on phones that are widely available and already owned by a majority of people. This will keep the availability of our app high and will reduce the need to purchase a new device.

7.3 Virtues

- Accuracy
    - We strive to provide accuracy for our users, as it is important for illegal pitches to be called correctly so as to not cause distress between players, officials, and fans.
    - We will achieve this virtue by continuously refine our ball tracking code and provide easy-to-use calibration steps so our app can be calibrated to fit the needs of each field.
- Empathy
    - Our app is solely focused on bettering the game for players, officials, and fans. We want to ensure that we are always prioritizing their needs over what makes it easier for us.
    - Communicating our design choices with our client and collecting feedback from testing are how we will ensure that we are always taking user needs into account.
- Innovation
    - We want our product to be easily integrating into existing leagues. So, we are innovating a single camera design approach that will make it easy to set up without the needs for multiple cameras.
    - Constant prototyping and research into different tracking methods will help keep us engaged with new ideas and strategies as to how to improve our single camera solution. Current team innovation like or height line system are already helping us improve our height detection methods.

Each team member should also answer the following:

- Identify one virtue you have demonstrated in your senior design work thus far? (Individual)
    - Why is it important to you?
    - How have you demonstrated it?
- Identify one virtue that is important to you that you have not demonstrated in your senior design work thus far? (Individual)
    - Why is it important to you?

- What might you do to demonstrate that virtue?

- Andrew
  - Demonstrated Virtue: Open-mindedness
    - Throughout the project, I consistently researched new and more effective ways for our team to track softball. As our project constraints evolved, I adapted my approach to object detection accordingly. For example, after spending time developing a solution using OpenCV, I discovered the Ultralytics plugin, which allowed us to run a YOLO model on the phone with fast enough performance to handle both detection and tracking. Even though this made much of my earlier work obsolete, I was willing to pivot and focus on learning how to integrate the new approach because it was ultimately better for the team and the project.
    - This virtue matters to me because in software engineering, nothing stays static, tools, requirements, and best practices change constantly. Being open to new ideas, even when it means moving on from something you've already invested in, is critical for building the best possible solution.
  - Non-Demonstrated Virtue: Communication
    - Communication was one virtue that I find very important even though I may have not been the best at it during this project. I often got deep in the weeds of researching and experimenting with different object detection technologies but ended up failing to fully communicate what I was doing and why. Communication is important to me because no matter how good the technical solution is, it doesn't matter if the rest of the team doesn't understand it or can't efficiently work with it or integrate it with the rest of the application.
    - Moving forward, I plan to be more intentional about communicating decisions, updates and reasoning behind changes, by summarizing key shifts in meetings and documenting decisions in shared notes.
- Casey
  - Demonstrated Virtue: Persistence
    - I have primarily been able to aid in code refactorization, UI revamping, and testing this past semester. I feel that my contributions aided in the success of our project despite various roadblocks that may have occurred along the way.
    - This virtue is important to me as being able to balance project needs with other work, study, or personal needs is critical to the success of a project.

  - Non-Demonstrated Virtue: Communication
    - I feel that this semester my communication wasn't as effective as it has been in the past. There were times with work, studies, and other

circumstances that caused me to regrettably fall behind on some important decisions; this caused miscommunication and error in my work when certain features were being worked on in parallel, for example causing certain feature changes to become obsolete in the face of new features being pushed out.

- Ultimately, this was a learning experience for me, and I am confident in our final product and my ability to communicate effectively in the future. I will continue to work on asserting myself in team spaces to ensure my progress aligns with others.

- Sully
    - Demonstrated Virtue: Persistence
        - With any project, there are bound to be roadblocks that come up that prevent progress. I believe it is essential that I am persistent in attacking these roadblocks and continuing to experiment so that we can solve our problems.
        - I demonstrated this virtue by continuing to attempt to integrate our C++ code into the Flutter app. There have been many issues with the integration, mainly stemming from the OpenCV libraries having issues running on iOS, and I need to persist through this issue so we can complete our app. While I haven't been able to fully integrate the code yet, I have made good progress through the OpenCV tracking framework for iOS I helped create.
    - Non-Demonstrated Virtue: Adaptability
        - We are utilizing many different technologies in the making of our app. So, it is important that I become familiar with the all of the technologies we are using, so I now how all of the components work together in detail.
        - I will demonstrate this virtue by pairing with other team members who are working on different areas and learning how they developed their ideas so I can help with issues in the future.

- Cameron
    - Versatility
        - This project has brought me outside of my comfort zone numerous times and required me to do things I have no experience with. Learning flutter and all of the ML topics was new to me and forced me to go outside my current knowledge to complete the tasks.

        - This is important to me because it is a necessity in the workforce. New technologies are constantly popping up, so a good engineer must be able to adapt to these new technologies and learn how to best use them.
    - Communication
        - I felt like this semester, I wasn't always communicating well with my team and that hurt my work. With better communication I would be able to work through issues quicker and help the engineering process as whole.
        - Hopefully, I'll be able to attend the weekly meetings next semester which will help immensely. Other than that, I just need to be more willing to reach out to my teammates for help and to keep them updated on my progress.
- Ethan
    - Demonstrated Virtue: Diligence
        - Throughout working on this senior project I felt I showed the virtue of diligence by persistently achieving weekly goals. I did this by making weekly update presentations for our advisory and team to outline our contributions for the week, the current integration of our components, and the steps we must take to move forward. Diligently staying updated with my team and planning for the future was my strongest virtue I showed in this project.
    - Non-Demonstrated Virtue: Conviction
        - During many of the project's life cycle there were times where I believe something wasn't working effectively or there was an issue with the team's development cycle. However, many times I did speak up to these issues. My actions and communication between the group occasionally did not align with my beliefs and I should have spoken up about issues within our project and showed conviction to stand up for what I believe was right. I will work on speaking up in groups and being direct in my thoughts of situations that arise during development cycles.
- Josh
    - Demonstrated Virtue: Creativity
        - One virtue I feel like I demonstrated pretty well during this project for the most part was creativity, and coming up with different and unique

solutions to some of our problems we were having within the group. We had quite a few problems and issues as a group, on top of just individually, but I felt like I did a good job at thinking of creative ways to come up with a solution to our project and different small problems.

- Non-Demonstrated Virtue: Persistence
    - One thing I struggled with personally on this project was finding the strength and strategies to push past some of my or our team's struggles. I obviously tried my best to get past some of these struggles, but I still felt like I was stuck on a wall at some parts of the project and even with some creativity some lack of solutions definitely demotivated me a little bit. Additionally, sometimes I wasn't sure in what direction I could help the project in at certain times, and I definitely tried improving this part of myself during most of this project to get to a point where I'm confident I can fix these issues to be less of an issue in the future.

8 Closing Material

8.1 Conclusion

We were able to fully integrate the three main parts of our project laid out from the first semester. This included the object and tracking functionality, calculations for determining the height of the pitch and a full app design. We ended up pivoting from using C++ code in our app since it caused performance issues while running Flutter and we found a better way to integrate the YOLO model into our app that allowed us to take advantage of the CoreML architecture on newer iPhones. By utilizing the ML architecture we were able to massively improve inference speeds to the point where we no longer needed OpenCV's tracking modules further reducing our need for running C++ code. Unfortunately though due to time constraints and issues with translating our trained YOLO model to a format compatible with Androids ML architecture we were unable to make our app cross platform. In the future if another senior design group ends up working on our project they will need to focus on training a more refined version of our YOLO model, translating it to a format compatible with Android and performing more exhaustive testing.

8.2 References

[1] "C Interop using Dart:FFI," Dart, https://dart.dev/interop/c-interop (accessed Dec. 7, 2024).

[2] "Required packages," OpenCV, https://docs.opencv.org/4.x/d5/da3/tutorial_ios_install.html (accessed Dec. 7, 2024).

[3] "Material library," material library - Dart API, https://api.flutter.dev/flutter/material/material-library.html (accessed Dec. 7, 2024).

9 Team

9.1 Team Members

- Ethan Gruening
- Casey Gehling
- Sullivan Fair
- Josh Hyde
- Cameron Mesman
- Andrew Vick

9.2 Required Skill Sets for Your Project

- App Development
- Flutter Framework
- Machine Learning
- Yolo/Ultralytics
- IOS Development

9.3 Skill Sets Covered by the Team

- Ethan Gruening: App Development, User Testing, Android Studio Emulation
- Casey Gehling: App Development, Flutter
- Sullivan Fair: App Development, Frameworks
- Josh Hyde: Flutter, App Development, Testing, Android
- Cameron Mesman: App development, Flutter, C++
- Andrew Vick: App development, ML training, Swift, Flutter

9.4 Project Management Style Adopted by the team

- Agile

9.5 Initial Project Management Roles

- **Team Organization**: Ethan Gruening
- **Client Interaction**: Casey Gehling
- **Machine Learning Integration**: Andrew Vick
- **Individual Component Development**: Sullivan Fair
- **Research**: Josh Hyde
- **Testing**: Cameron Mesman

9.6 Team Contract

Team Name _____sdmay25-11_____

Team Members:

1) _____Andrew Vick_____ 2) _____Casey Gehling_____

3) _____Cameron Mesman_____ 4) _____Joshua Hyde_____

5) _____Ethan Gruening_____ 6) _____Sullivan Fair_____

Team Procedures

Day, time, and location (face-to-face or virtual) for regular team meetings:

1 pm every Wednesday (possibly)

2 pm every Monday with advisor (Dr. Fila)

Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):

Discord

Decision-making policy (e.g., consensus, majority vote):

Majority Vote

Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

Andrew Vick will record meeting minutes, which will be stored in a document on our drive, and I will send a message in Discord for that meeting.

Ethan Gruening will record meeting notes, which will be stored in a document on the team's shared drive.

Participation Expectations

Expected individual attendance, punctuality, and participation at all team meetings:

We are expected to attend, share, and be on time for all meetings

Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

Everyone is expected to meet deadlines, make meaningful contributions to the project, and be on time.

Expected level of communication with other team members:

Everyone is expected to be active in the discord and respond to teamwide questions within one business day.

Expected level of commitment to team decisions and tasks:

Everyone is expected to be committed to team decisions and tasks. This extends to actively engaging in discussions and conveying your ideas.

Leadership

Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):

Team Organization: Ethan Gruening

Client Interaction: Casey Gehling

Machine Learning Integration: Andrew Vick

Individual Component Development: Sullivan Fair

Research: Josh Hyde

Testing: Cameron Mesman

Strategies for supporting and guiding the work of all team members:

Weekly standup meetings where we discuss the week's events and ensure everyone is keeping up with the workload.

Strategies for recognizing the contributions of all team members:

During weekly standup every member will share what they worked on and any information they found regarding the development of our product. This allows every member to share their contributions and receive feedback.

Collaboration and Inclusion

Andrew Vick, Software Engineering. Some of my relevant skills include C/C++, Python, and IoT devices. Most of my experience has come from personal projects for instance, using Python, I created my own virtual assistant.

Casey Gehling, Computer Engineering. Relevant technical skills are Embedded Programming, Networking, Fullstack Development, and UI Design. Additionally, I have relevant experience in client communication and requirements gathering.

Sullivan Fair, Software Engineering. Relevant skills include general coding knowledge of Java, C, C#, Python, JavaScript, AWS knowledge, GIT, and cybersecurity knowledge.

Ethan Gruening, Software Engineering. My relevant skills include, but are not limited to, Python, C, Java, JavaScipt, and AWS. Most of my projects/experiences involve user interphases, I/O devices, and API service integrations.

Josh Hyde, Computer Engineering. My relevant skills are generic coding in C/C++ and Java, and some database work with MySQL. I also have been in different groups before and have had to work together towards a common project goal.

Cameron Mesman, Computer Engineer. My relevant skills are coding with C and Java. I have experience programming embedded systems, app design (backend, frontend, and database languages), and computer architecture.

Strategies for encouraging and supporting contributions and ideas from all team members:

Goal-Setting, Planning, and Execution

Team goals for this semester:

Create a full implementation of our app

Perform user testing

Strategies for planning and assigning individual and teamwork:

During team meetings, we will identify what needs to be worked on for the week. From there, we will assign who tackles which task based on their expertise.

Strategies for keeping on task:

Weekly goals and check-ins

Consequences for Not Adhering to Team Contract

How will you handle infractions of any of the obligations of this team contract?

In the event of an infraction by a team member, the team as a whole will reach out to the member to see why they aren't adhering to our contract.

What will your team do if the infractions continue?

If the infractions continue and we cannot resolve the issue internally, we will finally reach out to Professor Fila regarding the member.

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the

consequences as stated in this contract.

1) _____Sullivan Fair_____ DATE __09/18/2024_____

2) _____Andrew Vick_____ DATE __09/18/2024_____

3) _____Casey Gehling_____ DATE __5/1/2025_____

4) _____Ethan Gruening_____ DATE __5/1/2025_____

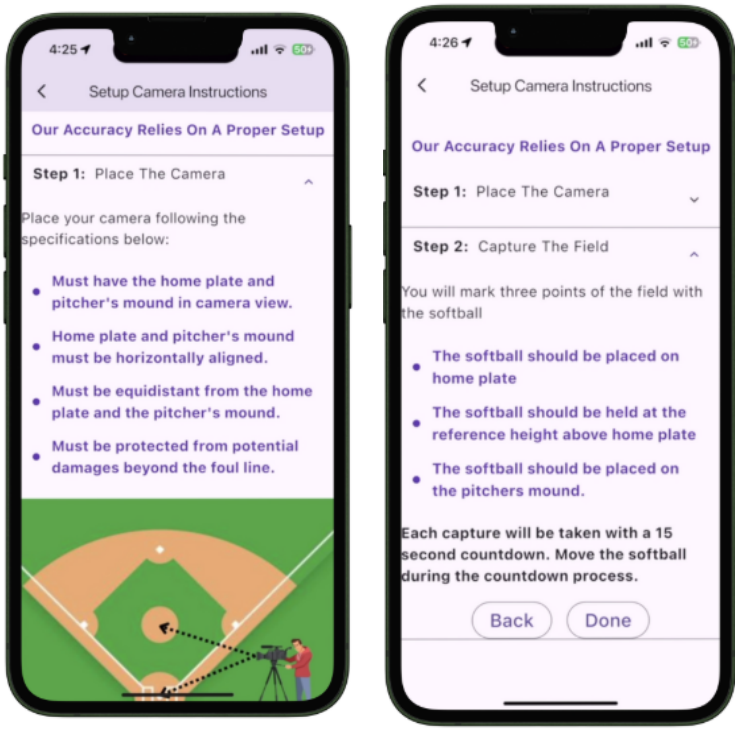
5) _____Josh Hyde_____ DATE __4/28/2025_____

6) _____Cameron Mesman_____ DATE __5/1/2025_____

Appendix 1: Instruction Manual

1. Open the app
    a. The user will see the Home page with a ‘Start’ button

2. Click ‘Start’
    a. The user will move to the instructions screen with details on how to properly set up the camera and calibration



3. Click ‘Done’

a. The user will be presented a screen to input their desired calibration values

4. Click 'Start Calibration'
    a. The user will be prompted to place the ball at three positions: on home plate, held at the reference height, and on the pitcher's mound
    b. There will be a 15 second countdown before each picture is taken

5. After the calibration process, the user will be taken to the tracking screen
    a. From here, the app is fully set up and the camera should not be moved as it would compromise the calibration
    b. The ball needs to be in the bounding box for two seconds for the system to arm itself


DEBUG
78
Height: 4.98
Yolo: [185, 702]
Max Height: 0.00
21.4 ms — 29.9 FPS
5:38
Home
Data

Appendix 2: Design Considerations

The team was able to create a working implementation of our app. However, there are certain areas of our design that we know can be improved upon in future work. These areas include the YOLO model, testing height accuracy, system stress testing, and adding a review section for past pitches.

First, our YOLO model is very one dimensional, meaning that it is trained in only a single lighting conditions in an ideal scenario. While we added augmentations to the images to make the model more robust, it likely isn't enough to cover the various lighting conditions that can be seen during a softball game. Training the model on a more diverse dataset will help make it more well-rounded for a variety of game conditions.

Next, further testing should be done on the height calculation. We currently have an accuracy of ±6 inches; however, because this is the core part of our app, more testing needs to be done to ensure that the range is consistent. Further training of our YOLO model will also result in more testing being needed for the height calculation as the accuracy of the model can directly affect the height calculation.

Third, our system needs to be fully stress tested. We tested in pretty ideal conditions this semester to ensure the core functionality of our app would work in practice. That being said, more rigorous testing needs to be done to ensure users are not able to break the app. Rigorous stress testing will allow for unknown bugs and errors to be found as the app is pushed to its limits. These kinds of tests could include placing the camera in non-ideal positions, having drastic lighting changes occur, or having multiple softballs in frame at a single time.

Finally, we would have liked to have a feature where users could go back and review past pitches and how they were called. With a "machine" umpire, users will likely want a way to verify what the app saw and see if it is correct. Having a past pitches screen would allow for users to go back and review the app's performance and ensure that it is providing accurate results. However, due to current limitations with the Ultralytics-Flutter plugin's proprietary camera controller, we were not able to add it successfully.

Appendix 3: Alternate Designs

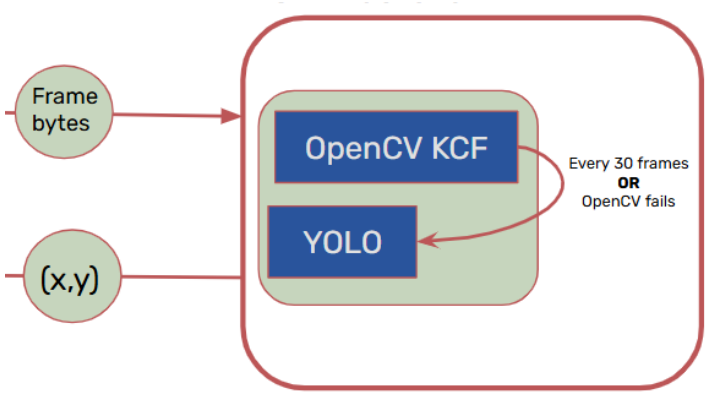
Throughout 491 and 492 we implemented many different prototypes to test the functionality of our application. Through testing and future development, the alternative design prototypes were overwritten or discarded from the final deliverables. It is important to note the designs that failed, and those that are now incompatible as they are a part of the design process and were a part of successful developmental cycles by leading our project in more effective direction.

OpenCV and KCF Tracking

In the planning phase of our development, we valued the accuracy of a system that integrates both machinelearning and non-machine learning techniques for object detection and object tracking. We developed multiple softball tracking prototypes that utilized the integration of OpenCV and YOLO tools to efficiently compute where a softball is located within a frame while restricting our reliance on machine learning models.

OpenCV is an image processing library for Python and C++ which can use multithreaded approaches to quickly analyze an image for specific colors, textures, and patterns. OpenCV also offers tracking modules which identify a moving object in frame, such as the KCF tracking algorithm.

In prototypes that were disconnected from the Flutter application, the OpenCV and KCF tracking tools performed very well when finding the softball by color, but could often not initially find the ball. The YOLO model, with a higher runtime and reliability, was able to initially find the ball and routinely check the OpenCV and KCF algorithms are still tracking the softball every 30 frames as shown below.



The YOLO model used within this prototype was a trained on a small collection of images where the softball was within 3 feet of the camera and indoors. This training data was too small and did not fit our application's true environment to train an effective model. As our development continued, we collected and annotated over 9,000 images in a new YOLO model, which performed at 30fps and had very accurate tracking results. The machine learning tracking outperformed the non-machine learning OpenCV and KCF algorithms and led us to focus on a machine learning object tracking system. Although the integrated tracking system was removed, it taught us the importance of replicating training data to the system's environment to better utalize modern computer vision tools.

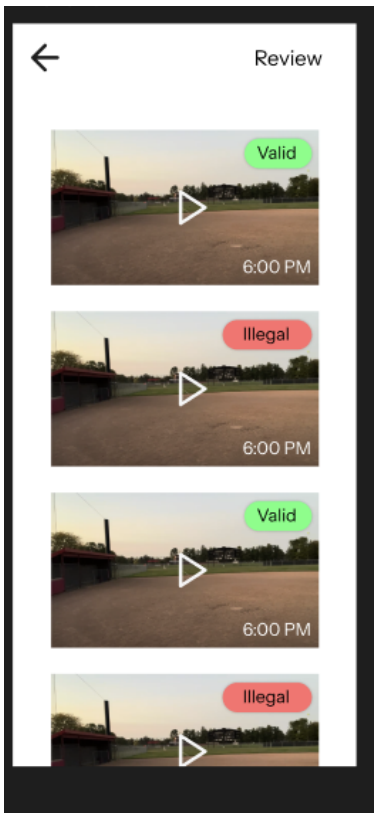
C++ Flutter Bridge

During the initial integration of our softball-tracking solution into our Flutter application, we were researching potential ways to create the bridge between our original C++ backend running OpenCV and our Flutter frontend. We had experimented with Dart FFI, a native Flutter library capable of interacting with C++ code from the front end of our application. We had made progress with constructing this bridge, even reaching the point where our model was callable from our front end returning with ball coordinates, albeit this process introduced much latency hindering the user experience of our application. We managed to obtain frames of real-time tracking which dramatically fell short of our project requirements, taking minutes to return each coordinate from the backend.

Our team attempted to reconstruct our application's communication between the frontend and backend and mitigate our CPU consumption by multithreading, passing memory pointers instead of images, and decreasing the image size. Ultimately, as we began to adopt a solution revolving around using the Ultralytics plugin to allow for automatic camera handling and pipelined model processing. The C++ backend prototype was removed as a underperformant alternative to the Ultralytics plugin. Although this prototype was removed from future development, it gave our team a chance to further investigate our app's CPU usage and analyze performance issues.

Past Pitches Screen

Originally, our application included recordings of past pitches as they transpired within a softball game. This was done by using Flutter's CameraController class to record from the camera and save the video within the application with an indication of "Illegal" or "Valid". The users could then click the "Past Pitches" button on the camera screen to view the last pitch.

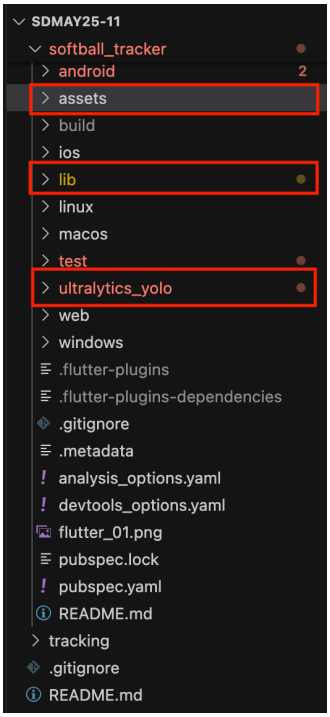


However, after adopting the Ultralytics plugin that used its own CameraController, it became difficult to refactor this feature since only one CameraController can be active at one time. Regaining access to the camera for the Past Pitches screen without heavily modifying the underlying source code revolving around the Ultralytics plugin would be too invasive to the library and remains an incompatible feature.

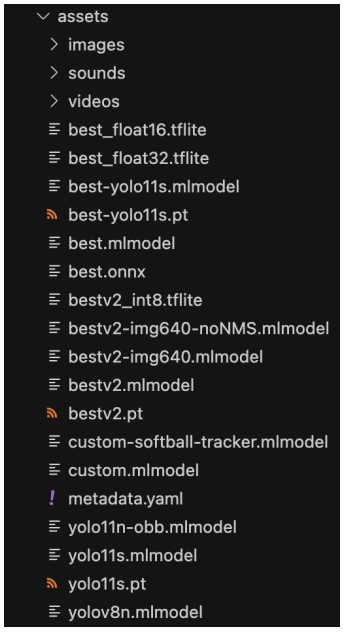
The Past Pitches screen prototype still encourages our team to find a way to communicate data from previously analyzed pitches to the user. With video unavailable, our team has discussed the opportunity to include graphs of the coordinates throughout the pitch, detected by the model, paired with lines indicating the maximum and minimum height requirements.

Appendix 4: Code Analysis

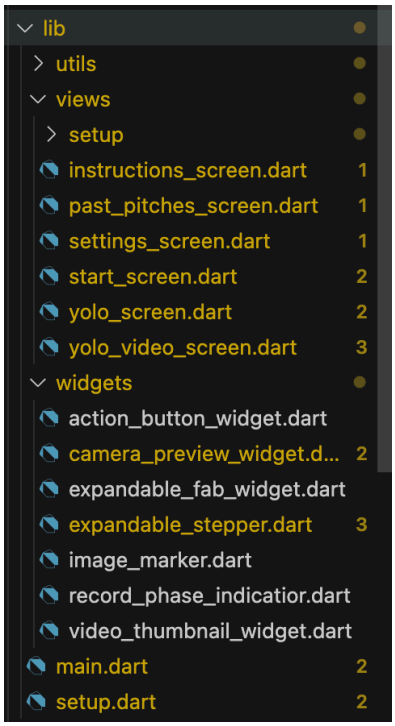
There are many directories in our code repository necessary for the Flutter framework. However, the bulk of our development lies in the 'lib', 'ultralytics_yolo' and 'assets' folders.



The assets folders contain the different YOLO models we trained, images, and the mp3 for the illegal audio. The best models we obtained from our training were the 'bestv8-img640.mlmodel', a YOLOv8n model, and the 'best-yolo11s.mlmodel', a YOLOv11s model.



For lib, that contains the various screens including the home page, instructions, calibration, and tracking screens. Main.dart routes the user to the various screens of the app. Setup.dart does the same things as main.dart, but for once the user enters the setup of the tracking. The rest of the .dart files in the views folder represent the various screens a user will interact with when using the app. Yolo_screen.dart is the screen where the tracking of the softball is performed. The widgets folder contains assets like buttons that can be plugged into various screens. This is one of the main benefits of the Flutter framework.



Finally, the ultralytics_yolo folder contains the entire Ultralytics-Flutter plugin that we included locally in our repo. The plugin is open-source, and because we needed to make some customizations to the code for it to work for our desired camera orientation, we copied it locally and made changes.