SLOW-ARC
Design Document

Authors:

Shiva Cheruvu

Roshan Gokul

Eric Liu

Maurice Yu

Group: 4

This page intentionally left blank.

Document Revision History

Date	Version	Description	Author
11/16/2023	0.1	Initial draft	Shiva Cheruvu, Roshan Gokul, Eric Liu, Maurice Yu

This page intentionally left blank.

Contents

Introduction

1.1 Purpose

The purpose of this document is to provide managers and engineers with the proper blueprint to successfully design the SLOW-ARC system. Within this document, there will be system design, system architecture, and structural design diagrams in addition to clear and concise descriptions of how to implement the requirements and specifications.

1.2 System Overview

The SLOW-ARC system is designed to provide an affordable and precise ball-strike determination solution for slow-pitch softball games. Using cameras and rule-based software logic, the system captures the trajectory and position of the ball in 3D space relative to the batter and home plate. This enables accurate decisions on whether a pitch is a ball or a strike, in line with the Official Rules of Softball 2023. Intended to be as precise as an umpire, the system offers a blend of technological accuracy and cost-effectiveness, catering to local sports organizations that seek a modern approach to officiating.

1.3 Design Objectives

The design of the SLOW-ARC system should appeal to USA Softball. This means that the design of the system should cater towards fulfilling a cheap, yet efficient and easy-to-use solution. In correspondence to designing the system, the solution should be able to solve all of the necessary tasks in order to remain a competitive option.

This includes an easy-to-use interface with customizable settings for each user to review plays from various perspectives and get comprehensive pitch data. Strong security mechanisms will be in place to stop unwanted access, real-time feedback on user activity will be provided, and a strong database capable of handling large amounts of team and league data will be included, along with backup, recovery, and routine maintenance. Pitch speed, trajectory, and position will be assessed by sophisticated querying and analytical features, providing post-game analysis and accurate batting movement monitoring. In accordance with USA softball laws, the camera system will track and record pitches, guaranteeing that data is recorded and saved for in-depth study.

1.4 References

Software Specifications for SLOW - ARC Version 1.1

1.5 Definitions, Acronyms, and Abbreviations

~

2 Design Overview

2.1 Introduction

For the SLOW-ARC system, we are using an object-oriented design with event-driven architecture of the system. Other techniques and tools are used, including Visual Studio Code for code implementation, LucidChart for UML modeling.

2.2 Environment Overview

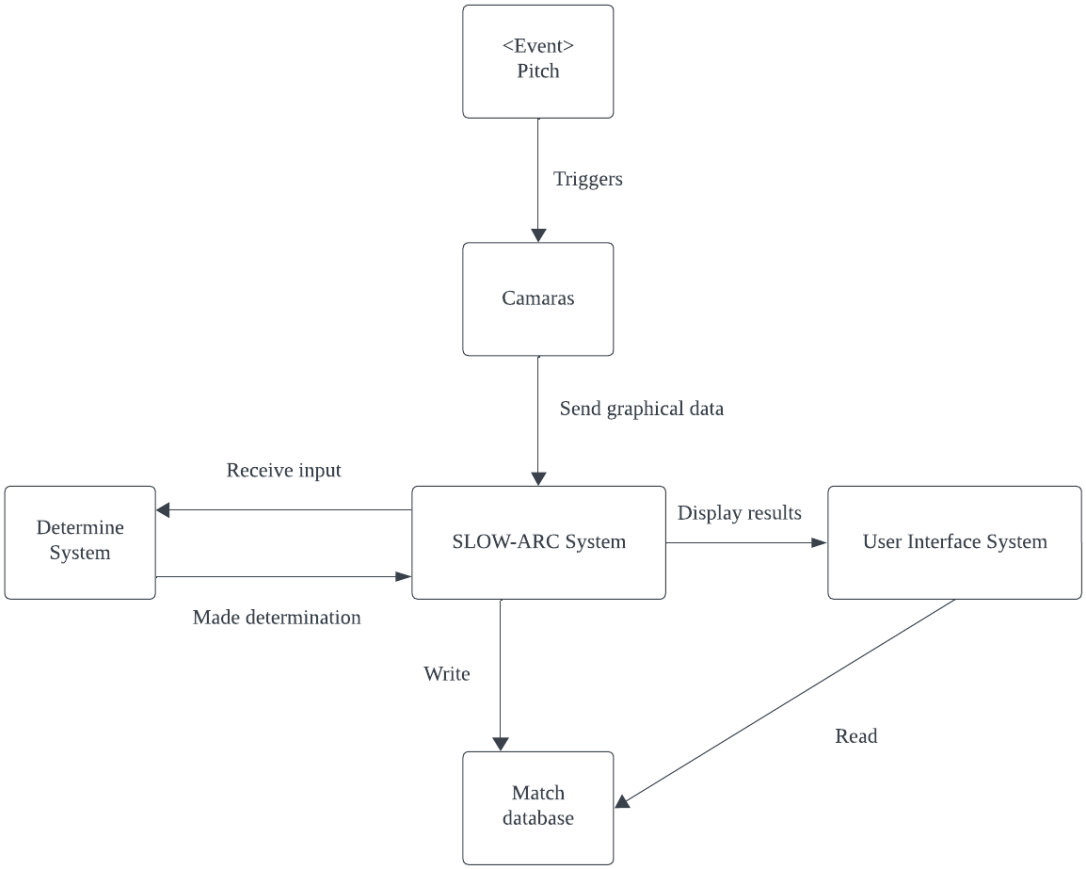
The SLOW-ARC system is divided into four subsystems. There is a camera system, determination system, notification system and match database system. Each of these has different operating environments.

For the camera system, the operating environment will be a regular softball/baseball field. We will have the cameras to accurately examine the pitch. It will be triggered once a pitch is made.

For the determination, notification and database system, the operating environment will be on a commercial computer. the software will be runned by the users before a pitch is made.

2.3 System Architecture

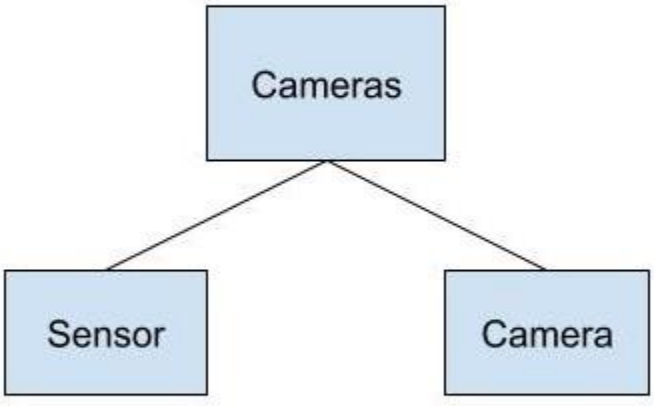
2.3.1: Top-level system structure of SLOW-ARC system:



The SLOW-ARC system uses event-driven architecture. When a pitch is made, an event is generated with the information about the pitch. This triggers the camera system which will

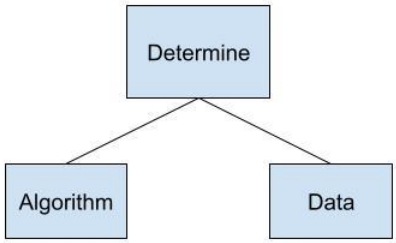
capture the graphical data of the pitch and send it as well as the event to the Determine System. The Determined system will receive input from the Camera System and give determination. The result will be displayed to the user by the User Interface System. Match database will record each matches' stats.

2.3.2 Camaras subsystem



The camera subsystem consists of the sensor and the camera.

2.3.3 Determine subsystem



The determine subsystem consists of an algorithm and data.

2.4 Constraints and Assumptions

1. One of the main points of the SLOW-ARC is to provide a cheap alternative to the systems that professional tennis and baseball use.

   This is addressed by using cameras and commercial computers as the platform of the system. Without using expensive high-speed cameras and external devices, the cost is greatly controlled.

2. The cameras should be able to be set up in under 20 minutes to entice umpires to enforce the system.

   This is addressed by using cameras and commercial computers as the platform of the system. Any computers installed with the software and mainstream OS can be used as the platform, reducing the time of deployment.

   We made the following assumptions:

1. The environment is never going to affect the SLOW-ARC system.
2. The ball the cameras will track will be constant.
3. The batter will not miss the pitch.

3 Interfaces and Data Stores

This section describes the interfaces into and out of the system as well as the data stores you will be including in your system.

3.1 System Interfaces

3.1.1 Camera Interface

This interface is used to capture the visual data used later on, to determine whether a pitch is a strike, ball, miss, or hit. The data can be accessed by the software used in the determination of the pitch. Right after the camera captures the visual data from the pitch, this data goes to the determination interface.

3.1.2 Determination Interface

The determination interface uses machine learning algorithms to precisely interpret the visual data received from the camera interface. Once this data is interpreted by the interface, the determination interface will distinguish between average statistics such as average ball speed, average pitch throws and the immediate result of the most recent pitch. Once all this data is gathered, it will be transferred into a file that will be stored in the database interface.

3.1.3 Database Interface

The database interface is here to distinguish between the different types of information sent by the determination system. Like we mentioned previously, the determination interface is sending a lot of data to the database interface. So once this data is interpreted by this interface, it will immediately send the most recent pitch to the user interface automatically. All of the other data included in the match statistics will be left alone unless called for by the User interface.

3.1.4 User Interface

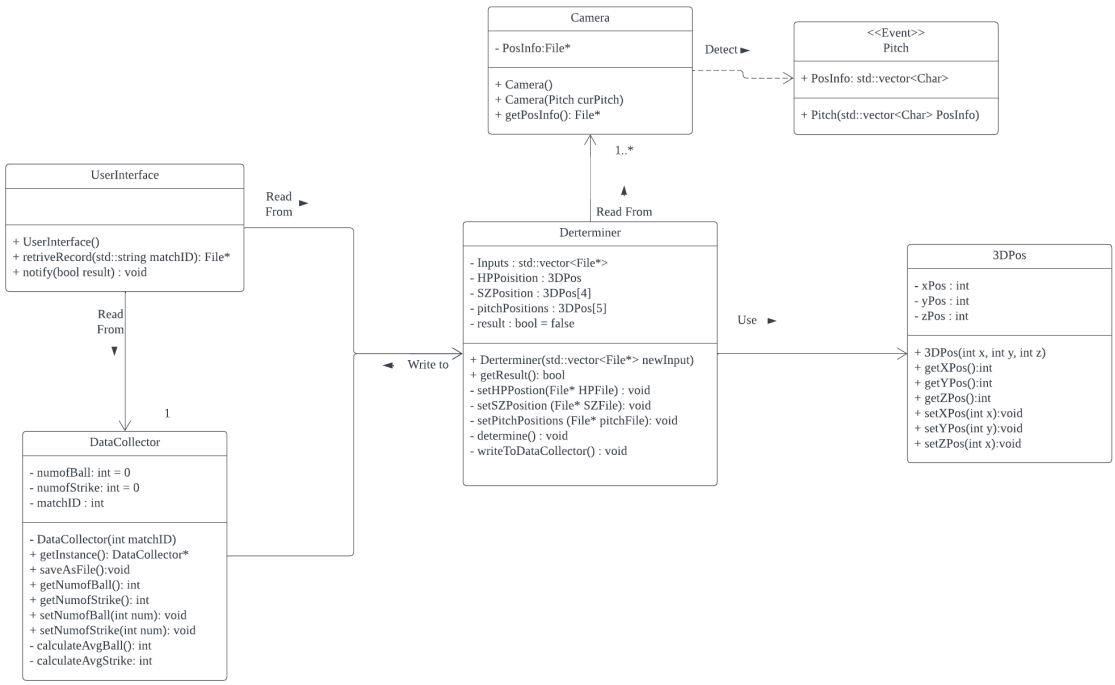
The user interface is how officials, coaches, and players will be able to interact with the results. The user interface automatically receives the most recent pitch statistic from the database interface so that the game is constantly in motion with no delays. In addition, the user interface has the option to ask the database interface for the other match statistics like average ball speed and average pitch results.

3.2 Data Stores

The SLOW-ARC system has one data store which is our database interface. The database interface is mainly implemented so match statistics can be kept long after their life in addition to providing the user interface the option to retrieve statistics that may not be as important to the current game, but could be important in other aspects of the game.

4 Structural Design

4.1 Class Diagram



When a pitch is made, an event is generated with the information about the pitch. This triggers the camera which will capture the graphical data of the pitch and send it as well as the event to the determiner. The determiner will receive input from the camera and give determination. The result will be displayed to the user by the UserInterface class. The stats of a match is written to the DataCollector class and stored as a file.

4.2 Class Descriptions

4.2.1 Class: Pitch

Purpose: *To express an event that a pitch is made.*
Constraints: *None*
Attributes: *PosInfo*
Methods*: Pitch(std::vector<Char> PosInfo)*

4.2.1.1 Attribute Descriptions

1. Attribute: *PosInfo*

Type: *std::vector<Char>*
Accessibility: *Public*
Description: *The encoded information about the position of a pitch.*
Constraints: *None*

4.2.1.2 Method Description

1. Method: *Pitch(std::vector<Char> PosInfo)*
   Return Type: *N/A (as it is a constructor)*
   Parameters: *PosInfo – the current pitch's position information*
   Pre-condition: *A pitch is made*
   Post-condition: *A pitch event with position information is created*
   Attributes read/used: *None*
   Methods called: *None*

   Processing logic:
   This method is called once a pitch is made, takes the pitch's position information and generates an event to notify the Camera class that a pitch is made.

4.2.2 Class: Camera

Purpose: *Translate the raw data captured to the forms of files used as the input for the determiner class.*
Constraints: There's at least one instance of Camera class.
Attributes: *PosInfo*
Methods*: Camera(Pitch curPosInfo),Camera(), getPosInfo()*

4.2.2.1 Attribute Descriptions

1. Attribute: PosInfo
   Type: *FIle**
   Accessibility: *Private*
   Description: *The information about the position of a pitch in file form.*
   Constraints: *None*

4.2.2.2 Method Description

1. Method: *Camera(Pitch curPosInfo)*
   Return Type: *N/A (as it is a constructor)*
   Parameters: *curPosInfo– the current pitch event object*
   Pre-condition: *A pitch event is generated*
   Post-condition: *the position information of the pitch is translated into file form.*
   Attributes read/used: *None*
   Methods called: *None*

   Processing logic:
   This method is called when a pitch event is generated, takes the pitch's position information as an encoded character vector and translates it into a file form for the Determiner class.

2. Method: *Camera()*
   Return Type: *N/A (as it is a constructor)*
   Parameters: *None*
   Pre-condition: *None*
   Post-condition: *The default Camera object is created.*
   Attributes read/used: *None*
   Methods called: *None*

   Processing logic:
   This method is called at the setup phase(before the match begins) to indicate cameras that are capturing the position of the home plate and strike zone instead of the pitch.

3. Method: *getPosInfo()*
   Return Type: *File**
   Parameters: *None*
   Pre-condition: A *Camera object is created.*
   Post-condition: *the position information of the pitch or the home plate and strike zone are sent to the caller.*
   Attributes read/used: *PosInfo*
   Methods called: *None*

   Processing logic:
   This method is called when the Determiner class is requesting positions of the pitch, strike zone and home plate. If the PosInfo for the camera object is NULL, then this method will take in the positions of the home plate and strike zone, translate them into file forms and send them to the Determiner class. Otherwise this method retrieves PosInfo and sends it.

4.2.3 Class: 3DPos

Purpose: *To express the position of a point in 3-Dimensional form.*
Constraints: *None*
Attributes: xPos, yPos, zPos
Methods*: 3DPos(int x, int y, int z), getXPos(), getYPos(), getZPos(), setXPos(int x), setYPos(int y), setZPos(int z)*

4.2.3.1 Attribute Descriptions

1. Attribute: xPos
   Type: *int*
   Accessibility: *Private*
   Description: *The x coordinate of a point.*
   Constraints: *None*

2. Attribute: yPos
   Type: *int*
   Accessibility: *Private*

Description: *The y coordinate of a point.*
Constraints: *None*

3. Attribute: zPos
   Type: *int*
   Accessibility: *Private*
   Description: *The z coordinate of a point.*
   Constraints: *None*

4.2.3.2 Method Description

1. Method: *3DPos(int x, int y, int z)*
   Return Type: *N/A (as it is a constructor)*
   Parameters: *x - the x coordinate of the new object; y - the y coordinate of the new object; z - the z coordinate of the new object*
   Pre-condition: *None*
   Post-condition: *A 3DPos object with x,y,z coordinates are created*
   Attributes read/used: *None*
   Methods called: *None*

   Processing logic:
   This method is called when the Determiner wants to create 3DPos objects to express the position of the pitch, the strike zone and the home plate.

2. Method: *getXPos()*
   Return Type: *int*
   Parameters: *None*
   Pre-condition: *A 3DPos object is created.*
   Post-condition: *x coordinate of the 3DPos object is provided.*
   Attributes read/used: *xPos*
   Methods called: *None*

   Processing logic:
   A getter function for xPos.

3. Method: *getYPos()*
   Return Type: *int*
   Parameters: *None*
   Pre-condition: *A 3DPos object is created.*
   Post-condition: *y coordinate of the 3DPos object is provided.*
   Attributes read/used: *yPos*
   Methods called: *None*

   Processing logic:
   A getter function for yPos.

4. Method: *getZPos()*
   Return Type: *int*
   Parameters: *None*
   Pre-condition: *A 3DPos object is created.*
   Post-condition: *z coordinate of the 3DPos object is provided.*
   Attributes read/used: *zPos*
   Methods called: *None*

Processing logic:
A getter function for zPos.

5. *Method: setXPos(int x)*
Return Type: *void*
Parameters: *int x - the new xPos value of the object.*
Pre-condition: *A 3DPos object is created.*
Post-condition: *x coordinate of the 3DPos object is changed.*
Attributes read/used: *xPos*
Methods called: *None*

Processing logic:
A setter function for xPos.

6. *Method: setYPos(int y)*
Return Type: *void*
Parameters: *int y - the new yPos value of the object.*
Pre-condition: *A 3DPos object is created.*
Post-condition: *y coordinate of the 3DPos object is changed.*
Attributes read/used: *yPos*
Methods called: *None*

Processing logic:
A setter function for yPos.

7. *Method: setZPos(int z)*
Return Type: *void*
Parameters: *int z - the new zPos value of the object.*
Pre-condition: *A 3DPos object is created.*
Post-condition: *z coordinate of the 3DPos object is changed.*
Attributes read/used: *zPos*
Methods called: *None*

Processing logic:
A setter function for zPos.

4.2.4 Class: Determiner

Purpose: *To determine whether a pitch is a ball or strike, and sends the result to both the* DataCollector and UserInterface class
Constraints: *None*
Attributes:*inputs, HPPosition, SZPosition, pitchPositions, result.*
Methods*: Determiner(std::vector<Files*> newInput), getResult(), setHPPosition(File* HPFile), setSZPosition(File* SZfile), setPitchPositions(File* pitchFile), determine(), WriteToDataCollector()*

4.2.4.1 Attribute Descriptions

1. Attribute: *inputs*
   Type: *std::vector<Files*>*
   Accessibility: *Private*
   Description: *the positions of the pitch, the home plate and the strike zone in file forms*
   Constraints: *None*

2. Attribute: *HPPosition*
   Type: *3DPos*
   Accessibility: *Private*
   Description: *the position of the home plate in 3DPos form.*
   Constraints: *None*

3. Attribute: *SZPosition*
   Type: *3DPos[4]*
   Accessibility: *Private*
   Description: *the position of the strike zone in 3DPos array form. The strike zone is a rectangle enclosed by four 3DPos points.*
   Constraints: *None*

4. Attribute: *pitchPositions*
   Type:*3DPos[5]*
   Accessibility: *Private*
   Description: *the positions of the pitch in 5 seconds in 3DPos form. Each 3DPos object is the position of the pitch in a second.*
   Constraints: *None*

5. Attribute: *result*
   Type: *bool*
   Accessibility: *Private*
   Description: *the result of a pitch. If it is "True" then the pitch is a strike. If it is "False", then the pitch is a ball.*
   Constraints: *Set to "False" by default.*

4.2.4.2 Method Description

1. Method: *Determiner(std::vector<Files*> newInput)*
   Return Type: *N/A (as it is a constructor)*
   Parameters: *newInput - The position information of the pitch, strike zone and home plate in file format.*
   Pre-condition: *the position information of the pitch, strike zone and home plate are available in file format.*
   Post-condition: A *Determiner object is created to decide a pitch.*
   Attributes read/used: *None*
   Methods called: *None*

   Processing logic:
   This method is called when the position information of the pitch, strike zone and home plate are sent by the Camera class and are ready to be compared, A *Determiner object will be created to compare them.*

2. Method*: getResult()*

Return Type: *bool*
Parameters: *None*
Pre-condition: *Method determine() is called and a result is generated.*
Post-condition: *The result of a pitch ("True" for strikes, "False" for ball) is provided.*
Attributes read/used: *result*
Methods called: *None*

Processing logic:
This method is called upon the request of the UserInterface class to notify the user of the result.
DO NOT call this method alone. (without the presence of the determine() method before it).

3. Method: *setHPPosition(File* HPFile)*
Return Type: *void*
Parameters: *HPFile - the position information for the home plate in file form.*
Pre-condition: *A Determiner object is created.*
Post-condition: *The position information for the home plate is available in 3DPos form.*
Attributes read/used: *inputs, HPPosition*
Methods called: *None*

Processing logic:
This method is called when the position information of the home plate is available in file form. It takes in that file and assigns HPPosition with the file's 3DPos equivalent.

4. Method: *setSZPosition(File* SZFile)*
Return Type: *void*
Parameters: *SZFile - the position information for the strike zone in file form.*
Pre-condition: *A Determiner object is created.*
Post-condition: *The position information for the strike zone is available in 3DPos form.*
Attributes read/used: *inputs, SZPosition*
Methods called: *None*

Processing logic:
This method is called when the position information of the strike zone is available in file form. It takes in that file and assigns SZPosition with the file's 3DPos equivalent.

5. Method: *setPitchPositions(File* pitchFile)*
Return Type: *void*
Parameters: *pitchFile- the position information for the pitch in file form.*
Pre-condition: *A Determiner object is created.*
Post-condition: *the position of the pitch in 5 seconds is available in 3DPos array form.*
Attributes read/used: *inputs, pitchPositions*
Methods called: *None*

Processing logic:
This method is called when the position information of the pitch in 5 seconds is available in file form. It takes in that file and assigns pitchFile with the file's 3DPos equivalent.

6. Method: *determine()*
Return Type: *void*
Parameters: *None*
Pre-condition: *the pitchPositions,*
Post-condition: *The result of a pitch is available.*
Attributes read/used: *pitchPositions, SZPosition, HPPosition*
Methods called: *None*

Processing logic:
It's essentially a setter method for the attribute "result". This method takes in the positions of the pitch, strike zone and home plate. Uses the home plate position for calibration. If any of the 5 3DPos objects in "pitchPositions" array is in the range of x, y and z coordinates of the "SZPosition". It's a strike and "result" will be assigned "True". If not, it's a ball and "result" will be assigned false.

7. Method: *WriteToDataCollector()*
Return Type: *void*
Parameters: *None*
Pre-condition: *The result of a pitch is available.*
Post-condition: The *numofBall or numofStrike in DataCollector class is increased.*

Attributes read/used: *DataCollector.numofBall, DataCollector.numofStrike*

Methods called:

DataCollector.getInstance(), DataCollector.getNumofBall(), DataCollector.getNumofStrike(), DataCollector.setNumofBall(int num), DataCollector.setNumofBall(int num)

Processing logic:
This function is called AFTER determine() is called. It will use the method in the DataCollector class to increment the value of either numofBall or numofStrike in the DataCollector object obtained by getInstance() method.

4.2.5 Class: DataCollector

Purpose: *To Collect stats (average number of balls/strikes, etc.) of a match and store them as a file that can be retrieved by the UserInterface class when needed.*

Constraints: *There's only one instance of DataCollector Class during a match.*

Attributes: *numofBall - the number of ball during a match*
numofStrike - the number of strike during a match.
matchID - ID of a match, used as the name of the record file.

Methods*:* *DataCollector(int matchID)*
static getInstance()
saveAsFile()
getNumofBall()
getNumofStrike()
setNumofBall(int num)
setNumofStrike(int num)
calculateAvgBall()
calculateAvgStrike()

4.2.5.1 Attribute Descriptions

1. Attribute: *numofBall*
   Type: *int*
   Accessibility: *Private*
   Description: *the total number of balls during a match.*
   Constraints: *cannot be negative*

2. Attribute: *numofStrike*
   Type: *int*
   Accessibility: *Private*
   Description: *the total number of strikes during a match.*
   Constraints: *cannot be negative*

3. Attribute: *matchID*
   Type: *std::string*
   Accessibility: *Private*
   Description: *The ID of a match, used as the name of the record file.*
   Constraints: *None*

4.2.5.2 Method Description

1. Method: *DataCollector(int matchID)*
   Return Type: *N/A*
   Parameters: *matchID – The name of the saved match file*
   Pre-condition: *Called by getInstance()*
   Post-condition: *A DataCollector object with a matchID is ready.*
   Attributes read/used: *None*
   Methods called: *None*

   Processing logic:
   This is a private constructor of Data. In order to meet the constraint, it should only be called by the getInstance() method.

2. Method: *getInstance()*
   Return Type:
   Parameters:*None*
   Pre-condition: *None*
   Post-condition: *A DataCollector object*
   Attributes read/used: *None*

Methods called: *DataCollector(matchID)*

Processing logic:
This is a static method that is designed to be used as the constructor of the DataCollection class. When calling this method, if there's an existing DataCollector object, return this object instead of a new DataColletor object. If there's not, a new object will be created.

3. Method: *saveAsFile()*
   Return Type: *void*
   Parameters:*None*
   Pre-condition: *a match is finished*
   Post-condition: *A file with the name same as matchID and stored the records of that match is created.*
   Attributes read/used: *None*
   Methods called: *calculateAvgBall(), calculateAvgStrike()*

   Processing logic:
   This method is called when the match is finished. It will create a file with the same name as the matchID, and write the average ball and average strike into the file.

4. Method: *getNumofBall()*
   Return Type: *int*
   Parameters:*None*
   Pre-condition: *None*
   Post-condition: *The current total number of balls is ready.*
   Attributes read/used: *numofBall*
   Methods called: *None*

   Processing logic:
   Getter method for *numofBall.*

5. Method: *getNumofStrike()*
   Return Type: *int*
   Parameters:*None*
   Pre-condition: *None*
   Post-condition: *The current total number of Strikes is ready.*
   Attributes read/used: *numofStrike*
   Methods called: *None*

   Processing logic:
   Getter method for *numofStrike.*

6. Method: *setNumofBall(int num)*
   Return Type: *void*
   Parameters: *num – the new number of total balls.*
   Pre-condition: *None*
   Post-condition: *The current total number of balls is updated.*
   Attributes read/used: *numofBall*
   Methods called: *None*

   Processing logic:
   Setter method for *numofBall.*

7. Method: *setNumofStrike(itn num)*
   Return Type: *void*
   Parameters: *num – the new number of total strikes.*
   Pre-condition: *None*
   Post-condition: *The current total number of strikes is updated.*
   Attributes read/used: *numofStrike*
   Methods called: *None*

   Processing logic:
   Setter method for *numofStrike.*

8. Method*: calculateAvgBall()*
   Return Type: *int*
   Parameters: *None*
   Pre-condition: *the match is finished.*
   Post-condition: *The*
   Attributes read/used: *numofStrike, numofBall*
   Methods called: *None*

   Processing logic:
   Calculate the *average percent of Ball by numofBall / (numofBall+ numofStrike). Return an int indicating the percentage. (E.g., 90 means 90% of all pitches are ball)*

9. Method*: calculateAvgBall()*
   Return Type: *int*
   Parameters: *None*
   Pre-condition: *the match is finished.*
   Post-condition: *The*
   Attributes read/used: *numofStrike,numofBall*
   Methods called: *None*

   Processing logic:
   Calculate the *average percent of strike by numofStrike / (numofBall+ numofStrike). Return an int indicating the percentage. (E.g., 90 means 90% of all pitches are strike)*

4.2.6 Class: UserInterface

Purpose: *Notify the user of the result of a pitch. Retrieve the corresponding match history file if the user asks.*
Constraints: *None*
Attributes: *None*
Methods*: retrieveRecords(std::string matchID), notify(bool result)*

4.2.6.1 Attribute Descriptions

Not Applicable for this class.

4.2.6.2 Method Description

1. Method: *retrieveRecords(std::string matchID)*
   Return Type: *File**
   Parameters: *matchID – The name of the saved match file*
   Pre-condition: *There's at least one match that is finished and saved as a file by the DataCollector* class.
   Post-condition: Record file of the selected match is available.
   Attributes read/used: *DataCollector.matchID*
   Methods called: *None*

   Processing logic:
   The User calls this method when they want to retrieve a record file of a match by its matchID. This method will try to locate and open a file with the name equal to matchID. If there's no qualified file, return NULL.

2. Method: *notify(bool result)*
   Return Type: *void*
   Parameters: *result– The result of a pitch.*
   Pre-condition: *the result of a pitch is available*.
   Post-condition: A message is sent to the user.
   Attributes read/used: *Determiner.result*
   Methods called: *Determiner.getResult()*

   Processing logic:
   The User calls this method when the result is ready. This method will print a corresponding message on the screen depending on the value of "result".

5 Dynamic Model

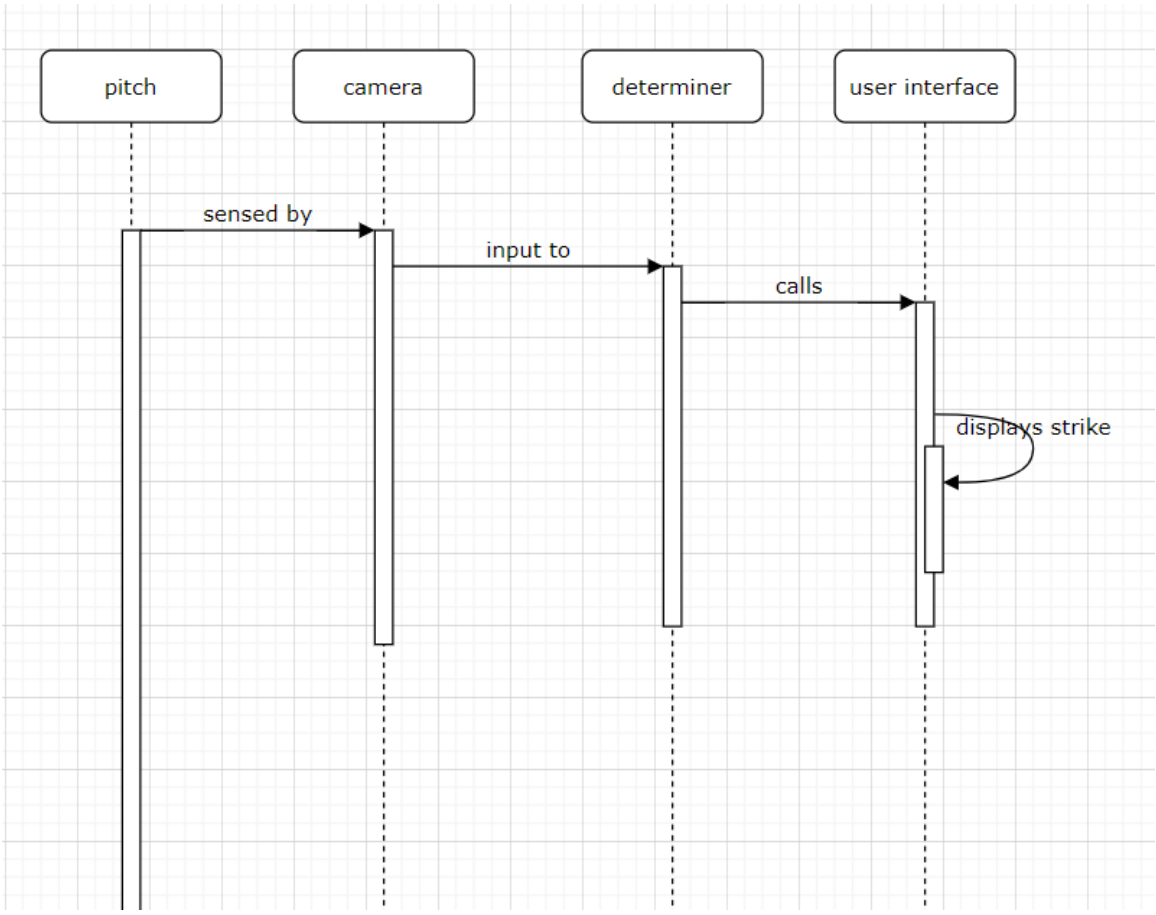
The purpose of this section is to model how the system responds to various events, i.e., model the system's behavior. We do this using UML sequence diagrams.

The first step is to identify different scenarios (e.g. Fuel Level Overshoots), making sure you address each use case in your requirements document. Do not invent scenarios, rather a general guideline is to include scenarios that would make sense to the customer. For example, for the course enrollment system, logging in is a valid scenario.

5.1 Scenarios

Scenario #1 - Pitch is a Strike

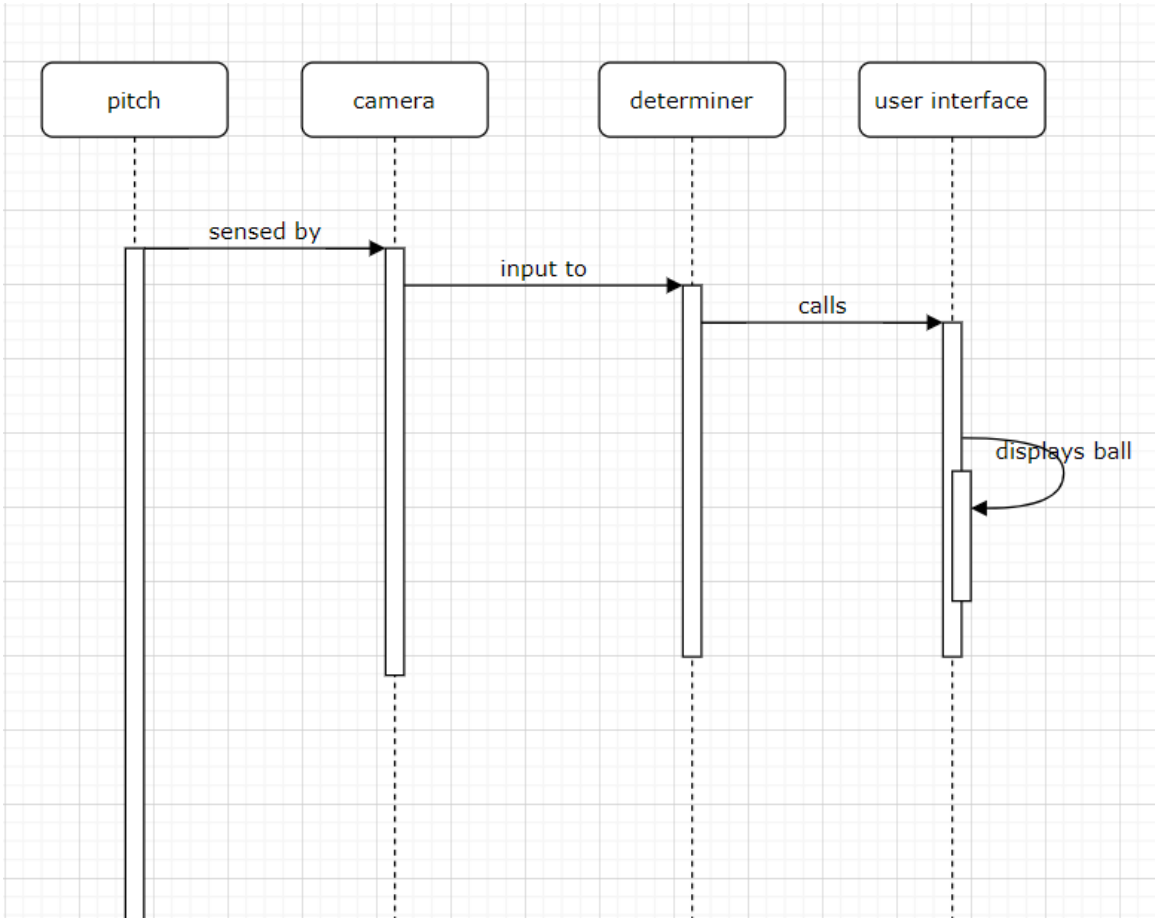
- *Scenario Name: This is the scenario where the pitch enters the strike zone*
- *Scenario Description: In this scenario, the SLOW-ARC system is able to calculate the pitch still. Since this is the case, the system will proceed as normal.*
- *Sequence Diagram:*



Scenario #2 - Pitch is a Ball

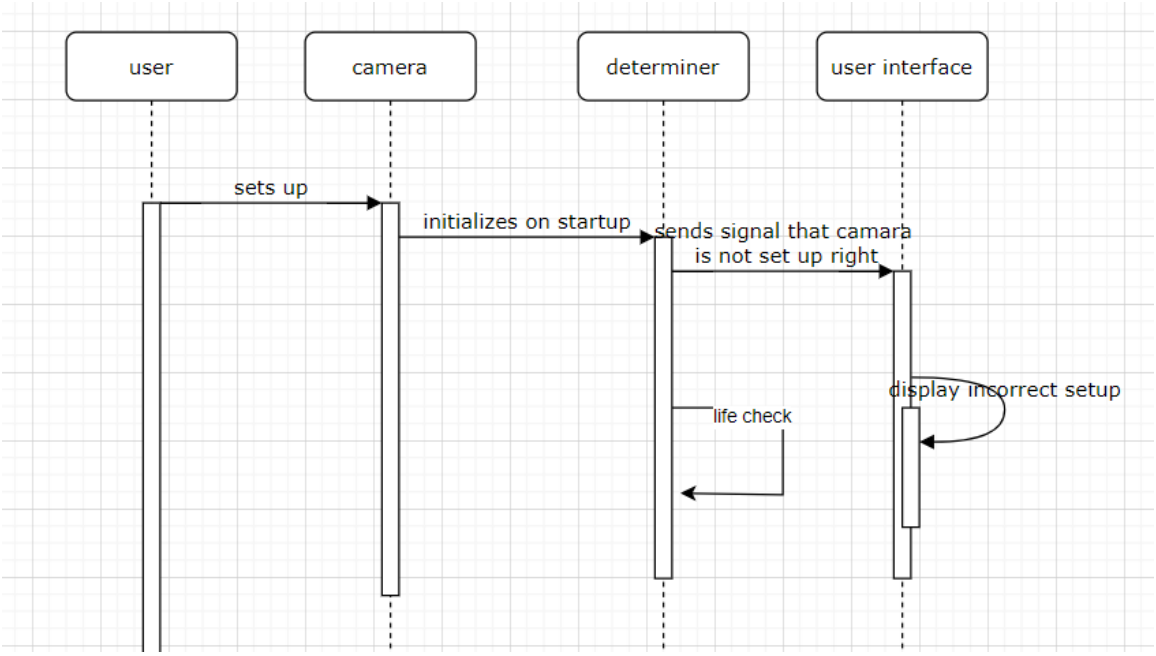
- *Scenario Name: This is the scenario where the pitch does not enter the strike zone*
- *Scenario Description: In this scenario, the SLOW-ARC system is able to calculate the pitch still. Since this is the case, the system will proceed as normal.*

- *Sequence Diagram:*



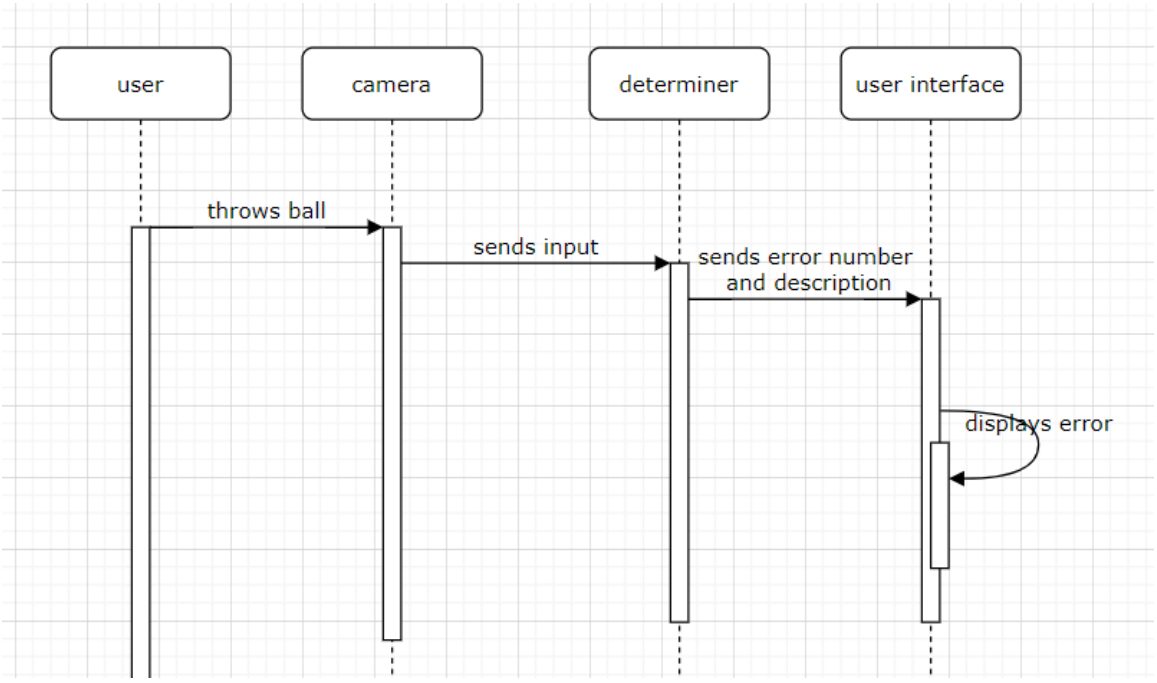
Scenario #3 - Cameras are unaligned

- *Scenario Name: Cameras are not set up properly*
- *Scenario Description: In this case the system senses that the camera is not set up correctly*
- *Sequence Diagram:*



Scenario #4 - Cameras Malfunction

- *Scenario Name: An unclassified error occurs with the cameras*
- *Scenario Description: Any unclassified error that causes an error exception in the system. If this occurs, the system will notify the user of the exception name*
- *Sequence Diagram:*



6 Non-functional requirements

Meeting non-functional requirements is a critical aspect of any system design, as these requirements often define the system's usability, efficiency, and effectiveness. Here's an approach to address the specified non-functional requirements:

Requirement: 95% Accuracy in Determining Pitches

Assessment and Approach

Algorithm Selection and Tuning: Choosing the right algorithm that can process pitch data with high accuracy is essential. This may involve machine learning or digital signal processing techniques known for their precision in audio analysis.
Data Quality and Preprocessing: Ensure that the input data (e.g., audio signals of pitches) is of high quality and free from noise. Implement preprocessing steps to enhance data quality which directly impacts accuracy.
Continuous Testing and Calibration: Regular testing with a diverse set of pitches and continuous calibration to account for environmental factors (like temperature, humidity) that might affect pitch detection.
Feedback Loop: Incorporate a user feedback mechanism to identify inaccuracies and continuously train the system to improve its performance.

Evolution and Future Adaptation

- Scalable Architecture: Design the system with a scalable architecture that can easily integrate advancements in audio processing technologies.
- Modular Design: Ensure components responsible for pitch detection are modular, allowing for easy updates or replacements as newer, more accurate methods are developed.

Requirement: Portability (Fit into a Compressed Car)

Assessment and Approach

Compact and Modular Design: All equipment should be designed to be compact, with a focus on modularity. This allows for easy disassembly and reassembly, making it more transportable.
Lightweight Materials: Use materials that are lightweight yet durable to reduce the overall weight, facilitating easier transport.
Efficient Packing Design: Develop a packing plan that utilizes space efficiently. Custom storage containers or bags can be designed to hold the equipment securely while optimizing space.

Evolution and Future Adaptation

- Versatility in Equipment Use: Design the system so that it can be adapted for use in different types of fields or environments, anticipating future needs.
- Expandability: Ensure the design allows for the addition of new components or technology without significantly affecting the portability.

Addressing External Factors

- Environmental Variability: Acknowledge that certain environmental conditions which are not under direct control might impact the system's performance (e.g., extreme weather conditions affecting pitch detection accuracy or portability).
- User Handling: User handling of the system can also impact its performance and portability. Providing comprehensive user manuals and training can mitigate this.

Conclusion

By carefully considering these approaches, the system can be designed to meet the current non-functional requirements while also being adaptable for future enhancements. Regular

reviews and updates based on technological advancements and user feedback will ensure the system remains relevant and effective.

7 Requirements Traceability Matrix

Requirement	Design Implementation
The camera setup will be capable of capturing data and output data of pitches during softball games in multiple forms of media.	the camera captures the visual data of the pitches and then sends this to the determination system to be interpreted
The system shall be able to track the x and y axis trajectory of the ball from the release of the pitch till the end of the current pitch. The system will be able to capture data that is distinguishable to the rules of USA softball calls of strikes and balls.	Use a custom data type called “3DPos”, which gives the x and y coordinate of the ball.
System shall continuously track the ball's movement from pitcher to batter.	The visual data resulted from the camera interface includes the ball all the way from pitch to bat.
System should store pitch data for subsequent analysis.	Database interface holds data that might not quite be useful immediately, but allows the option to be viewed from the user interface whenever it is needed
System must utilize sensors or technology to monitor batter's movements.	Use high-speed cameras to capture detailed video footage of the batter's movements. These cameras should be capable of recording at high frame rates to ensure smooth motion capture. Motion Sensors: Employ motion sensors, such as inertial measurement units (IMUs), attached to the batter's gear (like the bat, helmet, or gloves) to capture detailed movement data.
System should accurately measure the strike zone's dimensions.	The Camera class, with 3DPos data type, accurately measure the strike zone's dimensions.
Data about batter and strike zone should be readily available for the Pitch Evaluation System.	The Camera class makes the data available.
System shall analyze factors like pitch speed, trajectory, and location.	use a custom data called 3dpos which is collected from the cameras and determiner method uses it to calculate szposition and hp position to run algos to determine and

	analyze
System should generate evaluations assisting decision-making.	System sends a call signal to the ui and then the display shows the play from the ui.
Post-game pitch analysis insights should be available.	Given the pitch data collected the analysis according to US Softball Rulebook the display can display through the ui the statistics for pitches during games.
A user-friendly interface shall be developed for reviewing plays, changing camera angles, and accessing pitch data.	The ui will be developed to use joysticks and touch screen software.
Users shall be able to customize system settings based on their specific needs.	The ui can be calibrated suiting the needs of the player because of the getpos function of 3dpos which allows us to recalibrate with different data points as our base measures.
The system shall implement security measures to prevent unauthorized access.	Implement strong authentication protocols to prevent unauthorized access. Encryption: Use encryption to protect data in transit and at rest. Regular Security Updates: Plan for regular security updates and vulnerability assessments.
Ensure the system provides feedback for any user actions, including any errors or failures.	Ensure the interface provides immediate feedback for user actions, including visual and auditory cues for errors or confirmations.
The system shall provide a robust database system capable of handling large volumes of team and league data.	Choose a database architecture that can scale to handle large volumes of data. Performance Optimization: Optimize database performance for quick querying and data retrieval. 3dpos
Implement backup and recovery mechanisms for data security.	Implement automated regular backups of the database. Recovery Plan: Develop a comprehensive recovery plan in case of data loss.
Ensure database integrity through regular maintenance and checks.	Establish a regular maintenance schedule for the database. Integrity Checks: Implement tools for conducting regular integrity checks to ensure data accuracy and consistency.
Provide querying capabilities to extract specific data subsets for analysis.	Develop an advanced query interface that allows users to easily extract specific data subsets.Optimize queries for performance, especially when dealing with large datasets.

8 Appendices

Requirements Specification Document for SLOW-ARC: Copy of Slow Arc SRS DOC