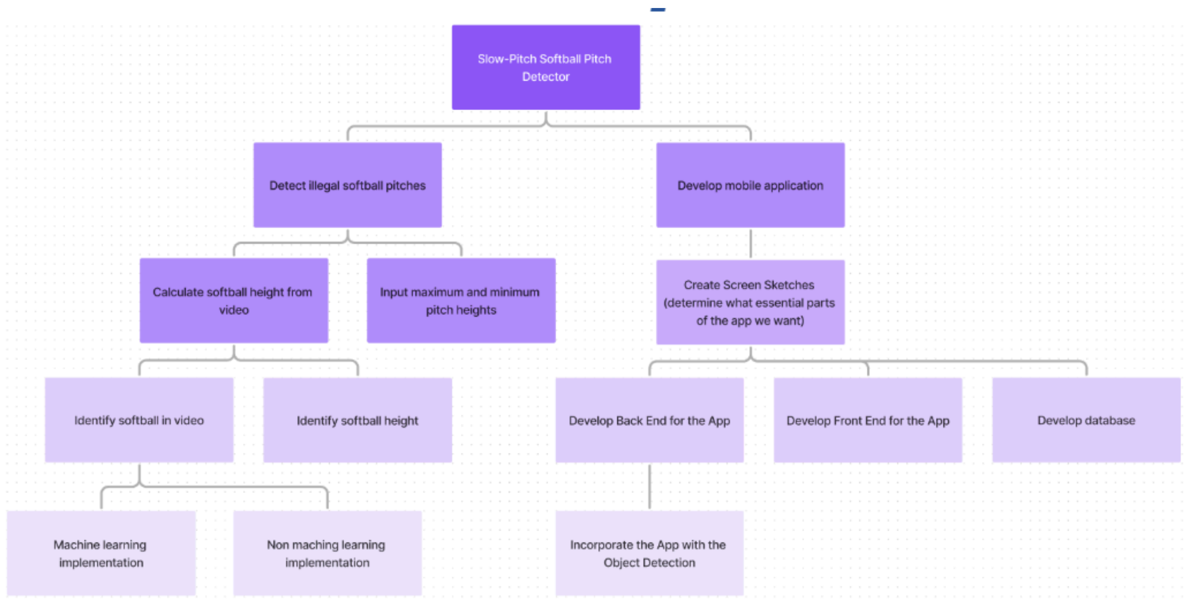
3.1 PROJECT MANAGEMENT/TRACKING PROCEDURES

Happy Halloween!

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 hybrid machine and non-machine learning approach and a mobile app utilizing Flutter and Dart FFI. 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 will help make a full implementation easier in the future. 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 need to be done. Using personal branches will help isolate development so we can develop individual components more efficiently without accidentally breaking the main branch of the project or having more than one person alter the same file in different ways.

3.2 TASK DECOMPOSITION


Slow-Pitch Softball Pitch Detector
Detect illegal softball pitches
Develop mobile application
Calculate softball height from video
Input maximum and minimum pitch heights
Create Screen Sketches (determine what essential parts of the app we want)
Identify softball in video
Identify softball height
Develop Back End for the App
Develop Front End for the App
Develop database
Machine learning implementation
Non maching learning implementation
Incorporate the App with the Object Detection

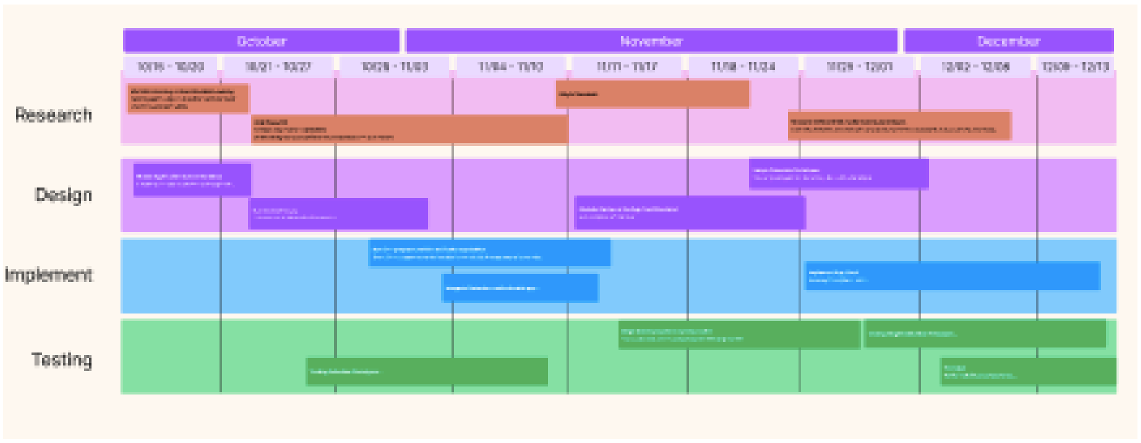
These tasks are intentionally left broad since we have not made final decisions on the entire design. Based on the agile methodology, many of these tasks will be adjusted as development continues and we receive feedback from the client. Any testing and prototyping we do could also affect our final implementation 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 pretty much sum up our project but are still broken down further in the image above.

3.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

What are some key milestones in your proposed project? It may be helpful to develop these milestones for each task and subtask from 3.2. How do you measure progress on a given task?

These metrics, preferably quantifiable, should be developed for each task. The milestones should be stated in terms of these metrics:
Machine learning algorithm XYZ will classify with 80% accuracy; the pattern recognition logic on FPGA will recognize a pattern every 1 ms (at 1K patterns/sec throughput). ML accuracy target might go up to 90% from 80%. 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: the research category, in which project specific applications and approaches are thoroughly assessed and evaluated, the design category, where compiled research is initially primed for development, the implement category, where designs created are prototyped and implemented into real applications, and the testing category where implementations are tested for their correctness and efficiency.

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**
10/15 - 10/20 (Sprint 1)	- Machine learning vs. non-machine learning research - Mobile application screen sketch design

10/21- 10/27 (Sprint 2)	- Flutter app research - Object detection prototype
10/28 - 11/03 (Sprint 3)	- Object detection prototype testing - Flutter and C++ integration research and development
11/04 -11/10 (Sprint 4)	- Implementation of object detection within mobile application
11/11 -11/17 (Sprint 5)	- Height calculation research - Modular design of full-stack application - ML model testing
11/18 -11/24 (Sprint 6)	- Height detection prototyping
11/25 - 12/01 (Sprint 7)	- ML model training techniques research (model refinement) - Full stack application implementation
12/02 - 12/08 (Sprint 8)	- Height detection testing
12/09 - 12/13 (Sprint 9)	- Mobile app integration testing

3.5 RISKS AND RISK MANAGEMENT/MITIGATION

For each task, identify all salient risks (certain performance target may not be met; certain tool may not work as expected) and assign an educated guess of probability for that risk. For any risk factor with a probability exceeding 0.5 and each high severity risk, develop a risk mitigation plan. Can you eliminate that task and add another task or set of tasks that might cost more? Can you buy something off-the-shelf from the market to achieve that functionality? Can you try an alternative tool, technology, algorithm, or board? Agile projects can associate risks and risk mitigation with each sprint.

Sprint 1:

- **Risks**:
    - ML may be overkill for object detection and too resource-intensive. (prob: 0.5)
    - Non-ML methods might lack the needed accuracy. (Prob: 0.6)
    - **Mitigation**: Explore using a combination of ML and non-ML strategies. ML can be used to check our non-ML tracking to correct for deviation.

Sprint 2:

- **Risks**:
    - Flutter's limitations in handling real-time processing or specific hardware task. (Prob: 0.3)

    - Object Detection prototype may not achieve the required performance or accuracy. Also may not integrate well with Flutter. (Prob: 0.9)
    - **Mitigation**: Rework prototype to use (C++/OpenCV) instead of Python.

Sprint 3:

- **Risks**:
    - Compatibility issues with FLutter and C++ integrations, particularly in efficient data passing. (Prob: 0.5)
    - Performance drop when integrating object detection into the mobile interface. (prob: 0.4)
    - **Mitigation**: Use Native Flutter plugins if integration lags. Test different data-handling techniques (i.e. JSON or shared libraries) to see what works best.

Sprint 4:

- **Risks**:
    - Computing power of mobile devices may limit detection effectiveness. (Prob: 0.6)
    - App design may require significant performance tuning to be responsive. (Prob: 0.5)
    - **Mitigation**: If mobile limitations are severe, explore using lightweight detection models like TensorFlow Lite. Set up performance tests in this sprint to identify bottlenecks.

Sprint 5:

- **Risks**:
    - Height calculations may require a more complex algorithm than anticipated. (Prob: 0.5)
    - Modular design complexity might extend timelines. (Prob: 0.4)
    - **Mitigation**: Prioritize simple algorithms and add complexity only as needed. Break down modular design into manageable components and integrate one at a time.

Sprint 6:

- **Risks**:
    - Height detection requires more data points or higher resolution. (Prob: 0.5)
    - Prototype might need hardware not accessible or feasible on mobile devices (Prob: 0.3)
    - **Mitigation**: Consider using simpler relative height detection if absolute values are challenging. Explore OpenCV's scaling and resolution techniques.

Sprint 7:

- **Risks**:
    - Model training techniques could require extensive datasets not readily available. (Prob: 0.6)
    - Full-stack implementation might increase overhead and limit application speed. (Prob: 0.4)
    - **Mitigation**: Consider using pre-trained models to limit the computational needs of training a new model.

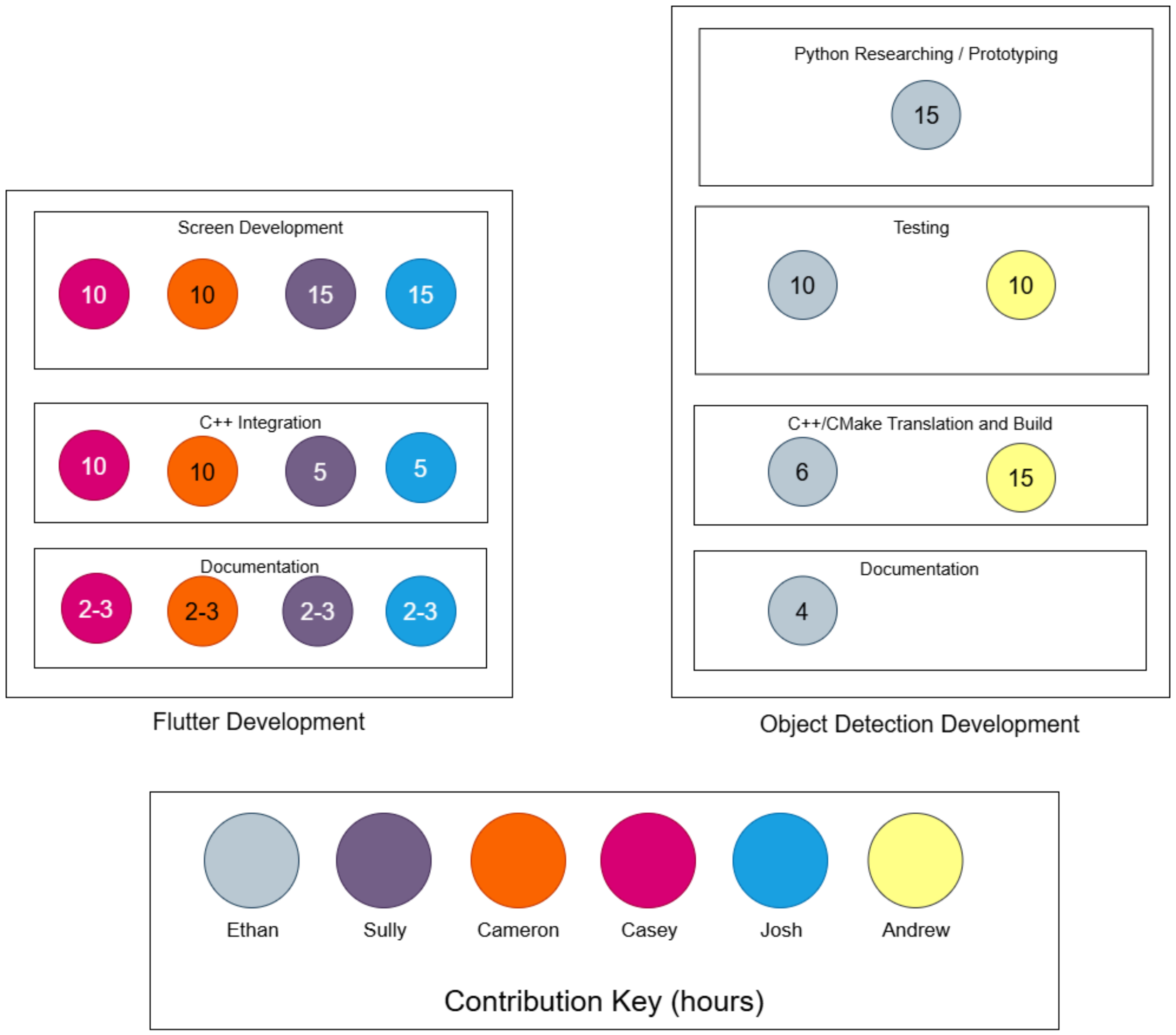
Sprint 8:

- **Risks:**
    - Detection algorithms may not generalize well across different test conditions. (Prob: 0.5)
    - **Mitigation**: Run multiple rounds of testing in varied lighting/angles and adjust thresholds. Update algorithms or training data based on observed issues.

Sprint 9:

- **Risks**:
    - Final app integration may encounter unexpected platform constraints, e.g., compatibility issues on different devices (Prob: 0.5)
    - **Mitigation**: Use platform-specific checks to handle potential discrepancies. Testing should include a variety of devices, screen sizes, and operating systems.

3.6 PERSONNEL EFFORT REQUIREMENTS



Flutter Development

- Screen Development
    - The creation of each screen/page of the mobile application.
    - Includes functionality of buttons, page transitions, and user data entry.
- C++ Integration
    - Connect the frontend display of Flutter to run the backend C++ scripts for object/height detection.
    - Must integrate the mobile sensors, cameras, and speakers that are used within the C++ scripts using Flutter plugins.
- Documentation
    - Clear documentation indicating each component within the application's codebase.

Object Detection Development

- Python Researching / Prototyping
    - Using OpenCV in Python, design and prototype strategies of data collection and identification.
    - Should be a model capable of being tested for accuracy and speed.
- Testing
    - Test both Python and C++ prototypes to determine if they are appropriate for final prototypes.
    - Testing should be done in multiple environments with multiple testing values.
    - Final prototype testing should have a smaller bound for error and more rigorous tests.
- C++ Translation / Build
    - Rewrite the Python object/height detection scripts to function in C++.
    - Create a CMake project to be built in a mobile application.
- Documentation
    - Write descriptions and identifications for each class and working component within the Python and C++ files for easy readability and professionalism.

3.7 OTHER RESOURCE REQUIREMENTS

Identify the other resources aside from financial (such as parts and materials) required to complete the project.

Physical Resources:

- Phone (Android or iOS)

Software Resources:

- Flutter Framework
- OpenCV Library
- YOLO or other Object Detection Model