Product Name: RePoste

Client: Terry Yoo

Team Name: Goobernauts

Team Member Names:

Stephen Goodridge, Logan Geiser, David Geng, Ryan Giles, Heath Miller

Date: 12/09/2024

RePoste Fencing Replay System

Critical Design Review Document

Table of Contents

1. Introduction

This capstone project is for the development of a video replay software system for the sport of fencing. This project is for our client, Dr. Terry Yoo, in partial fulfillment of the Computer Science BS degree for the University of Maine.

1.1 Project Overview

The goal of this project is to create an open-source video replay system for the sport of fencing. Video replay software used by USA fencing has become antiquated and only runs on Apple laptops that cannot be updated without causing issues for the software.

Our team aims to create an open-source software solution that will run on MacOS and Windows. By creating an open source project, we can offer a more accessible and cost effective solution than what is on the market. This project will also allow referees to continue professional development through video replay, ensuring that their calls are always as accurate as possible.

1.2 Purpose of This Document

The purpose of this Critical Design Review (CDR) document is to present and validate the RePoste system design and to ensure readiness for implementation. This document will seek approval from our client and all included members of the Goobernauts. This will verify both client and development team agree on project scope and specifications before continuing forward with development.

2. Project Design

This section details the system requirements–both functional and non-functional,–architecture, hardware and software utilization, and testing plan. The requirements and use cases remain the same from our Software Requirement Specification (SRS) document detailed in Appendix D. System architecture information and testing acceptance criteria have been transferred and updated from the Software Design Document (SDD) in Appendix D.

2.1. Requirements

The following subsections detail RePoste's complete list of functional and non-functional requirements along with their applicable use cases.

2.1.1. Functional Requirements

1. The system shall record real time video at 60 frames/second so that review can be accurate. (**USE CASE #1**)
    a. Acceptance Criteria: Program can capture high-definition video at a minimum of 60fps from camera input and display in a synchronized manner.
2. The system shall be able to replay specific moments immediately after they occur so the referee can make accurate judgements. (**USE CASE #1**)
    a. Acceptance Criteria: Program provides user interface that allows the referee to quickly replay recent moments, slow down playback, or pause the video.
3. The system shall record audio along with the video so the audio cues will be captured as well. (**USE CASE#2**)
    a. Acceptance Criteria: Program can capture audio from a connected microphone and synchronize it with the video recording in real time.
4. The system shall work on both MacOS and Windows, and eventually Linux, so it can be used regardless of operating system. (**USE CASE #4**)
    a. Acceptance Criteria: The program can be installed and run on MacOs and Windows platforms with identical functionality across all platforms.
5. The system shall save the recorded video and audio with event markers, so that it can be reviewed and shared after a match. (**USE CASE #3**)
    a. Acceptance Criteria: The program can export the recorded video and audio files into standard video format.
6. The system shall interface with an Arduino device to capture signals from fencing equipment and log those events during video capture. (**USE CASE #5**)
    a. Acceptance Criteria: The program can read signals sent from an Arduino device via USB and initiate the replay functionality.
7. The system shall interface with USB, USB-C, and HDMI inputs. (**USE CASE #6**)

    a. Acceptance Criteria: Program can receive signals and interface with USB, USB-C, or HDMI inputs.

8. The system shall change the replay speed so that the referee can better understand what actions occurred. (**USE CASE #7**)
    a. Acceptance Criteria: Program can slow down the replay at 10% intervals from 100% speed to 10% speed.
9. The system shall save recorded videos with a naming convention provided so the referee/official can find saved replays easier. (**USE CASE #8**)
    a. Acceptance Criteria: Program can save recorded video with a desired naming convention.
10. The system shall have the ability to function in fullscreen mode so the referee has the best view of the replay. (**USE CASE #9**)
    a. Acceptance Criteria: Program functions in a fullscreen mode without frame drops.

2.1.2. Non-Functional Requirements

1. The system must capture and process video and a minimum of 60 frames per second(FPS) and audio in real-time, without dropping frames or introducing noticeable latency.
    a. Priority: 5
    b. Metric: Video recording maintains at least 60FPS on systems with 8GB RAM.
2. The system must run on MacOS, Windows, and preferably Linux, using a single codebase with cross-platform support.
    a. Priority: 5
    b. Metric: System builds and runs successfully on all two/three platforms, with no major differences in functionality.
3. The system must use CPU, memory, and disk resources efficiently, ensuring it does not cause excessive strain on the user's machine, particularly during extended periods of use.
    a. Priority: 4
    b. Metric: CPU usage must remain below 50%, and memory usage below 4GB on a system with 8GB RAM during operation.
4. The system must process video, audio, and input signals from the Arduino device with minimal delay to ensure accurate real-time analysis
    a. Priority: 5
    b. The system must process each frame and audio input within 16.67 milliseconds(for 60 FPS), ensuring latency does not exceed 100 milliseconds.
5. The system must provide backup of video files to ensure data isn't lost.
    a. Priority: 4
    b. System will automatically save replays to a secondary storage location once the recording ends.

6. The system must put video files in a secure location to prevent unauthorized access.
    a. Priority: 5
    b. Files will be secured behind user authentication and logins that only authorized personnel should have access to.
7. The system must be created to ensure that it is easy to update and make modifications to, to ensure that new developers can update code.
    a. Priority: 3
    b. New developers should be able to update code easily, read the preexisting code, and make these changes without messing up previous functionalities.
8. The system must minimize the use of the system's CPU and GPU while idle.
    a. Priority: 3
    b. While software is idle, CPU and GPU should not go over 20-30% usage.
9. The system must have a simple, yet effective, UI to ensure the referees can use it with minimal training.
    a. Priority: 4
    b. New users/referees should be able to stop/start recording, initiate a replay, change speed of playback, etc. within a few minutes of familiarizing themselves with software.
10. The system must be able to handle high-definition video inputs without lowering performance.
    a. Priority: 5
    b. Software should be able to handle video up to 1080p resolution without a drop in frames.

2.1.3. Use Cases

Number	1	
Name	Capture and replay video	
Summary	Program shall capture video and be able to replay the video.	
Priority	5	
Preconditions	Must have a trigger for when a hit/foul is called.	
Postconditions	Video should immediately be able to be watched for review by the referee.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Arduino USB signal, Click on spacebar by video ref	
Main Scenario	Step	Action
	1	Referee calls hit or foul
	2	Video referee hits trigger
	3	Program captures video of what was happening before the hit or foul
	4	Referee watches playback
	5	Allow options for multiple speeds for video

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● Building tests ● Interface with SFS-Link	

Number	2	
Name	Capture Audio	
Summary	Program shall capture audio alongside with video	
Priority	5	
Preconditions	Must have laptop microphone	
Postconditions	Audio should be recorded and sync with video	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Video recording in progress	
Main Scenario	Step	Action
	1	Referee starts recording
	2	Microphone records audio
	3	System matches audio to video timestamps
	4	Referee watches playback with audio

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● Building tests ● Quality of microphone ● Managing memory usage on 8GB laptop	

Number	3	
Name	Save recorded video and audio to be shared/reviewed	
Summary	Program shall save recordings of the match to standard video format.	
Priority	4	
Preconditions	Video should be recorded and saved in the system	
Postconditions	Referee can share that video	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Referee uses the 'Play/Pause" button (Spacebar)	
Main Scenario	Step	Action
	1	Referee presses "Play/Pause" button to start recording
	2	Video of match is recorded
	3	Arduino signal is detected
	4	Video recording is halted
	5	Referee is presented 2-4 second playback
	6	After Review referee presses "Play/Pause" button to start recording

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● Building tests ● Managing memory on 8GB laptop	

Number	4	
Name	Operating System compatibility	
Summary	Program shall be able to be installed on both MacOS and Windows using the same source code.	
Priority	5	
Preconditions	Must have laptop with one of the two operating systems	
Postconditions	Program shall install and function the same regardless of OS.	
Primary Actor	Official Referee	
Secondary Actors	Video Referee	
Trigger	Installation	
Main Scenario	Step	Action
	1	Official chooses to use Reposte
	2	Official installs the program on their desired OS
	3	Program installation is successful
	4	Program runs the same on both OS

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● Build tests ● How do we make installation easy and accessible? ○ Installation manual	

Number	5	
Name	Interface with Arduino	
Summary	Program shall interface with an Arduino to receive a signal to initiate replay	
Priority	5	
Preconditions	Must have a trigger for when a hit/foul is called.	
Postconditions	Signal from the Arduino should be taken by the program and initiate replay.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Arduino USB signal.	
Main Scenario	Step	Action
	1	Fencing equipment sends signal to Arduino
	2	Arduino receives signal and transmits it via USB
	3	Program interprets signal then initiates replay functionality

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● Building tests ● Knowledge acquisition on microcontroller programming	

Number	6	
Name	Interface with USB, USB-C, and HDMI	
Summary	Program can receive signals and interface with USB, USB-C, and HDMI.	
Priority	5	
Preconditions	USB, USB-C, or HDMI device is plugged into laptop	
Postconditions	Program receives signals through those respective ports.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	USB, USB-C, or HDMI input	
Main Scenario	Step	Action
	1	Referee plugs in necessary devices
	2	System receives signal from device
	3	System interfaces with device

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● Building tests ● Memory management ● What products to support? ○ USB/USB-C: Microphone, Additional Cameras? ○ HDMI: Monitor support? Max resolution?	

Number	7	
Name	Adjustable replay speed	
Summary	Program can slow down replay speed at 10% intervals(10%-100% speeds)	
Priority	5	
Preconditions	Must have a trigger for when a hit/foul is called.	
Postconditions	Replay video will have adjustable speeds.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Arduino USB signal, Click on spacebar by video ref	
Main Scenario	Step	Action
	1	Replay function is triggered
	2	Replay footage is presented
	3	Replay footage offers adjustable playback speeds
	4	Referee watches playback at desired speeds

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● Building tests ● Maintaining image quality at reduced speeds ● Audio during slower replay speeds	

Number	8	
Name	Custom naming conventions	
Summary	Program can save recorded video with a desired naming convention	
Priority	3	
Preconditions	User runs software.	
Postconditions	Video should save in desired naming conventions.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger		
Main Scenario	Step	Action
	1	User runs program
	2	User is prompted to enter save file name
	3	User enters desired save file name
	4	Program saves recording file with given name

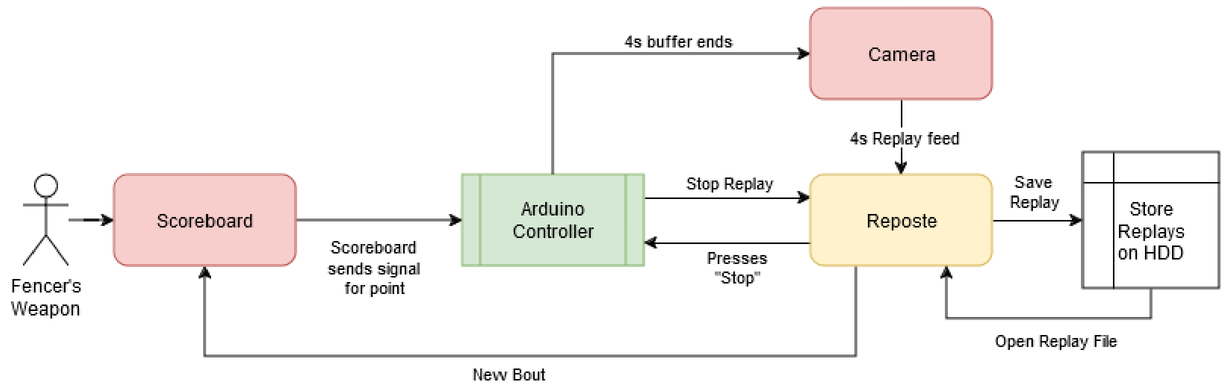
Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● UI design	

Number	9	
Name	Full Screen mode	
Summary	Program can run in full screen	
Priority	4	
Preconditions	Program and dependencies installed.	
Postconditions	Program runs in full screen mode	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Program ran	
Main Scenario	Step	Action
	1	Official opens program
	2	Program runs in full screen mode
	3	Referee can view replay system in full screen

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● Resolutions constraints ● Memory management	

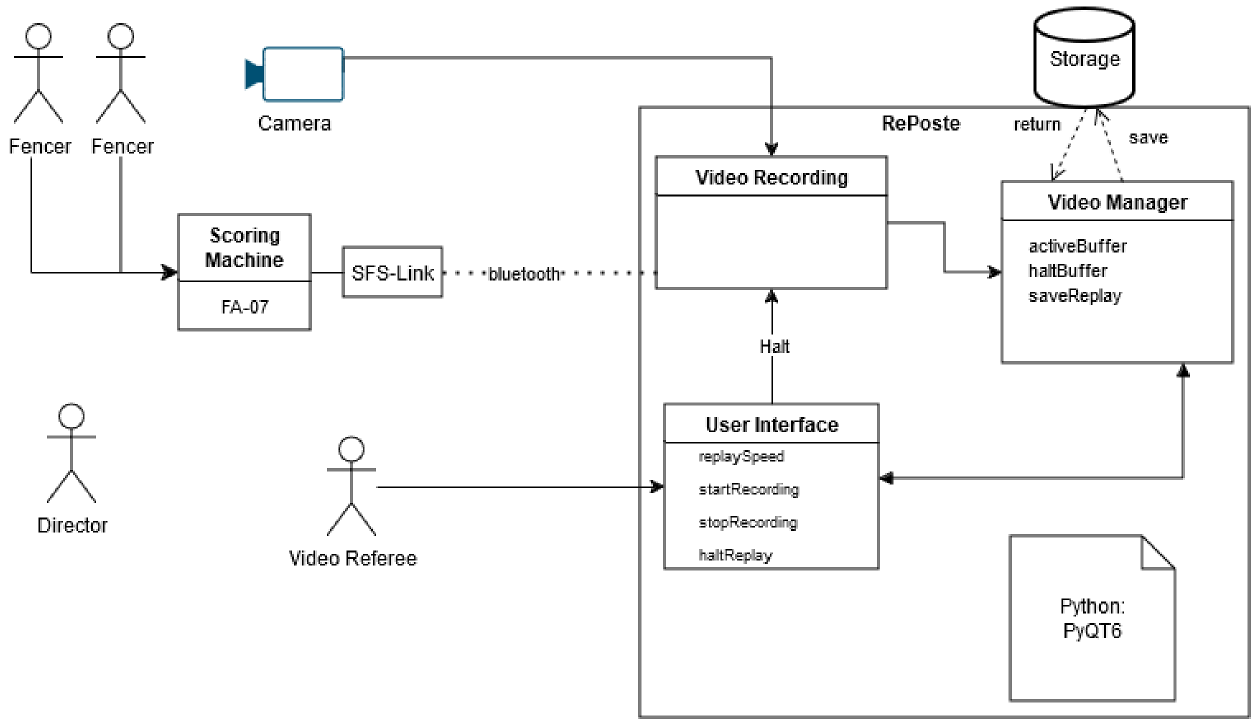
2.2. Architecture

2.2.1. Event-Driven Architecture



The diagram above showcases an event-driven architecture. The triggering event is when a fencer's weapon triggers the scoreboard, or when manually triggered by a referee through a keyboard input or the GUI. When the event occurs, the scoreboard signals the Arduino controller, which sends a signal to both the camera and the system to stop recording and save the last four seconds in the buffer. The four second video is then visible for review in the GUI and then saved on the device's selected hard drive. The files saved are able to be reviewed at a later time through GUI.

2.2.2. System Decomposition



(*Appendix D- (5d)*)

The diagram shows the decomposition diagram of the system and focuses on how each component interacts in order to manage the video playback.

Each one of the components is representing a part of the system. There are users such as the Director, Fencer and Video Referee. Director can signal to the video referee to initiate a replay. Fencers and Video Referees are interacting with the RePoste System more directly. Fencers are connected to a scoring machine(FA-07). The scoring machine receives this signal from the weapons, once receiving the signal from the fencing equipment, it will send an input to the SFS-Link to trigger the replay. This is done through a bluetooth signal to the RePoste system. Once the system receives that signal, a replay will be provided and saved for later viewing. The system will use two ring-buffers in the video manager section. One Active Buffer and one Halt Buffer. A continuous stream of video input will be sent to an active buffer, only saving the desired replay time into memory(eg -4 seconds). On trigger of replay, the halt buffer will reference the active buffer to store the replay footage for viewing. This will allow the video recording to continue in the active buffer and only use the halt buffer when replay is activated.

2.3. Hardware/Software

Our system utilizes certain hardware components. It will utilize an external camera, a Favero FA-07 scoring machine, and the SFS-Link; a microcontroller built to interface with the fencing scoring machine. This microcontroller will detect when fencers score a point.

It also utilizes an external camera that is required to be connected to the device running RePoste. SFS-Link manual*(Appendix D- (4d))*

For software, our project will utilize CMake and Imageio for development, ensuring its usability across operating systems and camera usage.

2.4. Testing

As RePoste utilizes external hardware, we will be conducting both system and acceptance testing before our final deadline. The software itself will be unit and integration tested using PyTest for the Python portion, and CTest for Arduino integration. The rest of this section details the system and acceptance test criteria.

1. **Capture and Replay Video**

    Test Case 1.1: Verify system captures video at 60fps

    Steps:

    1: Start recording
    2: Verify the video playing is 60 fps
    3: Trigger replay system
    4: Ensure video replay never falls below 60fps

    Result: Video replay stays above 60fps

    Test Case 1.2: Verify user can replay video immediately after capturing

    Steps:

    1: Start recording
    2:Trigger replay system
    2: Verify video replay is instant

    Result: Video replay is immediate once triggered

2. **Capture Audio**

    Test Case 2.1: Record audio with video

    Steps:

    1: Start recording
    2: Trigger replay system
    2: Audio is playing with video

    Result: Audio is playing with video

    Test Case 2.2: Sync audio with video

    Steps:

    1: Start recording
    2.Trigger replay system
    3: Audio is playing with video
    4: Verify audio is in sync with video

Result: Audio is synced with video and aligns with video cues

3. **Save Recorded Video and Audio to be Shared/Reviewed**

Test Case 3.1: Program should save recordings of the match to standard video format

Steps:

1: Start recording
2: Trigger replay system
3: Verify audio and video works and is in sync
4: Click the save button
5: Verify file is saved to a directory in standard video format
6: Ensure video and audio plays and are sync

Result: Video replays are saved to a directory and can be played for review or shared.

Test Case 3.2: Program should display event markers of saved video locations

Steps:

1:Start recording
2: Trigger replay system
3: Verify video and audio play
4: Click the save button
5: Check file directory for file containing event markers of when replay occurred

Result: Files display event markers for when replay happened. This promotes easier search for specific replays

4. **Operating System Compatibility**

Test Case 4.1: Verify program can be installed on MacOS

Steps:

1: Acquire installation for MacOS
2: Install program
3: Open application and select webcam and microphone
4: Play/Stop video
5: Trigger replay system
6: Verify file is saved to a directory in standard video format
7: Ensure video and audio play and are in sync

Result: Program installs and works as intended on MacOS

Test Case 4.2: Verify program can be installed on Windows

Steps:

1: Acquire installation for Windows
2: Install program
3: Open application and select webcam and microphone

4: Play/Stop video
5: Trigger replay system
6: Verify file is saved to a directory in standard video format
7: ensure video and audio play and are in sync

Result: Program installs and works as intended on Windows

(Optional) Test Case 4.3: Verify program can be installed on Linux

Steps:

1: Acquire installation for Linux
2: Install program
3: Open application and select webcam and microphone
4: Play/Stop video
5: Trigger replay system
6: Verify file is saved to a directory in standard video format
7: ensure video and audio play and are in sync

Result: Program installs and works as intended on Linux

5. **Interface with Arduino**

Test Case 5.1: Verify program receives signal from Arduino to initiate replay

Steps:

1: Verify Arduino is hooked up to computer and fencing equipment
2: Test Arduino signals
3: Start program
4: Verify Arduino triggers replay system by referee stop, hit, or foul

Result: Arduino sends replay signal when referee needs review

6. **Interface with USB, USB-C, and HDMI**

Test Case 6.1: Program receives signals and interfaces with USB, USB-C, and HDMI

Steps:

1: Verify USB, USB-C, or HDMI are properly connected
2: Test Arduino signals
3: Start Program
4: Verify system receives signal from device
5: Verify system interfaces with device

Result: USB, USB-C, and HDMI are all viable sources to receive signal

7. **Adjustable Replay System**

Test Case 7.1: Program can slow down video that varies from 10% - 100%

Steps:

1: Start recording
2: Trigger replay system

3: Select correct recording file from the directory

4: Replay offers adjustable speeds with corresponding number keys for video speed. (i.e. 1 = 10%, 2 = 20%, etc.)

Result: Video files have varying slow motion speeds for accurate hit/foul detection

8. **Custom Naming Convention**

Test Case 8.1: Each replay saved has a custom naming convention for organization

Steps:

1: Start recording

2: Prompt displays to enter save file name

3: Type desired name

4: Program saves each recording with that naming convention

Result: Each saved file is named to correspond with the match for organization purposes and easier to find within the directory

Test Case 8.2: User enters naming convention already in use in directory

Steps:

1: Start recording

2: Prompt displays to enter save file name

3: Type desired name

4: Program prompts user to enter new name for saved file

Result: Program prompts user to enter a new naming convention as the one entered is already taken

9. **Full Screen Mode**

Test Case 9.1: Program can run full screen and windowed

Steps:

1: Start program

2: Select a window button to bring program in or out of full screen

3: Trigger replay system

4: Select video recording for replay in directory

5: Select a window button to bring replay in or out of full screen

Result: Program runs smoothly in full screen or windowed mode and seamlessly transitions between the two.

Test Case 9.2: Program can run at different resolutions

Steps:

1: Start program in computers native resolution

2: Start recording

3: Trigger replay system

4: Select video replay in the directory for playback

5: Ensure 60 fps is consistent at that resolution

6: Apply same test at a range of resolutions (800x600 - 4k)
Result: Video recording and playback stay at or above 60 fps at any resolution

3. DevOps

This section details the practices to integrate automated testing into our infrastructure to shorten the development lifecycle, enhance the ability to deliver updates and features continuously, and improve the quality of the software. For this project, DevOps will be essential to our software deployment process, automating validity of code and stability, which ensures smooth integration of various components, and maintaining consistency across development, testing, and production environments.

3.1. CI/CD

The following subsections detail RePoste's DevOps lifecycle and the processes involved in ensuring a smooth, efficient, and continuous delivery pipeline for the project through GitHub.

3.1.1. Continuous Integration

Integration of code from multiple developers is posted to a shared repository on GitHub. This phase ensures that new code changes are integrated smoothly, preventing conflicts and lowering the risk for errors.

- Developers commit code to GitHub
- Automated builds are triggered and test committed code
- Tools such as Flake8 linter, and Black formatter are run to ensure developers are meeting code standards
- Feedback is provided if tests pass or fail.

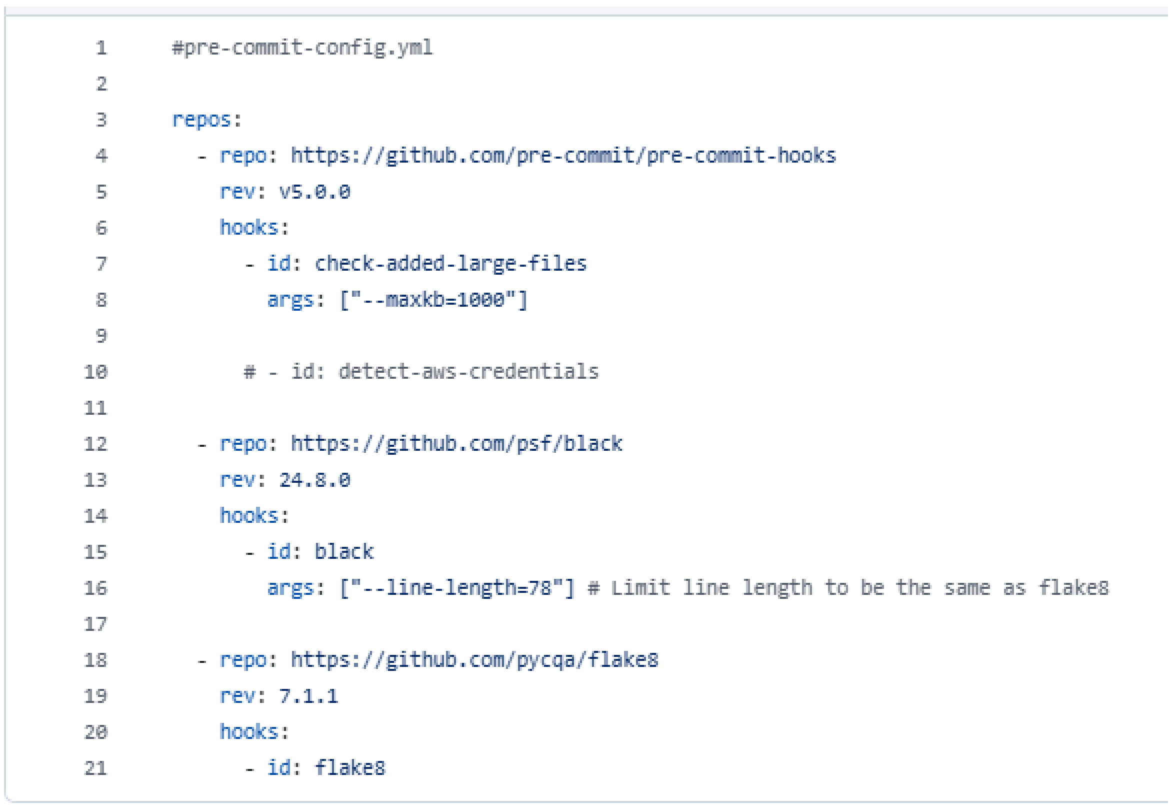
3.1.2. Continuous Deployment

After code is successfully integrated, automatically deploy different environments to ensure the software can be reliably released.

- RePoste is automatically staged to a test environment.
- A series of tests such as acceptance tests, unit testing, and performance testing.
- Continuous monitoring to ensure RePoste performs as expected during production.

3.1.3. GitHub Integration

RePoste's continuous integration .yaml file on GitHub. These checks are run on any commit to the GitHub repository.



```yaml
#pre-commit-config.yml

repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v5.0.0
    hooks:
      - id: check-added-large-files
        args: ["--maxkb=1000"]

      # - id: detect-aws-credentials

  - repo: https://github.com/psf/black
    rev: 24.8.0
    hooks:
      - id: black
        args: ["--line-length=78"] # Limit line length to be the same as flake8

  - repo: https://github.com/pycqa/flake8
    rev: 7.1.1
    hooks:
      - id: flake8
```

3.2. Lifecycle Development Plan

RePoste follows an Agile approach, focusing on flexibility and iterative development to adapt to requirements quickly. Each phase, from planning to deployment is broken down into sprints, ensuring each feature is developed fully, tested, and delivered in increments. This approach allows for rapid development as well as rapid testing to deliver a high quality product.

3.2.1. SDLC Breakdown

1. **Analysis**: Projects requirements and objectives are gathered, ensuring clear understanding of client needs and system requirements.
2. **Design**: Detailed architecture design and UI design mockups that correlate with requirements.
3. **Development**: Creating codebase for RePoste with core functionalities, design, and coding standards for robustness and scalability.
4. **Testing**: Manual and automated tests to verify system functionality, as well as identify bugs. This will include unit testing, integration testing, and system testing.
5. **Deployment**: Releasing the system to the client and end user. This will include ease of access and production ready.
6. **Maintenance**: Updating RePoste with any post launch issues, client feedback, or adding new features for long term stability. Also the point where we can hand off the project to the client if requested.

4. User Interface Design

This section will go into detail on our user interface design and the different features we will have implemented in our design to help give users a clear and understandable user interface. We will look into the buttons we have appearing on our software as well as the keybinds to help users manipulate the software quickly. On top of this we will also be looking at the key functionalities of the software and the different screens that the user will be looking at.

4.1. User Interface Standards

4.1.1. Accessibility

Vision disabilities: Color Blindness

-Use color ADA compliant color combinations and contrast

-Black and White

Cognitive disabilities: Dyslexia

-Utilize sans serif typeface

-Implement proper color contrast

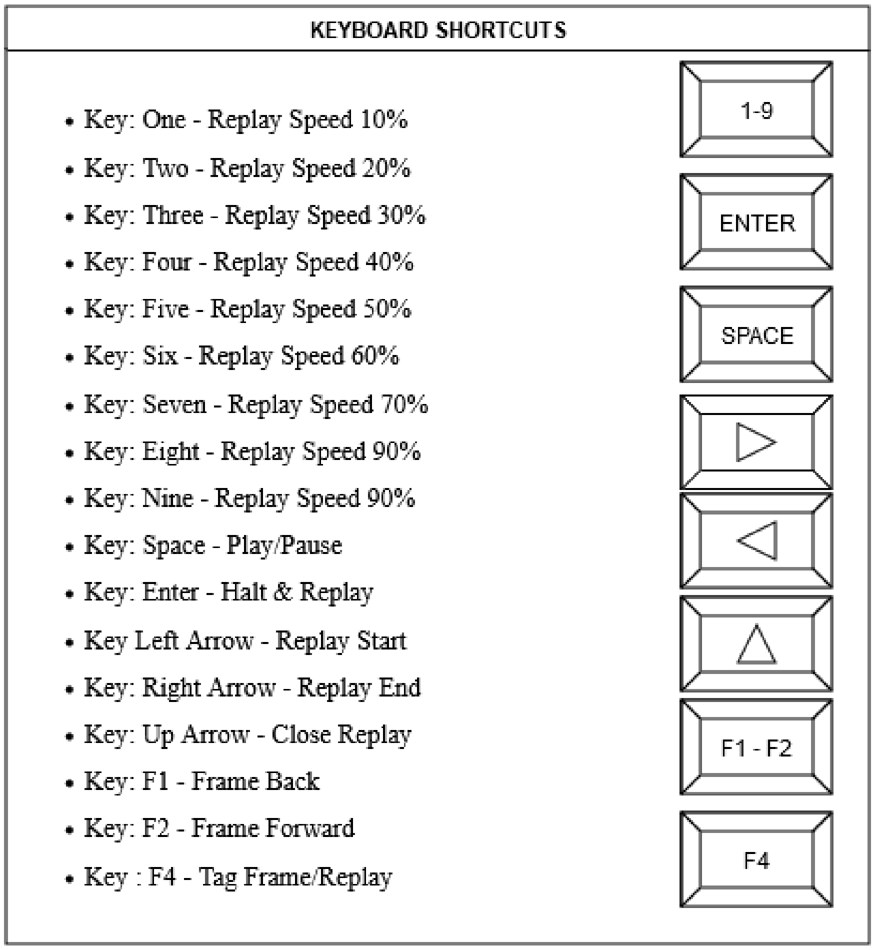
Hearing disabilities: Deafness, low-hearing

-Toggle audio

-Visual Cues

Motor problems: Loss of limb permanently, temporarily, or situationally.

-Full keyboard navigation


1-9
ENTER
SPACE
F1 - F2
F4

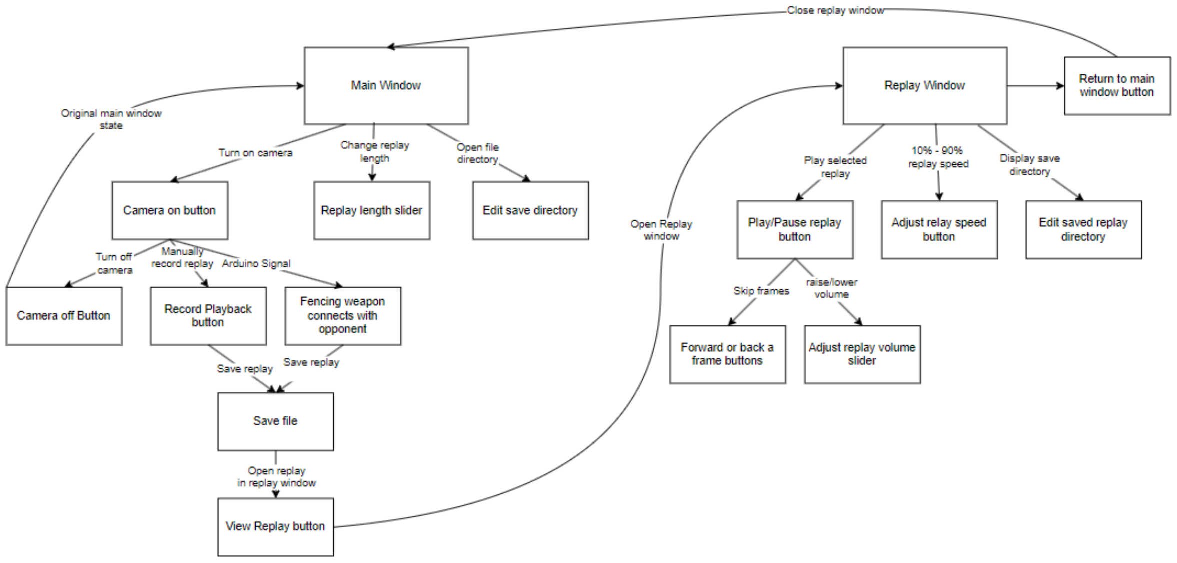
KEYBOARD SHORTCUTS

- Key: One - Replay Speed 10%
- Key: Two - Replay Speed 20%
- Key: Three - Replay Speed 30%
- Key: Four - Replay Speed 40%
- Key: Five - Replay Speed 50%
- Key: Six - Replay Speed 60%
- Key: Seven - Replay Speed 70%
- Key: Eight - Replay Speed 90%
- Key: Nine - Replay Speed 90%
- Key: Space - Play/Pause
- Key: Enter - Halt & Replay
- Key Left Arrow - Replay Start
- Key: Right Arrow - Replay End
- Key: Up Arrow - Close Replay
- Key: F1 - Frame Back
- Key: F2 - Frame Forward
- Key : F4 - Tag Frame/Replay

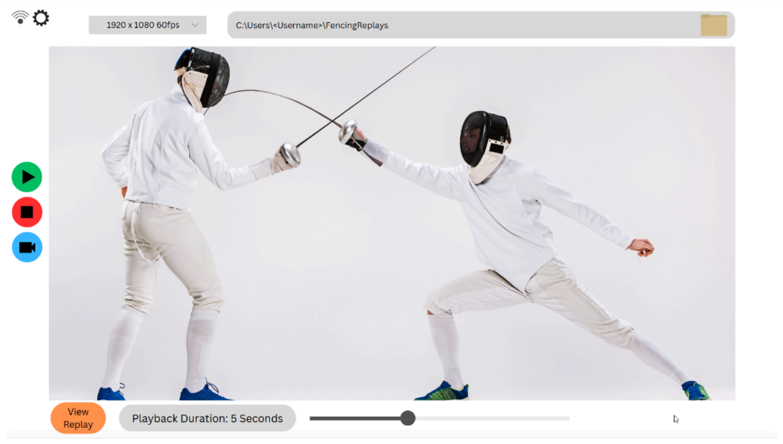
4.2. User Interface Walkthrough

4.2.1. Navigation Diagram

The following is the navigation diagram for the Reposte system. Primarily focusing on what happens between both our “main window” and our “replay window” and the functions of each different feature within our software and how it interacts with one another.



4.2.2. User Interface


1920 x 1080 60fps

C:\Users\<Username>\FencingReplays

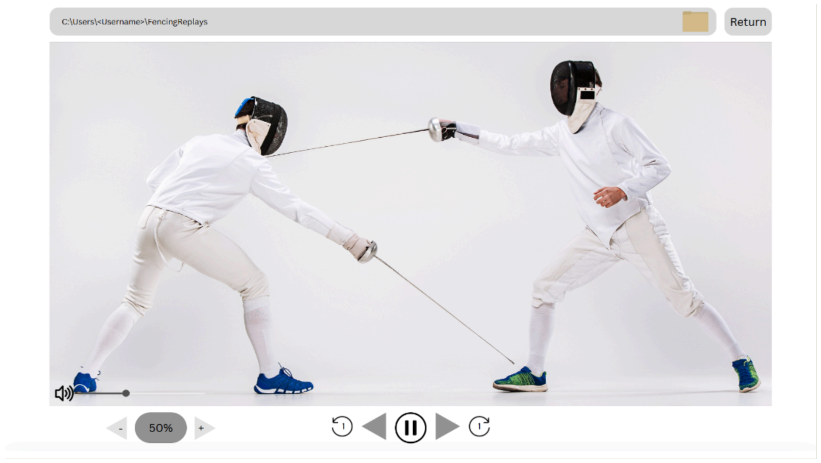
Playback Duration: 5 Seconds

Here is the first screen of the software. Here we can see multiple buttons for the user to interact with.

- To begin, we would like to preface that these buttons are all a part of a pull out menu to give the user an option to have a larger display and rely primarily on the keybinds feature of the app.
- In the top left hand corner you can see a cog representing the settings menu, in the settings menu there will be options for the user to map the keys on their keyboard to different keys if they choose they do not like the preset keybinds.
- Above the play window, there are options to change the resolutions and frame per second(FPS) of the video player. Next to that, there is the folder for which the user wishes to save videos and watch previous videos.
- On the left hand of the play window, we can see 3 buttons. These buttons go as follow:

- The green play button:
  - Ensures the camera is on and ready to record. This will be set to “P” on the keyboard.
- The red stop button:
  - This will turn the camera off/pause the recording. This will also be set to “P” on the keyboard
- The blue camera button:
  - This button ensures that the video is being saved and will replay the amount of time set in the playback duration. To record the last set amount of seconds the user can click space on the keyboard.

- Underneath the play window we also have a few more features.The following features are:
  - The orange view replay button:
    - This button brings up the second window where the user will be able to view the replays to rewatch a bout between two fighters.
  - The playback duration slider:
    - This slider gives the user the option to choose how long they want the replay to be in seconds. Right now the duration goes from 1 to 10.



This is the second screen of the software. On this screen users are able to view the replays of previous bouts to ensure that a hit was accurate. There are a few features on this screen that are available to the users, those features are as follows:

- Underneath the player window there are a few buttons.
  - To the far left, we have the playback speed option.
    - This feature gives the users the option to slow down with the “-” button and speed up with the “+” button. This feature will by default be mapped to the 1 to 0 keys on the keyboard and respectively each number will put the player to that number times 10 with the exception of 0 (1 = 10, 2=20, 3=30… 0=100).
- In the middle is the video player options.
  - Directly in the middle is the pause and play button:

        - These will pause and play the video. This keybind will be mapped to the space key.
    - Next to the pause button there are two arrows:
        - The left arrow will reset the clip. And if clicked again when the video is at 0 seconds. Will go to the previous video in the directory. This will be mapped to the left arrow key on the keyboard.
        - The right arrow will go to the next video in the directory. This will be mapped to the right arrow key on the keyboard.
    - On the far left and right there are partial circles with a 1 in the middle.
        - The button on the left will push the video back 1 frame. This will be set to the F1 key on the keyboard.
        - The button on the right will push the video ahead 1 frame. This will be mapped to the F2 key on the keyboard.
    - In the top right hand corner there is a "Return" button.
        - This button will send you back to the 1st window.

4.3. Data Validation

This section contains a description of data items that the system accepts from the user, the definition is broken down into the basic data type, limitations, and allowed formats the system accepts.

Name	Data Type	Limitation	Format
Video File	MP4	Must be MP4, if error reject and reprompt	MP4
Frame Image	NumPy Array	If values are missing, default color values to (0,0,0)	Dependent on Camera
Arduino Signal [ESP32] (Planned)	Boolean	Must be true or false, if error default to false	True/False
Video File Name (Planned)	String	Must follow OS naming limits, if error reject name and reprompt	videofilename.mp4
Save Location (Planned)	String	Must be a valid location, if error reject location and reprompt	C:\Users\Username\...

5. Timeline

This section details the success criteria we aim to achieve and the methods we plan to test them with, risks and our assessment of them in the usage and development of the project, and future plans going forward into the spring semester with a timeline of expected events and development. The risk assessment matrix is taken from our Risk Assessment document and the Success Criteria are adapted from the Requirement Matrix from the SDD and the Testing section from the SRS.

5.1. Success Criteria

This section covers the success criteria we are using to measure the completion of each step of the project. It also defines when and how we decide if something is considered successful within the project. The criteria numbers refer to each of the Functional Requirements in the table below

Criteria:

1. The system maintains a consistent recording frame rate of at least 60 frames per second during operation under standard conditions.
2. The system is able to replay recent events in the buffer when the scoreboard sends a trigger
3. The system is able to run on the latest versions of MacOS and Windows
4. The system is able to accurately record audio synced with the video
5. The system is able to save the recorded video with audio and event markers to the local system
6. The system successfully captures signals from fencing equipment by interfacing with a microcontroller
7. The system is able to interface with USB, USB-C, and HDMI inputs
8. The system is able to modify playback speeds by increments of 10%
9. The system will save videos using a naming convention
10. The system supports full-screen playback mode

Functional Requirements	System Component
1) System shall record real time video at 60 frames/second so that review can be accurate. (**USE CASE #1**)	int fps60_check() function
2) System shall be able to replay specific moments immediately after they occur so the referee can make accurate judgements. (**USE CASE #1**)	void replay() function
3) System shall record audio so audio cues	audio_recording method

will be captured as well. (**USE CASE#2**)	
4) System shall work on both MacOS and Windows, and eventually Linux, so it can be used regardless of operating system (**USE CASE #4**)	GitHub Action Build: Runs-on: [ubuntu-latest, macos-latest, windows-latest]
5) System shall save the recorded video and audio with event markers, so that it can be reviewed and shared after a match. (**USE CASE #3**)	void save_file() void event_marker()
6) System shall interface with an Arduino device to capture signals from fencing equipment and log those events during video capture. (**USE CASE #5**)	Boolean signal_receive() function Void log_event() function
7) System shall interface with USB, USB-C, and HDMI inputs. (**USE CASE #6**)	boolean received_input()
8) System shall change the replay speed so that the referee can better understand what actions occurred. (**USE CASE #7**)	Replay speed can be modified (Default 100%): float replay_speed = 1; change_replay_speed() method (modifies replay by +/- .1)
9) System shall save recorded videos with a naming convention provided so the referee/official can find saved replays easier. (**USE CASE #8**)	User will input a string before the match: string bout_name = input; save_bout(bout_name, file) function
10) System shall have the ability to function in fullscreen mode so the referee has the best view of the replay. (**USE CASE #9**)	aspect_ratio() function

5.2. Risk Assessment

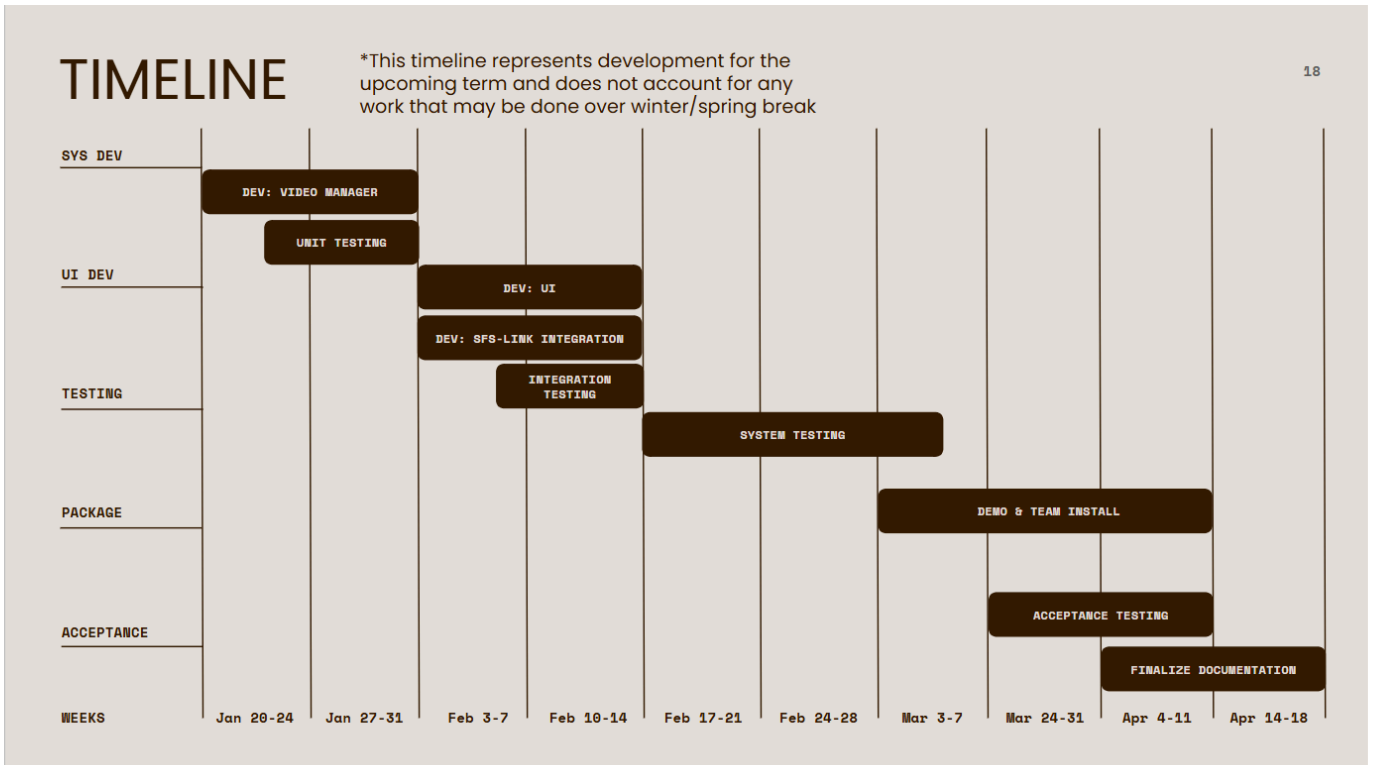
This section outlines the possible risks we may face during developing or testing the equipment along with our assessment of the risks possibility and severity. A risk matrix is included below

	Likelihood	**Consequence**	**Mitigation**
Cost	1	1	Avoid breaking given materials
Time/Schedule	3	3	Balance and communicate schedules
Technical	3	3	Work with up to date supported resources
Safety	1	4	Monitor Equipment for heat issues
Security	2	3	Ensure participants names are protected with code or substitute
Resources	3	4	Perform storage space check before using application and for dropped frames.

As shown we predict time, technical, and resources will likely be our biggest concerns due to balancing heavy course loads. It's likely we will face issues with scheduling, meetings, and syncing work schedules. Additionally technical issues are likely as we are working with multiple sources of hardware such as the scoring system, an Arduino, and likely a camera which require that we work with several resources to implement well. Additionally resources are a direct risk as the project aims to accommodate clubs and competitions some of which lack modern systems or equipment, so to ensure its usability we aim to limit how much system resources the application uses, meaning we need to be aware of the efficiency of our system while designing and testing.

5.3. Going Forward

This section outlines the plans moving forward and what steps and goals we plan to start and finish in the future to accomplish the success criteria.



As outlined we plan to develop the video manager and begin unit testing during Jan 20-Jan 31, then we plan to move on to developing the UI and integrating the SFS-Link microcontroller, during that time period we plan to also begin integration testing. System testing will begin on Feb 17 continuing to the week of March 3-7 during that time period we plan to test and deo the system with the fencing team here and initiate acceptance testing. During the final two weeks we plan to finish up acceptance testing and finalize our documentation of the system.

Appendix A – Agreement Between Customer and Contractor

This agreement confirms that the customer and the development team have reviewed and approved the contents of thisCritical Design Review Document (CDRD). By signing this document, both parties agree to the project scope, specifications, and responsibilities as outlined. The team commits to delivering the software according to the requirements, timeline, and quality standards defined. The customer commits to supporting the team by providing necessary feedback and resources in a timely manner.

In the event that future changes to the document are required, both the customer and the development team agree to follow a structured change management process. This process will include documenting proposed changes, assessing their impact on the project timeline and resources, and obtaining mutual written consent from both parties before any adjustments are made to the original agreement.

Team Member Name	Signature	Date	Comments
Stephen Goodridge		12/09/2024	
David Geng		12/09/2024	
Heath Miller		12/09/2024	
Ryan Giles	Ryan Giles	12/09/2024	
Logan Geiser		12/09/2024	

Client Name	Signature	Date	Comments
Terry Yoo			

Appendix B – Team Review Sign-off

All members of the team have thoroughly reviewed this Critical Design Review Document (CDRD) and agree on its content and project timeline. Each team member has provided their signature and the date below as a confirmation of their agreement. The comment area is available for any minor points that team members may want to note. There are no major points of contention, and all team members are aligned with the approach outlined in this document.

Team Member Name	Signature	Date	Comments
Stephen Goodridge		12/09/2024	
David Geng		12/09/2024	
Heath Miller		12/09/2024	
Ryan Giles	Ryan Giles	12/09/2024	
Logan Geiser		12/09/2024	

Appendix C – References

1. Yoo, Terry (2024). "Video Referee Replay System Software for Sport Fencing Competition" Project Proposal, University of Maine.
2. RePoste System Requirements Document (Version 1.0), Goobernauts, 2024
3. RePoste System Design Document (Version 1.0), Goobernauts, 2024
4. RePoste User Interface Design Document(Version 1.0), Goobernauts, 2024
5. GitHub. (n.d.). CMake quality GitHub action. https://github.com/marketplace/actions/cmake-quality
6. Imageio. (n.d.). Imageio(2.36.1). https://pypi.org/project/imageio/
7. Kitware, Inc. (n.d.). Documentation overview. CMake. https://cmake.org/documentation/
8. Kitware, Inc. (n.d.). Mastering CMake. CMake https://cmake.org/cmake/help/book/mastering-cmake/
9. Kitware, Inc. (n.d.). Welcome to CMake. CMake. https://cmake.org/
10. OpenCV. (n.d.). OpenCV: Open source computer vision library. https://opencv.org/
11. Riverbank Computing.(n.d.). PyQt6(6.7.1). https://pypi.org/project/PyQt6/
12. U.S. General Services Administration. (n.d.). Color overview: Design tokens. U.S. Web Design System. https://designsystem.digital.gov/design-tokens/color/overview/
13. U.S. General Services Administration. (n.d.). UX design: Accessibility for Teams. https://digital.gov/guides/accessibility-for-teams/ux-design/#content-start
14. Super Fencing System. (n.d.). Welcome to Super Fencing System. https://superfencingsystem.com/
15. Koov Broadcasting. (n.d.). Referee video replay user manual. http://escrime.koovbroadcasting.com/download/RefereeVideoReplayUserManual.pdf
16. Fédération Internationale d'Escrime. (n.d.). Handbook of specifications for video refereeing. https://static.fie.org/uploads/3/18851-Handbook_of_specification_Video%20Refereeing.pdf
17. Super Fencing System. (n.d.). SFS-Link manual (Version 1.1). https://superfencingsystem.com/SFS-Link_Manual_V1.1.pdf
18. panda, sonali. (2023, March 11). *SDLC (Software Development Life Cycle)Phases, Process. What is SDLC*. Numpy Ninja. https://www.numpyninja.com/post/sdlc-software-development-life-cycle-phases-process-what-is-sdlc

Appendix D – Documentation

(1d)- System Requirements Specifications (SRS)

1. Introduction

This capstone project is for the development of a video referee replay system software, in partial fulfillment of the Computer Science BS degree for the University of Maine. The goal of this project is to program an open-source video replay software for fencing referees, and should also work with operating systems such as MacOSX, Windows, and the possibility of Linux. The project will provide replay capabilities to fencing clubs, and improve the competitive experience.

1.1 Purpose of This Document

The purpose of this document is to outline the system requirements for a video referee replay system. It is intended to be read by the development team and project sponsors. This document gives an overview of the functional requirements, non-functional requirements, user interface guidelines, and testing criteria.

1.2. References

Yoo, Terry (2024). "Video Referee Replay System Software for Sport Fencing Competition" Project Proposal, University of Maine

Functional Requirements

1. System shall record real time video at 60 frames/second so that review can be accurate. (**USE CASE #1**)
    a. Acceptance Criteria: Program can capture high-definition video at a minimum of 60fps from camera input and display in a synchronized manner.
2. System shall be able to replay specific moments immediately after they occur so the referee can make accurate judgements. (**USE CASE #1**)
    a. Acceptance Criteria: Program provides user interface that allows the referee to quickly replay recent moments, slow down playback, or pause the video.
3. System shall record audio along with the video so the audio cues will be captured as well. (****USE CASE**#2**)
    a. Acceptance Criteria: Program can capture audio from a connected microphone and synchronize it with the video recording in real time.
4. System shall work on both MacOS and Windows, and eventually Linux, so it can be used regardless of operating system. (**USE CASE #4**)
    a. Acceptance Criteria: The program can be installed and run on MacOs and Windows platforms with identical functionality across all platforms.
5. System shall save the recorded video and audio with event markers, so that it can be reviewed and shared after a match. (**USE CASE #3**)
    a. Acceptance Criteria: The program can export the recorded video and audio files into standard video format.
6. System shall interface with an Arduino device to capture signals from fencing equipment and log those events during video capture. (**USE CASE #5**)
    a. Acceptance Criteria: The program can read signals sent from an Arduino device via USB and

initiate the replay functionality.

7. System shall interface with USB, USB-C, and HDMI inputs. (**USE CASE #6**)
    a. Acceptance Criteria: Program can receive signals and interface with USB, USB-C, or HDMI inputs.
8. System shall change the replay speed so that the referee can better understand what actions occurred. (**USE CASE #7**)
    a. Acceptance Criteria: Program can slow down the replay at 10% intervals from 100% speed to 10% speed.
9. System shall save recorded videos with a naming convention provided so the referee/official can find saved replays easier. (**USE CASE #8**)
    a. Acceptance Criteria: Program can save recorded video with a desired naming convention.
10. System shall have the ability to function in fullscreen mode so the referee has the best view of the replay. (**USE CASE #9**)
    a. Acceptance Criteria: Program functions in a fullscreen mode without frame drops.

Use Cases

Number	1	
Name	Capture and replay video	
Summary	Program shall capture video and be able to replay the video.	
Priority	5	
Preconditions	Must have a trigger for when a hit/foul is called.	
Postconditions	Video should immediately be able to be watched for review by the referee.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Arduino USB signal, Click on spacebar by video ref	
Main Scenario	Step	Action
	1	Referee calls hit or foul
	2	Video referee hits trigger
	3	Program captures video of what was happening before the hit or foul
	4	Referee watches playback
	5	Allow options for multiple speeds for video

Extensions	Step	Branching Action

	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● How to test ● Do we want to use a manual trigger?	

Number	2	
Name	Capture Audio	
Summary	Program shall capture audio alongside with video	
Priority	5	
Preconditions	Must have laptop microphone	
Postconditions	Audio should be recorded and sync with video	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Video recording in progress	
Main Scenario	Step	Action
	1	Referee starts recording
	2	Microphone records audio
	3	System matches audio to video timestamps
	4	Referee watches playback with audio

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● How to test ● Quality of microphone ● Managing memory usage on 8GB laptop ● Audio during slower replay speeds	

Number	3	
Name	Save recorded video and audio to be shared/reviewed	
Summary	Program shall save recordings of the match to standard video format.	
Priority	4	
Preconditions	Video should be recorded and saved in the system	
Postconditions	Referee can share that video	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Referee uses the ‘Play/Pause” button (Spacebar)	
Main Scenario	Step	Action
	1	Referee presses “Play/Pause” button to start recording
	2	Video of match is recorded
	3	Arduino signal is detected
	4	Video recording is halted
	5	Referee is presented 2-4 second playback
	6	After Review referee presses “Play/Pause” button to start recording

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● How to test ● How do we buffer the video for replay? ○ Circular/Ring buffer? ● Managing memory on 8GB laptop	

Number	4	
Name	Operating System compatibility	
Summary	Program shall be able to be installed on both MacOS and Windows using the same source code.	
Priority	5	
Preconditions	Must have laptop with one of the two operating systems	
Postconditions	Program shall install and function the same regardless of OS.	
Primary Actor	Official Referee	
Secondary Actors	Video Referee	
Trigger	Installation	
Main Scenario	Step	Action
	1	Official chooses to use Reposte
	2	Official installs the program on their desired OS
	3	Program installation is successful
	4	Program runs the same on both OS

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● How to test ● How do we make installation easy and accessible? ○ Use GitHub?	

Number	5
Name	Interface with Arduino
Summary	Program shall interface with an Arduino to receive a signal to initiate replay
Priority	5

Preconditions	Must have a trigger for when a hit/foul is called.	
Postconditions	Signal from the Arduino should be taken by the program and initiate replay.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Arduino USB signal.	
Main Scenario	Step	Action
	1	Fencing equipment sends signal to Arduino
	2	Arduino receives signal and transmits it via USB
	3	Program interprets signal then initiates replay functionality

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● How to test ● Knowledge acquisition on microcontroller programming	

Number	6	
Name	Interface with USB, USB-C, and HDMI	
Summary	Program can receive signals and interface with USB, USB-C, and HDMI.	
Priority	5	
Preconditions	USB, USB-C, or HDMI device is plugged into laptop	
Postconditions	Program receives signals through those respective ports.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	USB, USB-C, or HDMI input	
Main Scenario	Step	Action
	1	Referee plugs in necessary devices

	2	System receives signal from device
	3	System interfaces with device

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● How to test ● Memory management ● What products to support? ○ USB/USB-C: Microphone, Additional Camera? ○ HDMI: Monitor support? Max resolution?	

Number	7	
Name	Adjustable replay speed	
Summary	Program can slow down replay speed at 10% intervals(10%-100% speeds)	
Priority	5	
Preconditions	Must have a trigger for when a hit/foul is called.	
Postconditions	Replay video will have adjustable speeds.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Arduino USB signal, Click on spacebar by video ref	
Main Scenario	Step	Action
	1	Replay function is triggered
	2	Replay footage is presented
	3	Replay footage offers adjustable playback speeds
	4	Referee watches playback at desired speeds

Extensions	Step	Branching Action

	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● How to test ● Image quality at reduced speeds ● Audio during slower replay speeds	

Number	8	
Name	Custom naming conventions	
Summary	Program can save recorded video with a desired naming convention	
Priority	3	
Preconditions	User runs software.	
Postconditions	Video should save in desired naming conventions.	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger		
Main Scenario	Step	Action
	1	User runs program
	2	User is prompted to enter save file name
	3	User enters desired save file name
	4	Program saves recording file with given name

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● How to test ● Memory management ● UI design	

Number	9	
Name	Full Screen mode	
Summary	Program can run in full screen	
Priority	4	
Preconditions	Program and dependencies installed.	
Postconditions	Program runs in full screen mode	
Primary Actor	Video Referee	
Secondary Actors	Official Referee	
Trigger	Program ran	
Main Scenario	Step	Action
	1	Official opens program
	2	Program runs in full screen mode
	3	Referee can view replay system in full screen

Extensions	Step	Branching Action
	1a	< condition causing branching > : < action or name of sub use case >
Open Issues	● How to test ● Resolutions constraints ● Memory management	

2. Testing

Through a series of test cases, the following will ensure that the system meets performance and functionality requirements, as well as evaluate the system's ability to capture, save, and replay videos for the referee, while integrating with external devices.

1. **Capture and Replay Video**
   Test Case 1.1: Verify system captures video at 60fps
      Steps:
         1: Start recording

2: Verify the video playing is 60 fps
3: Trigger replay system
4: Ensure video replay never falls below 60fps

Result: Video replay stays above 60fps

Test Case 1.2: Verify user can replay video immediately after capturing

Steps:

1: Start recording
2:Trigger replay system
2: Verify video replay is instant

Result: Video replay is immediate once triggered

2. **Capture Audio**

Test Case 2.1: Record audio with video

Steps:

1: Start recording
2: Trigger replay system
2: Audio is playing with video

Result: Audio is playing with video

Test Case 2.2: Sync audio with video

Steps:

1: Start recording
2.Trigger replay system
3: Audio is playing with video
4: Verify audio is in sync with video

Result: Audio is synced with video and aligns with video cues

3. **Save Recorded Video and Audio to be Shared/Reviewed**

Test Case 3.1: Program should save recordings of the match to standard video format

Steps:

1: Start recording
2: Trigger replay system
3: Verify audio and video works and is in sync
4: Click the save button
5: Verify file is saved to a directory in standard video format
6: Ensure video and audio plays and are sync

Result: Video replays are saved to a directory and can be played for review or shared.

Test Case 3.2: Program should display event markers of saved video locations

Steps:

1:Start recording
2: Trigger replay system
3: Verify video and audio play
4: Click the save button
5: Check file directory for file containing event markers of when replay occurred

Result: Files display event markers for when replay happened. This promotes easier search for specific replays

4. **Operating System Compatibility**

Test Case 4.1: Verify program can be installed on MacOS

Steps:

1: Acquire installation for MacOS
2: Install program
3: Open application and select webcam and microphone
4: Play/Stop video
5: Trigger replay system
6: Verify file is saved to a directory in standard video format
7: Ensure video and audio play and are in sync

Result: Program installs and works as intended on MacOS

Test Case 4.2: Verify program can be installed on Windows

Steps:

1: Acquire installation for Windows
2: Install program
3: Open application and select webcam and microphone
4: Play/Stop video
5: Trigger replay system
6: Verify file is saved to a directory in standard video format
7: ensure video and audio play and are in sync

Result: Program installs and works as intended on Windows

(Optional) Test Case 4.3: Verify program can be installed on Linux

Steps:

1: Acquire installation for Linux
2: Install program
3: Open application and select webcam and microphone
4: Play/Stop video
5: Trigger replay system
6: Verify file is saved to a directory in standard video format
7: ensure video and audio play and are in sync

Result: Program installs and works as intended on Linux

5. **Interface with Arduino**

Test Case 5.1: Verify program receives signal from Arduino to initiate replay

Steps:

1: Verify Arduino is hooked up to computer and fencing equipment
2: Test Arduino signals
3: Start program
4: Verify Arduino triggers replay system by referee stop, hit, or foul

Result: Arduino sends replay signal when referee needs review

6. **Interface with USB, USB-C, and HDMI**

Test Case 6.1: Program receives signals and interfaces with USB, USB-C, and HDMI

Steps:

1: Verify USB, USB-C, or HDMI are properly connected
2: Test Arduino signals
3: Start Program
4: Verify system receives signal from device
5: Verify system interfaces with device

Result: USB, USB-C, and HDMI are all viable sources to receive signal

7. **Adjustable Replay System**

Test Case 7.1: Program can slow down video that varies from 10% - 100%

Steps:

1: Start recording

2: Trigger replay system

3: Select correct recording file from the directory

4: Replay offers adjustable speeds with corresponding number keys for video speed. (i.e. 1 = 10%, 2 = 20%, etc.)

Result: Video files have varying slow motion speeds for accurate hit/foul detection

8. **Custom Naming Convention**

Test Case 8.1: Each replay saved has a custom naming convention for organization

Steps:

1: Start recording

2: Prompt displays to enter save file name

3: Type desired name

4: Program saves each recording with that naming convention

Result: Each saved file is named to correspond with the match for organization purposes and easier to find within the directory

Test Case 8.2: User enters naming convention already in use in directory

Steps:

1: Start recording

2: Prompt displays to enter save file name

3: Type desired name

4: Program prompts user to enter new name for saved file

Result: Program prompts user to enter a new naming convention as the one entered is already taken

9. **Full Screen Mode**

Test Case 9.1: Program can run full screen and windowed

Steps:

1: Start program

2: Select a window button to bring program in or out of full screen

3: Trigger replay system

4: Select video recording for replay in directory

5: Select a window button to bring replay in or out of full screen

Result: Program runs smoothly in full screen or windowed mode and seamlessly transitions between the two.

Test Case 9.2: Program can run at different resolutions

Steps:

1: Start program in computers native resolution

2: Start recording

3: Trigger replay system

4: Select video replay in the directory for playback

5: Ensure 60 fps is consistent at that resolution

6: Apply same test at a range of resolutions (800x600 - 4k)

Result: Video recording and playback stay at or above 60 fps at any resolution

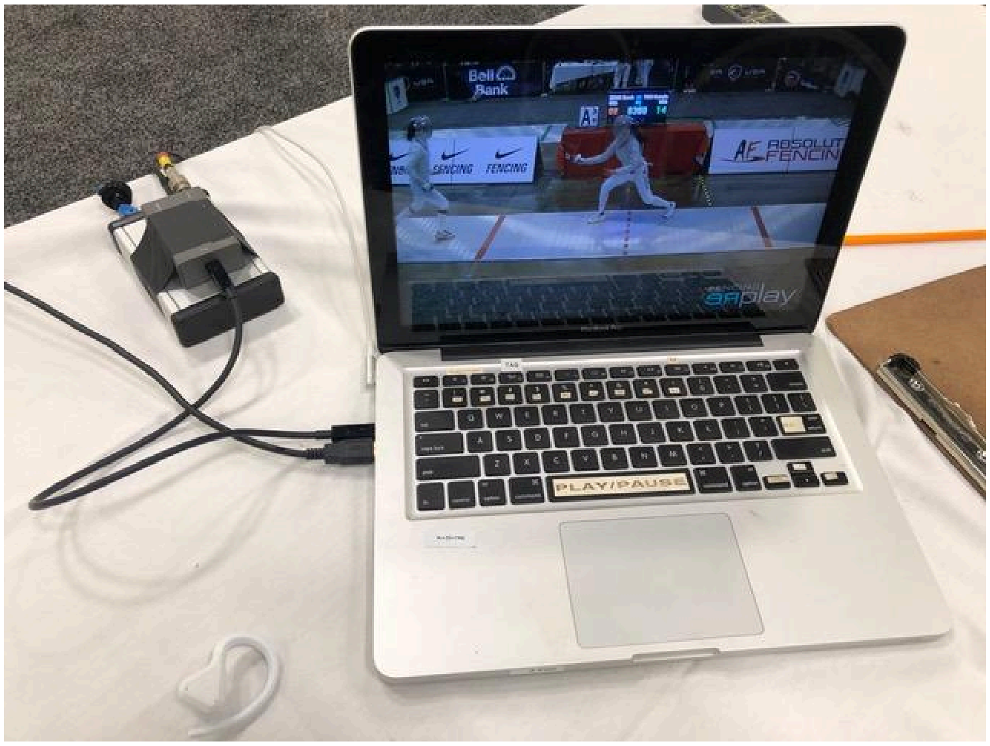
3. Non-Functional Requirements

1. The system must capture and process video and a minimum of 60 frames per second(FPS) and audio in real-time, without dropping frames or introducing noticeable latency.
    a. Priority: 5
    b. Metric: Video recording maintains at least 60FPS on systems with 8GB RAM.
2. The software must run on MacOS, Windows, and preferably Linux, using a single codebase with cross-platform support.
    a. Priority: 5
    b. Metric: System builds and runs successfully on all two/three platforms, with no major differences in functionality.
3. The system must use CPU, memory, and disk resources efficiently, ensuring it does not cause excessive strain on the user's machine, particularly during extended periods of use.
    a. Priority: 4
    b. Metric: CPU usage must remain below 50%, and memory usage below 4GB on a system with 8GB RAM during operation.
4. Software must process video, audio, and input signals from the Arduino device with minimal delay to ensure accurate real-time analysis
    a. Priority: 5
    b. The system must process each frame and audio input within 16.67 milliseconds(for 60 FPS), ensuring latency does not exceed 100 milliseconds.
5. System must provide backup of video files to ensure data isn't lost.
    a. Priority: 4
    b. System will automatically save replays to a secondary storage location once the recording ends.
6. System must put video files in a secure location to prevent unauthorized access.
    a. Priority: 5
    b. Files will be secured behind user authentication and logins that only authorized personnel should have access to.
7. Software must be created to ensure that it is easy to update and make modifications to, to ensure that new developers can update code.
    a. Priority: 3
    b. New developers should be able to update code easily, read the preexisting code, and make these changes without messing up previous functionalities.
8. Software must minimize the use of the systems CPU and GPU while idle.
    a. Priority: 3
    b. While software is idle, CPU and GPU should not go over 20-30% usage.
9. Software must have a simple, yet effective, UI to ensure the referees can use it with minimal training.
    a. Priority: 4
    b. New users/referees should be able to stop/start recording, initiate a replay, change speed of playback, etc. within a few minutes of familiarizing themselves with software.
10. System must be able to handle high-definition video inputs without lowering performance.
    a. Priority: 5
    b. Software should be able to handle video up to 1080p resolution without a drop in frames.

4. User Interface Requirements

In Reposte, user interface design will be minimal as to save computer resources. As the project progresses, the UI design will change to accommodate certain specifications or quality of life updates. At that time, we will assess the needed accessibility features to optimize inclusion and usability.

At the most basic level the user interface will be handled through the keyboard. The functions that will be handled by the keyboard are play/pause, replay speed(10-90%), and ‘halt & replay’. Utilizing the spacebar for play/pause functions. The number keys 1-9 will control the speed of the replay. 1 being 10% speed and 9 being 90% speed. There will be a need for a ‘Halt and Replay’ button. This button, after being pressed, will stop video recording and provide the user with the replay functionality. Client has added a request for a slider for managing the replay speed and area to tag competitors in the replay footage.



(2d)- System Design Document (SDD)

1. Introduction

This capstone project is for the development of a video replay software system for the sport of fencing. This project is for Dr. Terry Yoo, in partial fulfillment of the Computer Science BS degree for the University of Maine. Our team will also be working closely with the University of Maine's fencing club, the Blade Society, when developing this product.

The goal of this project is to create open-source video replay software that can run on a laptop using MacOS or Windows. The USA National Fencing requires a video replay system to help referees review calls during fencing matches. Currently the replay system used is outdated, expensive, closed-sourced and needs to be updated. This product aims to give a solution to clubs and competitions by offering an open-source solution that can increase accessibility to video replay systems.

1.1 Purpose of This Document

The purpose of this Software Design Document is to provide a guide for the design and implementation of the replay system. This document captures the system architecture, data design, and requirements. By detailing architectural decisions, module decomposition, and data handling, this SDD ensures that team members have consistent understanding and execution across development.

This document will serve as the primary reference throughout development, ensuring that the design aligns with requirements and stakeholder expectations. Additionally, the appendix outlines review and agreement protocols.

1.2 References

Project Proposal:
Yoo, Terry (2024). "Video Referee Replay System Software for Sport Fencing Competition" Project Proposal, University of Maine

SRS:
RePoste System Requirements Document (Version 1.0), Goobernauts, 2024

Knowledge Acquisition:
CMake. (n.d.). Welcome to CMake. https://cmake.org/
CMake Documentation. (n.d.). Documentation overview. https://cmake.org/documentation/
CMake. (n.d.). Mastering CMake. https://cmake.org/cmake/help/book/mastering-cmake/
GitHub. (n.d.). CMake quality GitHub action. https://github.com/marketplace/actions/cmake-quality
OpenCV. (n.d.). OpenCV: Open source computer vision library. https://opencv.org/

Client Supplied:
Super Fencing System. (n.d.). Welcome to Super Fencing System. https://superfencingsystem.com/

Koov Broadcasting. (n.d.). Referee video replay user manual.
http://escrime.koovbroadcasting.com/download/RefereeVideoReplayUserManual.pdf

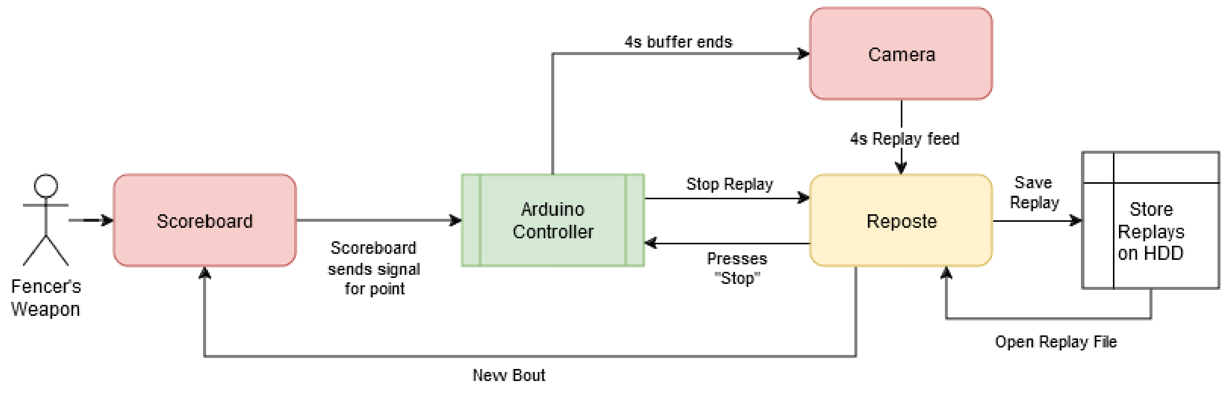
Fédération Internationale d'Escrime. (n.d.). Handbook of specifications for video refereeing. https://static.fie.org/uploads/3/18851-Handbook_of_specification_Video%20Refereeing.pdf

Super Fencing System. (n.d.). SFS-Link manual (Version 1.1). https://superfencingsystem.com/SFS-Link_Manual_V1.1.pdf

2 System Architecture

This section covers the architectural design diagram, hardware, software utilization, and decomposition of the system. The architecture design chosen is the Event-Driven style as the system activates when a trigger occurs and saves the current buffer from the camera for further review.

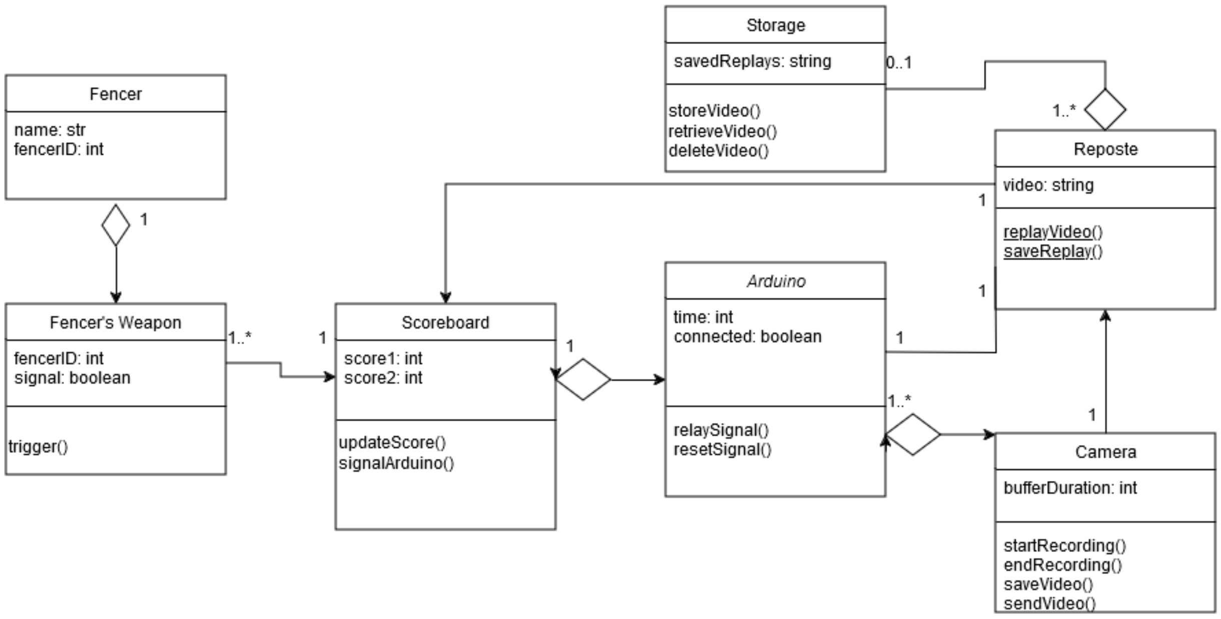
2.1 Architectural Design



The diagram above showcases an event-driven architecture. The triggering event is when the fencer weapon triggers the scoreboard, or when manually triggered through the GUI . When the event occurs, the scoreboard signals the Arduino controller, which sends a signal to both the camera and the system to stop recording and save the last four seconds in the buffer. The four second video is then visible for review in the GUI and then is able to be saved on a hard drive. The files saved on the hard drive are able to be reviewed at a later time through GUI.

Our system will utilize the laptop webcam and an Arduino that is built to work with the fencing scoring machine used by the Blade Society (Favero FA-07) to detect when fencers score a point. When the Arduino triggers or when manually activated, it will signal the system to save the recording from the webcam of the last four seconds where it can be manually reviewed. The file can be later retrieved from the harddrive to be reviewed in the GUI. For software, our project will utilize CMake, Visual Studio Code, and OpenCV for development, ensuring its usability across operating systems, and webcam usage.

2.2 Decomposition Description

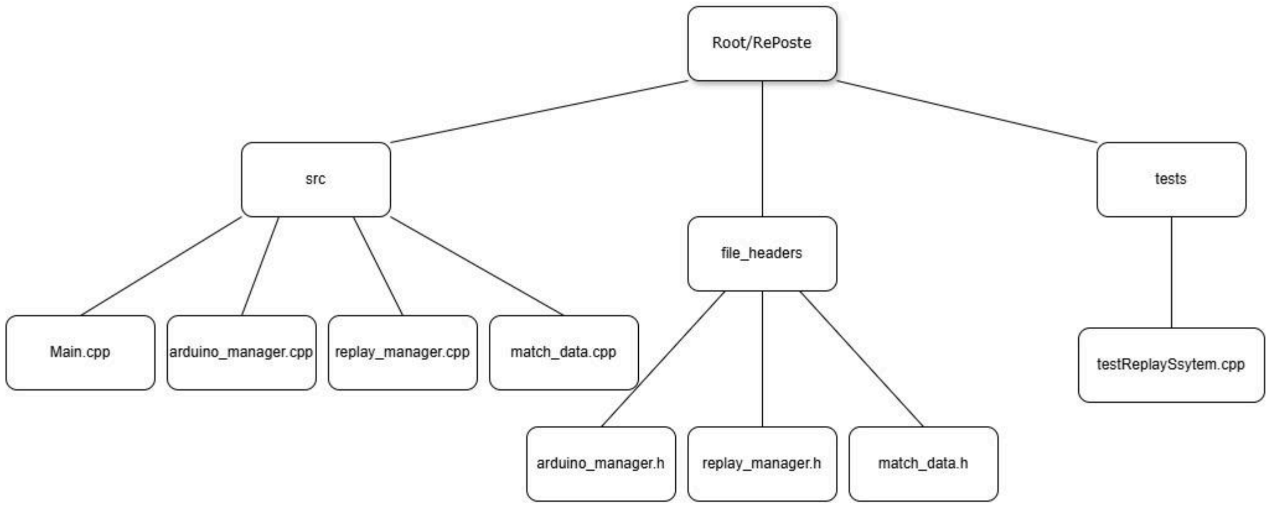


The diagram shows the object-oriented decomposition of our Reposte replay system. This diagram focuses on how each component interacts in order to manage the video playback system. The diagram shows an event-driven architecture where actions like points being scored trigger the rest of the components, making sure that the referee will be able to review. Reposte in this diagram acts as a controller that manages the user interactions and provides an interface. While the other devices like the camera provide a model by capturing the footage.

Each one of the components is representing a class. Each has their own attributes and methods that the class will use. Like for example, the Fencer and Fencing Sabre class both are there to represent the physical inputs that will trigger the rest of the system. The scoreboard receives this signal first, once receiving the signal it then will send a message to the arduino which is used to trigger our camera. Once the camera receives that signal, it begins to record the last few seconds and save that video and send it to the Reposte system. While the Reposte system may interact with several things, it can interact with other classes like the camera and arduino, but is primarily used to access our storage device holding all the videos.

File Descriptions

RePoste will have a concise file format for C++. A src folder will be a child of the root RePoste to help keep everything organized. 'Main.cpp' will include most of the work as it will handle interacting with the arduino, managing replays, and saving replays locally. There will be three other .cpp files, 'arduino_manager.cpp', 'replay_manager.cpp', and 'match_data.cpp' that will handle logic and operations for the arduino, replays, and match information. File_Headers will be another child of the RePoste root and will include the header files for 'arduino_manager.cpp', 'replay_manager.cpp', and 'match_data.cpp' named 'arduino_manager.h', 'replay_manager.h', and 'match_data.h' respectively. These header files will provide interfaces for their .cpp files to manage data and functionality. Tests will be another child of the root RePoste and will include test files 'testReplaySystem.cpp' to run automated tests.



Main.cpp:

- **matchData**: DataType = string/double
    - Contains match details
- **arduinoManager**: DataType = string/double
    - Handle arduino interactions
- **isRecording**: DataType = bool
    - Check if app is recording
- **replayCount**: DataType = double
    - Increment replay number for new save file

Arduino_manager.cpp:

- **arduinoConnection**: DataType = bool
    - Check if arduino is hooked up

Replay_system.cpp:

- **replayFile**: DataType = string(filePath)
    - File path for replays

Match_data.cpp:

- **matchID**: DataType = int
    - ID for current match
- **playerScore**: DataType = int
    - current score of each fencer
- **matchDuration**: DataType = double
    - Match timer
- **matchDate**: DataType = string
    - Date and time of match

Requirements Matrix

This section covers the requirements section of the SDD lining the functional requirements with their associated use cases and the corresponding system component that will satisfy or test the requirement. Some requirements do not have a direct component that would make sense to check so are instead represented in other explanations.

Functional Requirements	System Component
1) System shall record real time video at 60 frames/second so that review can be accurate. (**USE CASE #1**)	int fps60_check() function
2) System shall be able to replay specific moments immediately after they occur so the referee can make accurate judgements. (**USE CASE #1**)	void replay() function
3) System shall record audio along with the video so the audio cues will be captured as well. (****USE CASE**#2**)	audio_recording method
4) System shall work on both MacOS and Windows, and eventually Linux, so it can be used regardless of operating system (**USE CASE #4**)	GitHub Action Build: Runs-on: [ubuntu-latest, macos-latest, windows-latest]
5) System shall save the recorded video and audio with event markers, so that it can be reviewed and shared after a match. (**USE CASE #3**)	void save_file() void event_marker()
6) System shall interface with an Arduino device to capture signals from fencing equipment and log those events during video capture. (**USE CASE #5**)	Boolean signal_receive() function Void log_event() function
7) System shall interface with USB, USB-C, and HDMI inputs. (**USE CASE #6**)	boolean received_input()
8) System shall change the replay speed so that the referee can better understand what	Replay speed can be modified (Default 100%): float replay_speed = 1;

actions occurred. (**USE CASE #7**)	change_replay_speed() method (modifies replay by +/- .1)
9) System shall save recorded videos with a naming convention provided so the referee/official can find saved replays easier. (**USE CASE #8**)	User will input a string before the match: string bout_name = input; save_bout(bout_name, file) function
10) System shall have the ability to function in fullscreen mode so the referee has the best view of the replay. (**USE CASE #9**)	aspect_ratio() function

(3d)- User Interface Design Document(UIDD)

Introduction

This capstone project is for the development of a video replay software system for the sport of fencing. This project is for Dr. Terry Yoo, in partial fulfillment of the Computer Science BS degree for the University of Maine. Our team will also be working closely with the University of Maine's fencing club, the Blade Society, when developing this product.

The goal of this project is to create open-source video replay software that can run on a laptop using MacOS or Windows. The USA National Fencing requires a video replay system to help referees review calls during fencing matches. Currently the replay system used is outdated, expensive, closed-sourced and needs to be updated. This product aims to give a solution to clubs and competitions by offering an open-source solution that can increase accessibility to video replay systems.

1.1 Purpose of This Document

The purpose of this UIDD document is to provide a guide for the user interface design choices for the replay system. The document covers the user interface standards, user walkthrough of the interface, and data validation. It also includes accessibilities and inclusive design explanations, diagrams detailing screen navigation, button functionality, and data validation. The document is meant to serve as a resource for stakeholders and development

1.2 References

This section includes citations to project documentation and other sources that were used to develop the project's design and interface thus far. PyQt6 and ImageIO are Python libraries used to handle UI construction and video feed playback respectively. The use of these libraries are the result of testing the functionality of OpenCV. The team chose to go forward with ImageIO for video functionalities over OpenCV; the reason being we encountered issues with video playback speeds when using the OpenCV library. Super Fencing System is an accessible and mobile iOS application that produces the replay abilities we hope RePoste to have as a completed project. The final two references are design guides related to color and its use in web and application design, and user accessibility and inclusive design.

1. **Proposal:**

   Yoo, Terry (2024). "Video Referee Replay System Software for Sport Fencing Competition" Project Proposal, University of Maine.
2. **SRS:**

   RePoste System Requirements Document (Version 1.0), Goobernauts, 2024
3. **SDD:**

   RePoste System Design Document (Version 1.0), Goobernauts, 2024

Knowledge Acquisition:
1. GitHub. (n.d.). CMake quality GitHub action. https://github.com/marketplace/actions/cmake-quality
2. Imageio. (n.d.). Imageio(2.36.1). https://pypi.org/project/imageio/
3. Kitware, Inc. (n.d.). Documentation overview. CMake. https://cmake.org/documentation/
4. Kitware, Inc. (n.d.). Mastering CMake. CMake https://cmake.org/cmake/help/book/mastering-cmake/

5. Kitware, Inc. (n.d.). Welcome to CMake. CMake. https://cmake.org/
6. OpenCV. (n.d.). OpenCV: Open source computer vision library. https://opencv.org/
7. Riverbank Computing.(n.d.). PyQt6(6.7.1). https://pypi.org/project/PyQt6/
8. U.S. General Services Administration. (n.d.). Color overview: Design tokens. U.S. Web Design System. https://designsystem.digital.gov/design-tokens/color/overview/
9. U.S. General Services Administration. (n.d.). UX design: Accessibility for Teams. https://digital.gov/guides/accessibility-for-teams/ux-design/#content-start

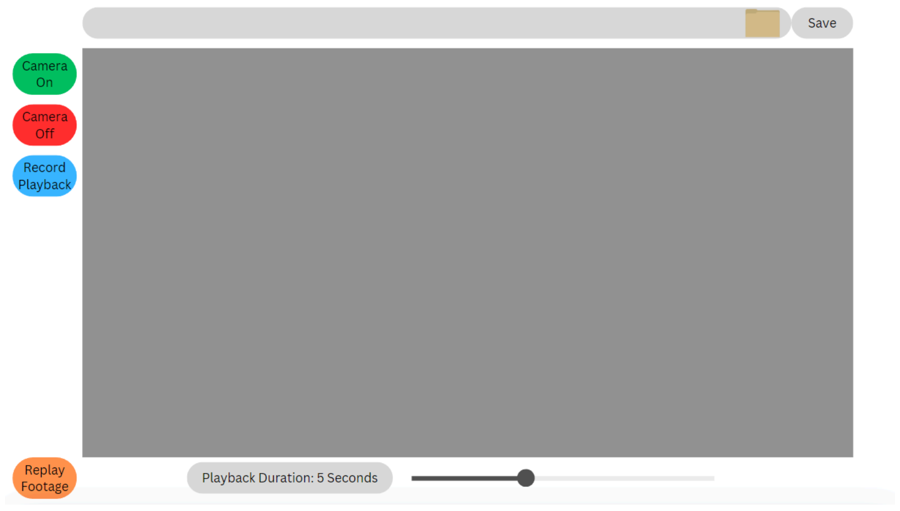
Client Supplied:

1. Super Fencing System. (n.d.). Welcome to Super Fencing System. https://superfencingsystem.com/
2. Koov Broadcasting. (n.d.). Referee video replay user manual. http://escrime.koovbroadcasting.com/download/RefereeVideoReplayUserManual.pdf
3. Fédération Internationale d'Escrime. (n.d.). Handbook of specifications for video refereeing. https://static.fie.org/uploads/3/18851-Handbook_of_specification_Video%20Refereeing.pdf
4. Super Fencing System. (n.d.). SFS-Link manual (Version 1.1). https://superfencingsystem.com/SFS-Link_Manual_V1.1.pdf

1. User Interface Standards

Our user interface standards section is going to remain simple and provide the referees with an easy to follow video replay system UI. We will do this through making use of the keyboard, buttons, sliders, and arrows with text that will help make sure referees understand the controls. This section will ensure that interface design standards are being met throughout the RePoste system.

For the first menu, the following standards should be followed.



Layout:

- The UI should remain in the following format.
  - There will be space on the left to have the buttons needed to turn the camera on and off, record the playback, and be redirected to the replay menu through the Replay Footage button.

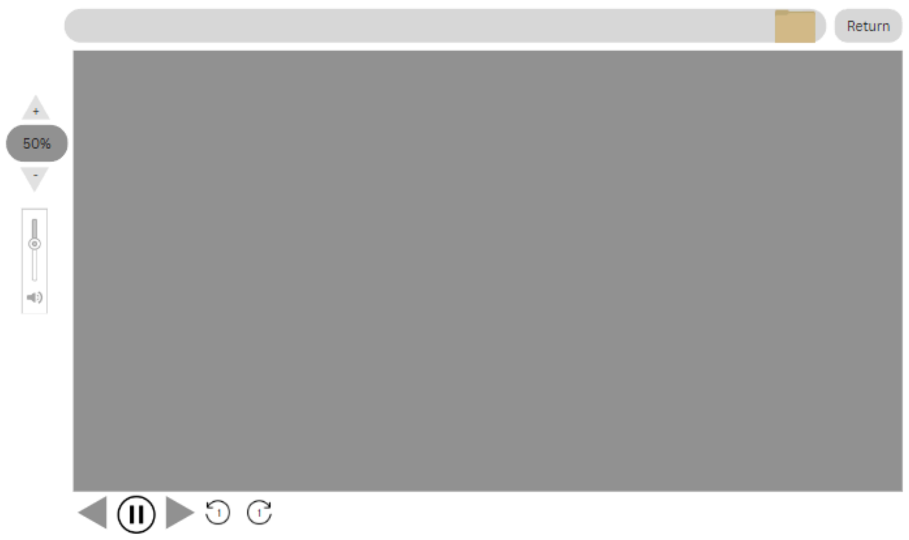
        - If more buttons are included on the left side, ensure that a scroll feature is implemented where a user can scroll to desired function.
        - The buttons are color coded to have some form of recognition between the functions
            - Green: Camera On
            - Red: Camera Off
            - Blue: Playback recording functions
            - Orange: Replay Footage menu.
    - Center of the screen is the video display.
        - Once the Camera On button is clicked, the middle of the screen should show the camera being used to record the fencing competition.
    - The top of the layout is reserved for the file directory the playbacks are being sent to.
        - On this bar there is a folder icon to choose the desired footage storage location.
        - The save button ensures that all recordings are being saved into the users wished location.
    - Underneath the camera player should be left for features like sliders.
        - Currently in this section is a Playback Duration slider that changes how much playback is being recorded

Color:

- The color scheme will be relatively simple. Keeping a plain gray background will help maintain a professional look without having too many distracting colors.
- Buttons should have vibrant colors to show that they are clear to be used and used for functions

Text:

- The text should remain simple and be readable for users. This includes:
    - Having the text remains large.
    - Have the text remain as a simple font to ensure readability.
    - Mark features like sliders, arrows, and buttons.



For the second menu the format stays relatively the same. The middle for the replay footage, the left and bottom for any features relating to the video playback. The top for directory location/what video is being pulled up.

- For this menu some things will be added. Such as a menu to pull up clips that are saved inside of the directory.
- Section left of the replay footage contains replay functions.
    - First there is the video playback speed represented by the + and - arrows with text between them stating the percent speed.
    - Underneath that, we have the audio control for the video.
- Bottom left section has the controls for video playback.
    - The play/pause buttons are represented by arrows for skipping and replaying the clips.

    - Buttons to replay the clips by 1 second either forward or backwards.
- Top bar has a Return button to bring the users back to the first menu.

2.1 Accessibility and Inclusive Design Elements

Accessibility features will be handled primarily by stylings in the UI. The main focus to start will be ADA compliant colors in our design to be inclusive for users with visual disabilities. This means avoiding color combinations that might hinder people with colorblindness from operating the system. Another focus will be on ensuring the design is compatible with cognitive disabilities like dyslexia. We plan to use appropriate sans serif font styles and proper color contrast to accommodate these users. When accommodating users with hearing impairments, the system will have a feature to toggle on/off the audio recording. Accommodations for users with motor impairments will be full keyboard navigation and clear visual cues. Stretch goals could include accessibility features like fencer pose detection to start video recording and controller support. This would limit the amount of interactions that the video referee would need to have with the system if they are dealing with limited dexterity.

Vision disabilities: Color Blindness
-Use color ADA compliant color combinations
-Black and White

Cognitive disabilities: Dyslexia
-Utilize sans serif typeface
-Implement proper color contrast

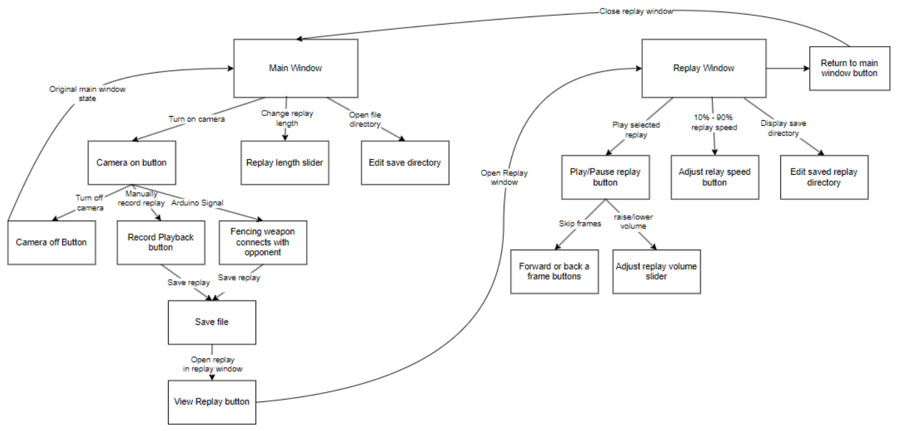
Hearing disabilities: Deafness, low-hearing
-Toggle audio
-Visual Cues

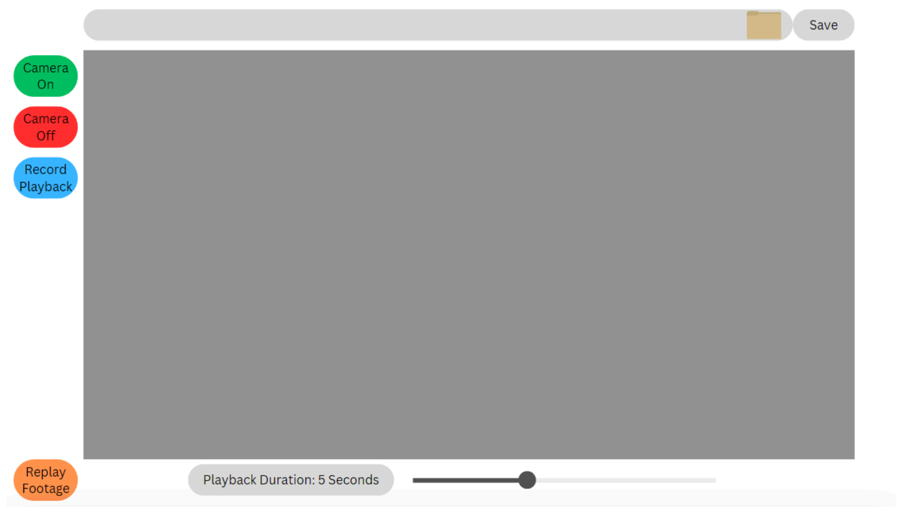
Motor problems: Loss of limb permanently, temporarily, or situationally.
-Full keyboard navigation

2. User Interface Walkthrough

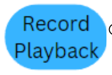
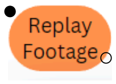
This section contains our walkthrough for the User Interface, below are diagrams for user navigation and a key containing symbol meanings. There are additional screenshots of all system screens numbered, labeled, with explanations for what the user is seeing, menus, functions, and navigation instructions.

Navigation Diagram:



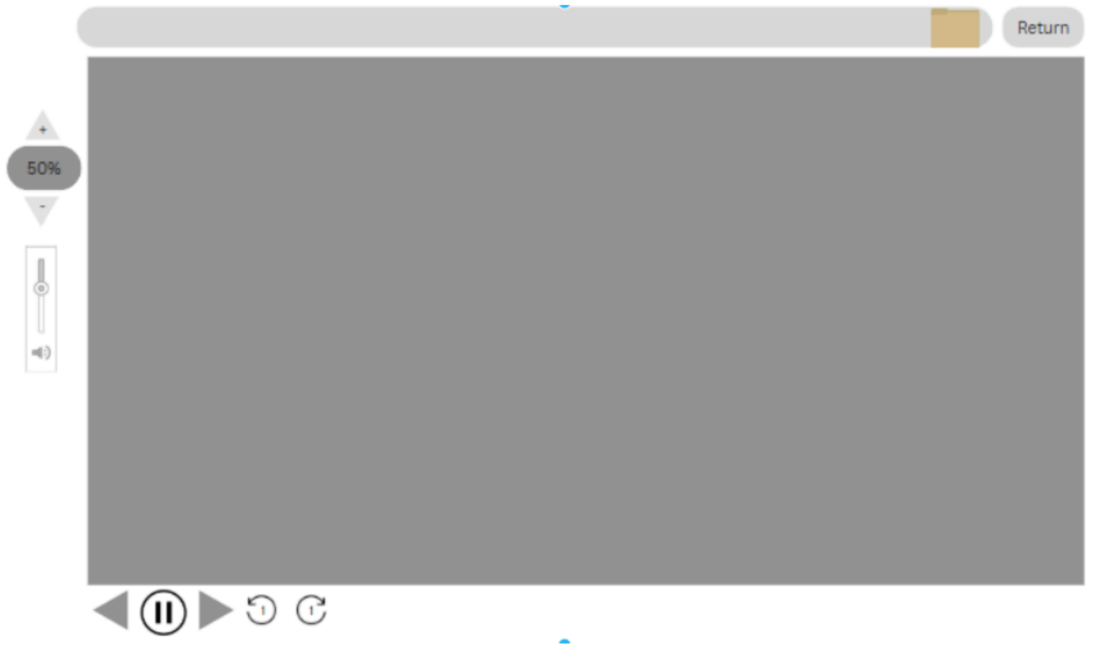


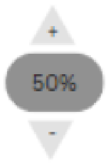
Menu:

- (*Figure 1*)
  - The Camera On button will turn your camera on within the viewing window.
- (*Figure 2*)
  - The Camera Off Button turns off your camera and gets rid of the display in the viewing window.
- (*Figure 3*)
  
  - The Record Playback button will record how many seconds you have set in the Playback Duration slider.
-  (*Figure 4*)
  - The Replay Footage button brings you to the second menu where you can replay the footage you have saved using the Record Playback button.
- (*Figure 5*)
  - The Playback Duration slider is used to set your playback duration, currently we have it set to 1-10 seconds of recording.

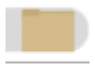
- (*Figure 6*)
  - The Folder button is for opening and setting the directory you wish for your replays to be stored in.
- (*Figure 7)*

  - The Save button ensures that all footage is being saved to the directory.



- (*Figure 8*)
  - Play/Pause button for video replay. The left and right arrows are for starting at the beginning or end of the clip. Triggered by the Spacebar
-  (*Figure 9*)

  - Replay playback button which can adjust the video speed of the replay ranging from 10% - 90%. Handled by the 1-9 keys on the keyboard.
- (*Figure 10*)
  - Skip frame button allows the user to skip forward or backward one frame at a time. Handled by the left and right arrow buttons
- (*Figure 11*)
  - Volume slider to adjust volume levels of video replay



- (*Figure 12*)
    - The Folder button is for displaying the file directory that the replay will be saved to.

- (*Figure 13*)
    - Return button to leave the replay window. “Return” is a temporary name.

3. Data Validation

Name	Data Type	Limitation	Format
Video File	MP4	Must be MP4, if error reject and reprompt	MP4
Frame Image	NumPy Array	If values are missing, default color values to (0,0,0)	Dependent on Camera
Arduino Signal [ESP32] (Planned)	Boolean	Must be true or false, if error default to false	True/False
Video File Name (Planned)	String	Must follow OS naming limits, if error reject name and reprompt	videofilename.mp4
Save Location (Planned)	String	Must be a valid location, if error reject location and reprompt	C:\Users\Username\...

(4d) Super Fencing System SFS- Link Manual

Super Fencing System **V1.1**
SFS-Link **Manual**

superfencingsystem.com
superfencingsystem@tuta.com

Instructions

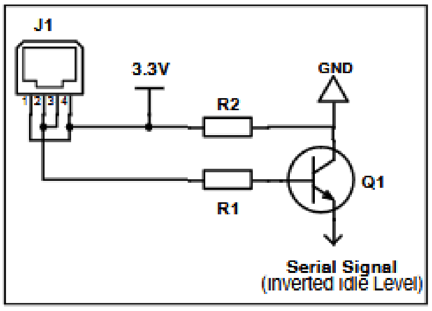
1. Plug RJ11 cable into FA-01 / FA-05 / FA-07 / FA-15 DATA OUT jack and SFS-Link RJ11 jack.
2. Power SFS-Link with USB-C cable.
3. SFS-Link will now be discoverable by Super Fencing System.

Microcontroller LED Functions

"D4" LED:	**FLASHES** when scoring machine data is **NOT** detected. **SOLID** when scoring machine data **IS** detected.
"D5" LED:	**OFF** when SFS-Link is **NOT** connected to Super Fencing System. **ON** when SFS-Link **IS** connected to Super Fencing System.

PCB Components
R2: 0805 5.6KΩ
R1: 0805 10KΩ
Q1: SOT-23 MMBT3904 NPN BJT
J1: MOLEX 432026105 6P4C RJ11 JACK
Microcontroller: ESP32 C3*
*AirM2M_CORE_ESP32C3 in Arduino IDE

PCB contains lead-free HASL.
Lead-free solder used for soldering PCB components.

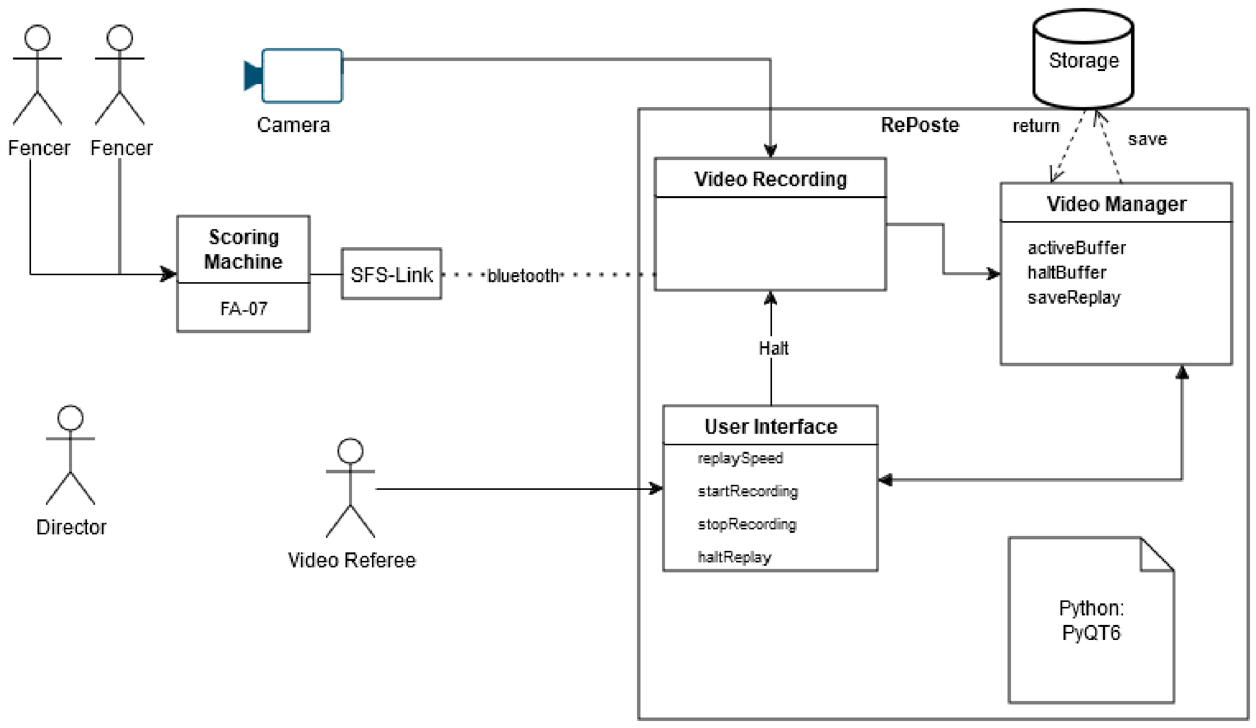


SFS-Link Hardware Schematic V1.1

SFS-Link Bluetooth Protocol V1.1
Bluetooth (BLE) Device Name: "SFS-Link [S/N]"
Service & Characteristic UUID: "6F000000-B5A3-F393-E0A9-E50E24DCCA9E"
Characteristic is a 14-character string.
Characteristic is initialized to "00000000000000"
Characteristic is set to "00000000000000" when no Favero data is read for over 0.5 seconds.
Characteristic has read and notify properties.
Characteristic is concatenated bytes 2-7 and 9 of Favero Serial 10-byte message, in hex.
Example: "06125602140A38" where 06…38 is byte 2…9.

NOTE: The Favero serial protocol sends delayed machine data. This delay is 50-200 milliseconds. To account for this delay, set 'Machine Data Time Offset' in SFS settings to ~0.10 seconds when using a SFS-Link.

(5d) System Decomposition Diagram


Fencer
Fencer
Camera
Storage
RePoste
return
save
Video Recording
Video Manager
activeBuffer
haltBuffer
saveReplay
Scoring Machine
FA-07
SFS-Link
bluetooth
Halt
User Interface
replaySpeed
startRecording
stopRecording
haltReplay
Director
Video Referee
Python:
PyQT6