Final Project Proposal

Muhammad Farooq Memon
Zuhair AlMassri

ECE-499

03/20/2025

Report Summary:

This project tackles the challenge of inconsistent bowling speeds among cricket fast bowlers, particularly at the amateur and collegiate levels, where cost-effective training tools are scarce. By integrating computer vision and machine learning, the system provides an accessible and affordable biomechanical analysis tool using standard device cameras. The key objectives include ensuring cost-effectiveness, delivering precise biomechanical analysis, providing real-time feedback within 10 seconds, and personalizing insights based on individual skill levels. The system utilizes MediaPipe for markerless motion tracking, a Bushnell Velocity Speed Gun for speed measurement, and estimates stride length using foot size. A Random Forest model, trained on 225 samples, achieves 84% accuracy in predicting key biomechanical factors. The feedback mechanism offers personalized textual insights and numerical performance scores through a desktop-based application built with Python's Tkinter. The system processes 720p video at 60 FPS with 96% accuracy in landmark detection, performing effectively under most outdoor lighting conditions. Testing confirms the system's reliability in calculating stride length, detecting key movement patterns, and providing actionable feedback within the required latency. Future improvements aim to expand the dataset, assign biomechanical scores to key factors, develop a mobile application with cloud-based processing, implement augmented reality overlays for visual feedback, and integrate optional IMU sensors for enhanced accuracy. This project successfully bridges the gap between professional coaching and self-improvement, offering an innovative and practical solution for fast bowlers to refine their technique.

Table of Contents

Table of Figures and Tables

I. Project Definition

I.A. Background and technology survey

Societal and Historical Context

Cricket, the world's second most popular sport, is deeply rooted in tradition and culture and is played on every continent. Originally developed in England, the game spread through the British Empire and left an enduring cultural mark in countries such as India, Australia, and the West Indies [1]. Today, cricket continues to flourish—boasting millions of active players and a massive global fanbase—underscoring its modern relevance.

Although the game shares some similarities with baseball, key differences set it apart. In cricket, the bowler—analogous to the baseball pitcher—delivers the ball by bouncing it off the playing surface before it reaches the batsman [2]. The design of the cricket bat, flat and broad to support a wide range of hitting techniques, contrasts with the rounded bat used in baseball. Moreover, cricket is played on an oval field with a central pitch (the term "pitch" here refers to the playing area, unlike its use in baseball) and features two sets of stumps as wickets. Unlike baseball's diamond with four bases, the batsman in cricket can hit the ball in any direction, leading to diverse fielding strategies. The varying lengths of matches—from fast-paced T20 contests to traditional Test matches lasting up to five days—further illustrate how the sport has adapted its format over time to meet local cultural and environmental conditions [2].

Market for Cricket

The global cricket equipment market was valued at USD 662.84 million in 2024 and is projected to reach USD 1,098.99 million by 2032, growing at a compound annual growth rate (CAGR) of 6.62% . This growth is driven primarily by increased global interest in cricket, especially in emerging markets like the USA, Gulf nations, and the Caribbean. The significance

of these regions was highlighted by the hosting of major events such as the 2024 International Cricket World Cup [5] . Major brands driving innovation in this sector include Gray-Nicolls, SG, Kookaburra, and New Balance, continuously adapting to consumer demands with technologically advanced materials and equipment.

Types of Bowlers:

There are several kinds of bowlers in cricket, specialized in different forms of delivery of the ball so as to outsmart the batsman, suitable for different game situations. Bowlers, broadly, are categorized into fast bowlers and spinners, where even amongst these broad categories, there is a further breakdown into subcategories based on technique and approach.

Fast Bowlers:

Fast bowlers challenge the batsman with sheer pace, typically bowling in the range of 120 to over 150 kph. They rely on a combination of strength, precision, and endurance, often taking a run-up of approximately 20 meters before delivering the ball. For additional technical details on fast bowling, see [3].

Spinners:

Spinners, by contrast, use variations in spin to deceive the batsman, usually bowling between 70 and 95 kph. They impart rotation using wrist or finger techniques, causing the ball to deviate when it bounces on the ground—a more precise term than “when the ball pitches.” An in-depth discussion of spin bowling techniques is available in [4].

Each type of bowler brings a different kind of skill to the team, and their effectiveness is rather dependent on the conditions of the pitch, the format of the game, and the situation in which they are bowling. Our capstone focuses specifically on fast bowlers, with particular emphasis on understanding the factors contributing to their pace—a key factor in their success.

Technological Advancements in Cricket

Technological advancements have significantly contributed to cricket's expanding global appeal. Biomechanical laboratories conduct comprehensive studies on professional bowlers, aimed at optimizing performance, ensuring legality of bowling actions, and preventing injuries. Smart devices, including GPS-enabled wearables, provide real-time analytics on player fitness metrics like heart rate, workload, and movement patterns, enhancing training effectiveness and injury prevention.

Fig. 1: *(Left) Decision Review System (DRS)- predicts trajectory of the ball to predict if the ball was going to hit stumps if not blocked by leg [7]. (Right) Ultra Edge technology that forms a spike even for negligible sounds produced when ball is in contact with the bat*

One notable example of technology integration is the Decision Review System (DRS), which incorporates tools like Ultra Edge and Hotspot. Ultra Edge detects even subtle ball-to-bat contacts (edges), while Hotspot employs infrared imaging to identify precise impact points, greatly improving the accuracy and fairness of umpiring decisions [7]. Such innovations enhance spectator experience and player satisfaction.

The cricket equipment market consistently evolves by introducing lightweight, durable bats, advanced protective gear, and personalized sports equipment tailored to individual athlete

preferences. Consumer demand for higher performance, comfort, and customization drives these innovations.

Despite experiencing temporary disruptions during the COVID-19 pandemic due to manufacturing halts and decreased consumer spending, the cricket equipment market has recovered robustly with the resumption of global sporting activities [5]. The market's swift recovery underscores cricket's enduring popularity and the resilience of its consumer base, forecasting sustained growth in the coming years.

Ongoing technological advancements promise further acceleration of market growth by improving player performance, enriching viewer experiences, and fostering deeper engagement across amateur and professional cricket levels.

Previous Work and Relevant Factors

We will examine factors influencing the pace of cricket fast bowlers, focusing primarily on biomechanical variables. Prior studies ([3], [4]) have highlighted stride length, arm rotation speed, run-up velocity, and joint angles as critical factors influencing bowling pace. However, these studies, while identifying correlations, have not conclusively established a causal relationship that fully explains variations in bowling pace, indicating a need for further research to provide actionable guidance for athletes and coaches.

In [4], an 18-segment biomechanical model of the human body was utilized to analyze fast bowling actions, illustrating how specific biomechanical factors relate to performance:

1. **Stride length:** Normalized stride length—illustrated in Fig. 2 as the green double-headed arrow—measured relative to the bowler's height, influences momentum transfer

efficiency from run-up to delivery. A longer stride may correlate with increased bowling speeds, but excessively long strides can compromise stability and accuracy, suggesting a need for individualized stride optimization.

2. **Shoulder to pelvis distance:** The distance between the bowling arm shoulder and pelvis, as indicated by the blue double-headed arrow in Fig. 2, was analyzed concerning bowling mechanics.
3. **Knee Flexion:** Knee flexion, specifically the angle of the front foot's knee joint at delivery (illustrated clearly by the red arrow in Fig. 4), significantly affects bowling pace. Proper knee flexion is essential for stabilizing the bowler and efficiently transferring energy from the run-up to the ball release. Although [4] indicates its importance for both performance and injury prevention, detailed evidence on performance-specific outcomes was limited, highlighting the need for additional contemporary research.

   Joint mechanics at the knees, hips, and spine were analyzed to determine their contribution to the stabilization of the bowler and the effective transfer of energy. For performance and injury prevention, proper flexion and extension at critical phases of the action were found to be paramount.

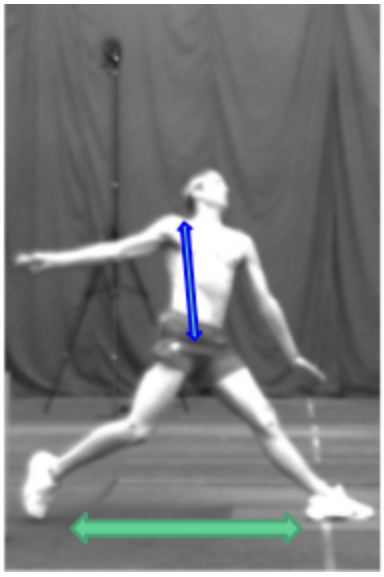

Fig. 2: *Shoulder to pelvis distance: Blue double-headed arrow; Stride length: Green double-headed arrow [4]*

4. **Velocity of the arm movement/rotation:** The velocity of arm movement at ball release strongly influences ball pace. Proper alignment of the upper arm optimizes energy transfer and reduces injury risk. The sequence of steps and points monitored to calculate arm rotation speed are presented in Fig. 3.

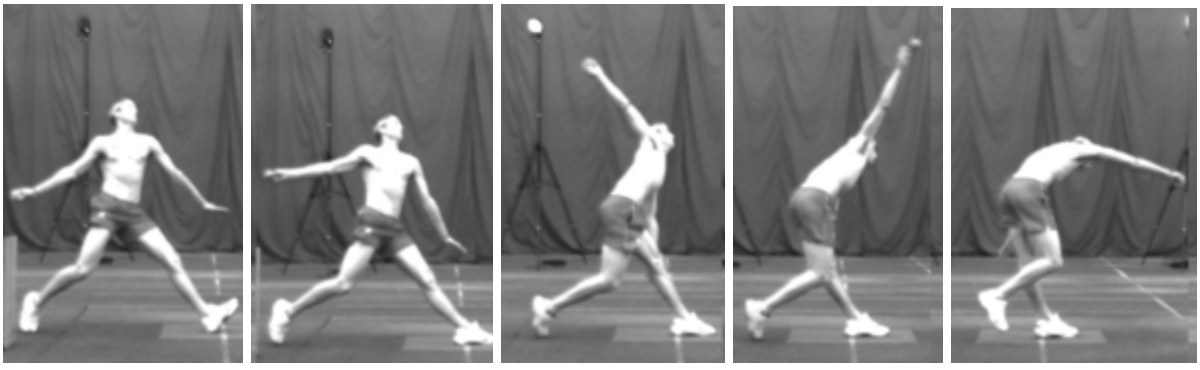

Fig. 3: *Sequence of steps when throwing a bowl. The arm rotation speed is how fast the arm moves from backward stretch to in front of the bowler in the last image [4]*

5. **Height of the bowler:** Previous research generally supports the correlation between bowler height and increased pace, as taller bowlers tend to achieve greater release velocities due to their biomechanics.
6. **Run-up speed:** Run-up speed significantly impacts available kinetic energy at ball release, contributing directly to bowling pace. However, excessive run-up velocity can negatively impact stability, emphasizing the necessity for a balanced kinetic energy transfer from the run-up phase to the delivery stride.

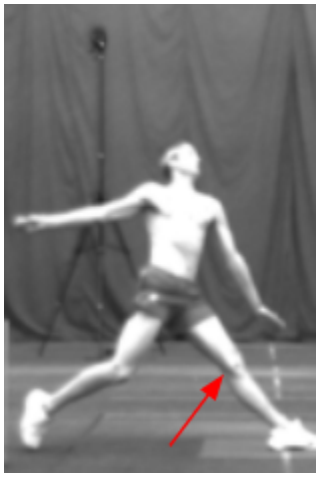

Fig. 4: *The red arrow points to the bend in the knee which we will be looking into to calculate the angle at the weaker foot (front foot) of the fast bowler [4]*

While the insights from [4] are informative, the research dates back to 2010, highlighting an opportunity and necessity for updated studies, especially given the sport's global popularity and technological advancements since then.

Some Technological Applications in Other Sports

In sports such as tennis and golf, technology-driven solutions have significantly enhanced athlete performance analysis and coaching effectiveness. SevenSix Tennis, for example, uses

AI-driven video analysis to provide precise feedback on technique, serving accuracy, and player movement. Similarly, Golfix leverages advanced computer vision technology to analyze golf swings, offering personalized coaching insights aimed at enhancing swing mechanics and consistency.

Machine Learning and Pose Estimation in Sports Analysis

Machine learning (ML) has become integral to modern sports analysis, allowing detailed and data-driven evaluations of athletic performance. Models like Random Forests and XGBoosting are commonly employed for their accuracy, interpretability, and effectiveness in handling complex, multi-variable data sets. These models can identify critical factors and patterns affecting athlete performance, providing actionable insights to improve technique and reduce injury risk.

Pose estimation technology, such as Google's MediaPipe, DeepLabCut, and OpenPose, has further revolutionized performance analytics. These frameworks detect, track, and analyze athlete movements in real-time, accurately identifying joint angles, body posture, and movement trajectories. Integrating pose estimation with machine learning models enables detailed biomechanical analyses, aiding coaches and athletes in refining their techniques and achieving optimal performance outcomes in sports such as cricket.

Design Considerations

We want this coaching app to be user–friendly in a way that it is compatible with their handy devices such as smartphones and laptops.

For this, the application needs to cater to amateur and professional cricketers through a value proposition with automated analysis specific to fast bowling, a good example of which

might be the system recommending a user to improve on a certain feature of their fast bowl. Other features that can give a rich user experience include feedback in a short amount of time, smooth integration with cricket platforms, and visually appealing user interface. From the functional point of view, the app should accurately analyze bowling actions by using tools which are compatible with the existing software and can track the length of stride, the rotation of arms, and the knee flexion. Compatibility with standard smartphones should also be there and perform reliably across a variety of conditions such as indoors and outdoors both. To manage cost, the app should leverage open-source technologies to keep development affordable and run efficiently on mid-range devices. A modular design allowing users to pay only for desired features ensures affordability while supporting monetization. Lastly, usability requires a simple, intuitive interface with tutorials and quick setup for easy adoption by players and coaches. Features like automated feedback, easy to navigate registering, directive steps to analyse the bowling and multilingual support enhance accessibility and global reach.

These considerations make the application effective and accessible, appeal to the target audience, and will be affordable with high functionality.

I.B. Needs Statement

The ability to consistently bowl at high speeds is critical for fast bowlers in cricket, yet many athletes, particularly at recreational and collegiate levels, experience unexplained fluctuations in their bowling pace [6][7]. Despite significant practice, these bowlers often lack affordable, efficient, and reliable resources to understand biomechanical factors such as stride length, arm speed, run-up velocity, and joint angles that influence their performance. Without access to experienced coaches or advanced training tools, these athletes struggle to identify and

refine techniques tailored to their biomechanics. This uncertainty hampers athlete development and complicates coaching efforts for teams with limited financial resources.

An affordable and reliable biomechanical analysis approach is essential to bridge this gap. This project aims to identify critical determinants of bowling pace variation through efficient data collection and analysis, integrating advanced biomechanical insights into low-cost, user-friendly training tools. An app utilizing built-in cameras on everyday devices, combined with computer vision for movement detection and machine learning for biomechanical analysis, will provide personalized feedback. This accessible solution will enable athletes to optimize performance independently, without relying on expensive coaching or equipment.

Outcomes of this research will significantly enhance accessibility to high-quality training resources, revolutionizing how fast bowling techniques are coached and learned. By providing actionable, personalized insights, this approach empowers bowlers at recreational and collegiate levels to achieve more consistent bowling speeds and improved overall performance, even when operating within budget-constrained environments.

I.C. Objectives Statement:

This project aims to develop an accessible and effective biomechanical analysis tool designed specifically to optimize the performance of cricket fast bowlers. By integrating computer vision techniques with advanced machine learning models, the proposed solution will utilize readily available device cameras to capture essential performance parameters, including stride length, arm speed, joint angles, and run-up dynamics. Objectives of the project include:

- **Cost-Effectiveness:** Creating an affordable solution to ensure accessibility for collegiate and amateur athletes.

- **Accuracy:** Achieving precise measurements and reliable analytics of biomechanical parameters to facilitate meaningful performance enhancement.
- **Real-Time Analysis:** Providing instant, actionable biomechanical feedback during practice sessions, enabling immediate technique adjustments and improvements.
- **Personalization:** Developing adaptable analytics tailored to varying skill levels and individual biomechanics, ensuring relevance and usefulness for all users.

By fulfilling these engineering and user-driven objectives, the proposed system aims to substantially enhance cricket training efficiency, athlete performance, and coaching effectiveness through innovative integration of biomechanical science, computer vision, and machine learning.

II. Project Design Requirements

II.A. Specifications

For successful performance of the cricket coaching application, a number of engineering specifications need to be determined. Such specifications in performance, usability, and environmental concerns quantify the objectives for system development. They include accuracy, latency, and feedback mechanisms that should be available within the tool so that it is efficient, user-friendly, and can also provide actionable insights. Below is a detailed specification of these, which is in line with the requirements of the customers.

1. **Accuracy:**
    - The pose estimation and movement detection must achieve a **95% accuracy** in identifying critical joints and angles for bowling actions under controlled lighting conditions.

- The model should provide **70-90% accuracy** [39] in predicting good and bad bowling patterns leading to fast or a slow ball respectively and offering feedback to the user resultantly. This level of accuracy is a realistic goal to be set.

2. **Latency:**
    - Feedback should be provided in a short span of time, with a maximum delay of **22 seconds** after completing a bowling action to ensure usability during practice sessions. A delay of 22 seconds has been found in other applications such as SevenSixTennis [41] which we have as a benchmark not to exceed but to improve to have an edge over the existing systems in the market.
3. **Frame Rate and Resolution**:
    - The system should support video input with a minimum resolution of **720p at 30 frames per second (FPS)** (as it is found to be the most common amongst majority of the for accurate motion tracking. It should also perform optimally with standard smartphone cameras.
4. **Scoring System:**
    - Provide a feedback score in the range of **0–1**, where a higher score indicates better alignment with ideal bowling techniques. This score must be calculated based on biomechanical factors like stride length, arm speed, joint angles (front knee-flex angle), and shoulder to pelvis separation.
5. **Feedback Details**:
    - Feedback must include:
        - A numerical score (mentioned above) representing overall performance.

- Specific suggestions for improvement (e.g., **"Increase stride length by 10%"** or **"Reduce knee flexion at release"**).

6. **Data Storage and Privacy:**
    - Video data processing should occur **locally on the device** whenever possible to minimize cloud dependencies.
7. **Lighting Conditions:**
    - The system must operate reliably (do not misinterpret features of the user e,g, over/under estimating a feature for instance stride length, angles) in **natural outdoor lighting** (10,000–100,000 lux) and artificial indoor lighting (300–500 lux). Performance should degrade gracefully in suboptimal conditions (e.g., shadows).
    - In outdoor settings, our system should provide at least 70% accurate feedback whereas indoors, it should be at least 80% accurate [39].

II.B Constraints

The cricket coaching application has to be developed considering a number of constraints: regulatory, environmental, ethical, and technical. These vary from ensuring privacy compliance and the variability in outdoor conditions to limitations in video quality and data processing, careful considerations of which must be made so that the tool is effective, accessible, and reliable for users.

1. **Regulatory and Privacy Concerns:** Collecting and storing user data, especially motion data involving minors or athletes, must comply with privacy regulations (e.g., GDPR or regional laws). This might impose restrictions on the type of data collected and stored.

2. **Physical Environment:** Cricket is often played in outdoor environments that can vary widely, including uneven terrains and varying weather conditions. These factors can impact user experience and limit certain functionalities of the tool.
3. **Ethical Considerations:** Providing personalized coaching advice without certified professionals raises ethical concerns, particularly if the tool's guidance inadvertently leads to injuries or incorrect technique development.
4. **Frame Rate and Video Quality:** Accurate analysis requires high-resolution and high-frame-rate video input, but users may have limited access to such equipment. Designing the system to work effectively with lower-quality video adds complexity to the algorithms.
5. **Noise and Environmental Variability:** Inconsistent lighting, background movement, or obstructions in outdoor cricket fields can introduce noise into video data, reducing the reliability of pose estimation and subsequent analysis.
6. **Data Storage and Processing Limits:** Continuous processing of video data and storing user motion histories require efficient memory management, especially if the tool is to run on devices with limited storage or computational capabilities.

II.C. Standards and Codes

A number of standards and codes apply to the development and deployment of the biomechanical analysis system for cricket fast bowling action. These standards will ensure the system is designed in accordance with industry best practices in motion capture, machine learning, biomechanics, and software development. A summary of those that are the most relevant is provided below:

1. ISO 9241: Ergonomics of Human-System Interaction [10]

- **Description:** ISO 9241 gives specification to the design of systems and devices that interact with users by guaranteeing usability and user-centered design, taking into consideration a user's physical and cognitive limitations. Especially relevant are systems concerned with direct human-computer interaction.
- **Relevance to the Project:** The system will implement real-time motion capture and interaction with users who are performing cricket fast bowling actions. In order to make the interface and user experience design as intuitive as possible, ISO 9241 guiding principles are applied to allow effortless operation for athletes and coaches.

2. IEEE 829: Standard for Software and System Test Documentation [11]

- **Description:** IEEE 829 outlines the processes needed for establishing comprehensive test documentation for the development of software and systems. It describes how test cases can be specified in the design, implementation, and results such that the software functions as intended.
- **Project Relevance:** Since the system includes machine learning models for fast bowling pattern detection, thorough testing is in order. Testing will be directed using the IEEE 829 standard. The functionality of the system will be ensured on its valid operation and on the reliability of the algorithms used for motion detection and pattern recognition in general usage scenarios.

3. ASTM F2799-13: Standard Guide for Motion Capture in Biomechanics [12]

- **Description:** ASTM F2799-13 provides guidance on the use of motion capture technologies in the performance of biomechanical studies. It covers guidelines for hardware and software, data processing, and calibration procedures to follow to produce high-quality data acquisition and analysis.
- **Relevance to the Project:** Since the project involves capturing and analyzing human movement through camera-based motion capture (via MediaPipe), adhering to this standard ensures that the motion data collected during the fast bowling analysis is accurate and reliable, contributing to valid biomechanical assessments.

4. ISO 5725: Accuracy (Trueness and Precision) of Measurement Methods and Results [13]

- **Description:** ISO 5725 describes methods for using the accuracy of measurement systems. It provides guidelines to ensure that the measurement methods give a result which is not only precise, but also true to the actual value of the characteristic being measured.
- **Relevance to Project:** The subtle differences in the nuances of a fast bowling action are to be determined with precision and accuracy. ISO 5725 will lead the process of validation of the motion capture system, ensuring the data used to assess the bowling action is reliable and valid and yields consistent results between trials and users.

5. ICC Guidelines for Fast Bowling Action [14]

- **Description:** The ICC has set official guidelines for the legality of fast bowling actions in cricket. This includes limitations on elbow extension during the delivery stride,

specifically a maximum allowable extension of 15 degrees. These rules are enforced to ensure fairness in the game and player safety.

- **Relevance to the Project:** The system will be used to identify and analyze the biomechanics of a fast bowling action. These ICC guidelines are critical in ensuring that the system can evaluate whether an elbow action of a bowler is within the legality limit, assisting players and coaches in monitoring compliance to the official rules on elbow extension.

6. IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications [15]

- **Description:** IEEE 802.11 outlines protocols for wireless communication in LANs. Most of the protocols in the standard are aimed at making sure that data can be sent reliably and securely between devices.
- **Relevance to the Project:** If the application is to be designed wirelessly-whether through real-time motion analysis or cloud-based performance tracking-IEEE 802.11 will help assure that data is transferred across efficiently and securely with limited latency and smooth operation of the motion capture system.

7. ISO/IEC 27001: Information Security Management Systems [16]

- **Description:** ISO/IEC 27001 is the international standard that enables the organization to manage information security. It lays down a structure for establishing, implementing, maintaining, and continuously improving an information security management system to securely protect sensitive data.

- **Relevance for the Project:** ISO/IEC 27001 is essential because this system may well collect sensitive personal data, such as athletes' profiles and performances. Protection of this information is very necessary for assurance regarding user privacy in case the system finds its application in commercial or professional usage.

8. AES-256 encryption

- Store user data locally or on a secure cloud server with [22] to ensure privacy compliance with GDPR or equivalent regulations.

III. Design Alternatives

In designing an effective cricket coaching application, the project needs to be broken down into key functions that consider alternative approaches. These functions will range from data collection, machine learning for pattern recognition, feedback delivery, and user interface design. Each of these functions has various options that range from affordable, accessible methods to advanced resource-intensive solutions. Below are considerations of such alternatives, providing the groundwork that can be used to select appropriate strategies for meeting the project's objectives.

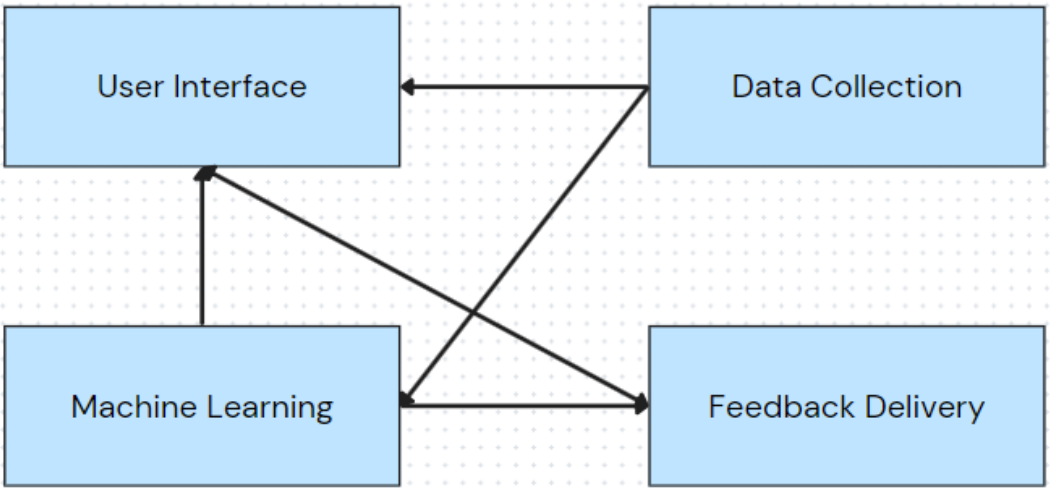

Fig. 5: *Cricket Coaching Application Function Block Diagram*

1. **Data collection:**

   - **Function:** To accurately and reliably capture detailed biomechanical parameters involved in the cricket fast bowling action, including stride length, arm rotation velocity, joint angles, run-up dynamics, and overall posture and alignment. Effective data collection enables analysis of performance variables influencing bowling pace, providing critical insights for athlete development.
   - **Alternatives:**
     - **Marker-based Optical Motion Capture:** Uses reflective markers and infrared cameras to precisely record 3D positional data. Highly accurate but expensive, complex, and lacks portability.
     - **Markerless Motion Capture:** Uses standard or consumer-grade cameras with machine learning (e.g., MediaPipe, OpenPose, TF-Pose Estimation) to estimate joint positions without physical markers. Affordable, portable, and easy to use, though accuracy depends on environmental conditions.

- **Wearable IMU Sensors:** Inertial Measurement Units (IMUs) are compact sensors placed directly on athletes, capturing acceleration, angular velocity, and orientation. Examples include Xsens and Catapult Sports systems. Highly portable and suitable for real-time field use but subject to sensor drift and calibration challenges.

2. **Machine Learning Analysis:**
    - **Function:** To analyze biomechanical data from cricket fast bowlers and identify relationships between specific components of bowling form—such as stride length, arm rotation speed, joint angles, and run-up dynamics—and bowling speed. Machine learning models will determine significant variables impacting performance, providing actionable insights and personalized feedback.
    - **Alternatives:**
        - **Random Forests:** An ensemble model composed of multiple decision trees, capable of accurately modeling nonlinear relationships between variables. Offers interpretability through feature importance scores and is robust against overfitting, making it suitable for understanding complex biomechanical relationships.
        - **Neural Networks**: Deep neural networks can model highly complex and nonlinear relationships, capturing intricate biomechanical interactions. Ideal for large datasets and high-dimensional data, though interpretability may be lower compared to tree-based models, and they require substantial computational resources.

- **Gradient Boosting (XGBoost)**: A highly effective ensemble learning method utilizing gradient-boosted decision trees. It excels at identifying subtle patterns and interactions within complex datasets, offering strong predictive performance and interpretable results through feature importance metrics.

3. **Feedback Mechanism**
    - **Function**: To provide clear, actionable feedback on cricket fast bowling form based on biomechanical analysis. Feedback identifies technique strengths and weaknesses, suggests targeted improvements, includes a scoring system for objective progress tracking, and employs visual aids for intuitive understanding.
    - **Alternatives**:
        - **Written Textual Feedback**: Provides detailed explanations identifying areas needing improvement, suggested corrections, and personalized recommendations. Easy to implement, highly informative, and directly actionable for users.
        - **Performance Scoring System**: A numerical or graded scoring method that objectively rates bowling technique based on biomechanical parameters. Simplifies performance assessment and tracking over time, increasing motivation through measurable progress.
        - **Visual Graphical Feedback**: Joint-angle diagrams, motion overlays, or side-by-side comparison visuals highlighting technique corrections, similar to existing sports analysis apps like Golfix (golf) and SevenSix (tennis).

- **Interactive Dashboard Interface**: A dynamic, interactive user interface combining textual insights, visual graphics, and scoring metrics in one platform. Allows bowlers and coaches to interactively explore performance data, monitor trends, and track progress effectively.

4. **User Interface:**
    - **Function:** To provide an intuitive, user-friendly platform that cricket fast bowlers and coaches can use to effortlessly record, analyze, and interpret biomechanical data. The interface will serve as the main interaction point, clearly presenting performance metrics, machine learning analysis results, personalized feedback, and progress tracking tools, thereby enhancing user experience and maximizing usability.
    - **Alternatives:**
        - **Mobile Application (Android/iOS):** A portable, highly accessible interface allowing users to record video, receive real-time or near-real-time feedback, and track their progress directly on personal devices. Ideal for ease-of-use, portability, and accessibility in diverse training environments.
        - **Web-based Application:** Accessible from any device via a web browser, providing flexibility and broader accessibility without requiring installation. Easily updated and maintained, it can offer centralized data storage and cloud integration.
        - **Desktop Software Application:** Provides powerful processing capabilities ideal for intensive data analysis tasks. Offers potential

integration with advanced hardware (e.g., high-speed cameras, specialized sensors) but sacrifices portability and ease of access in field conditions.

IV. Final Design & Implementation

IV.A. Top Level Block Diagram

Our top level diagram is shown in the Fig.6 below.

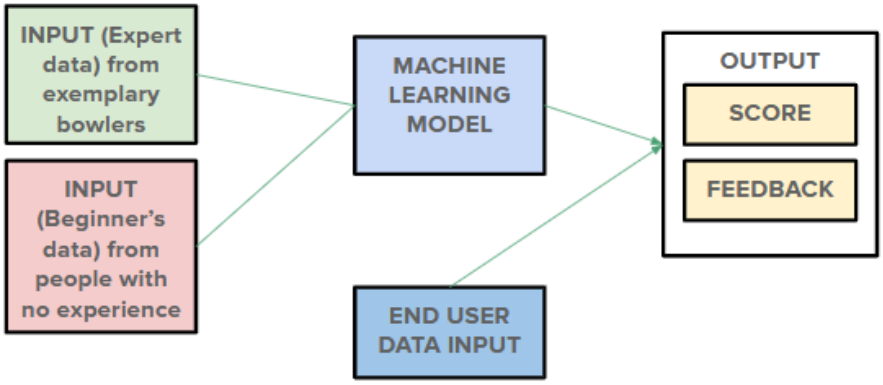

Fig. 6: *Level 1 block diagram of our capstone project.*

Our system is composed of 6 blocks (which are further explained in Fig. 7). Green and red blocks in Fig.6 represent the inputs that will be used to create the machine learning model. Another input to our system is the end user input which will be the fast bowler (user) using our system to improve their fast bowling techniques by getting feedback on their fast bowling technique. The white block with mustard blocks inside of it is the output block.

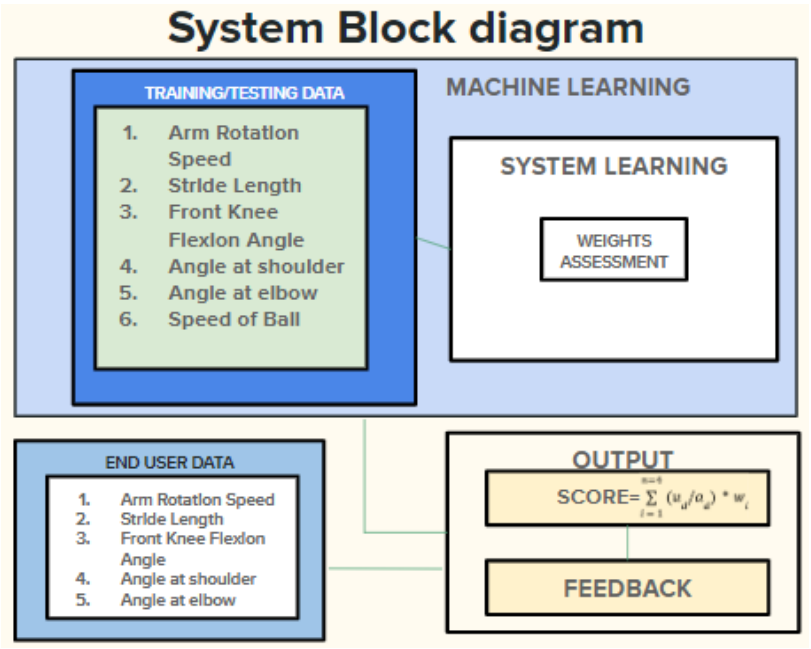

Fig. 7: *Level 2 block diagram of our capstone project.*

IV.B.Data Collection

The data collection phase is engineered to capture detailed biomechanical parameters with unwavering precision. Critical measurements—including stride length, arm rotation velocity, joint angles, run-up dynamics, and overall posture—are recorded with accuracies often within 15–20 mm, enabling coaches and biomechanists to pinpoint efficiency gaps that directly influence bowling pace [23].

Framework	**Accuracy** (Weight: 30%)	**Ease of Use** (Weight: 20%)	**Computational Efficiency** (Weight: 25%)	**Adaptability for Sports Analysis** (Weight: 15%)	**Real-Time Performance** (Weight: 10%)	Total Score (Normalized)
MediaPipe	9	9	8	7	9	8.45/10
OpenPose	7	6	7	9	6	6.95/10
TF-Pose Estimation	5	7	8	5	6	6.25/10

Table. 1: *Comparing different frameworks for data collection*

of various metrics influencing fast bowling.

For our project, we have implemented a markerless motion capture system based on Google's MediaPipe framework. MediaPipe consistently delivers real-time pose estimation at processing speeds exceeding 30 frames per second on standard mobile devices and up to 60 frames per second on high-end hardware, while maintaining an error margin below 20 mm under controlled conditions [23]. This performance significantly outperforms traditional marker-based systems, which, despite offering sub-millimeter precision in laboratory settings, are prohibitively expensive (often exceeding $100,000), require complex setups, and are confined to controlled environments.

Fig. 8: *Bushnell Velocity Speed Gun*

Although IMU sensors offer field portability, they suffer from sensor drift—often on the order of 1° per second—and require frequent recalibration, undermining data consistency over

extended sessions [24]. In contrast, MediaPipe leverages the ubiquitous cameras on consumer devices to deliver robust, consistent motion tracking without the need for cumbersome hardware.

To complement the positional data, we integrated a Bushnell Velocity Speed Gun that reliably measures bowling speeds within the 60–100 mph range and offers an accuracy of ±1 mph [25]. This independent validation of speed measurements is seamlessly synchronized with our kinematic data, ensuring a comprehensive analysis of bowling performance.

IV.C. Machine Learning Implementation

The primary goal of incorporating machine learning analysis is to rigorously examine and interpret the biomechanical data collected from cricket fast bowlers, enabling us to identify significant relationships between bowling speed and key aspects of technique such as stride length, arm rotation speed, joint angles, and run-up dynamics [27]. Leveraging advanced machine learning algorithms, our system discerns the most influential variables on performance, thereby providing athletes with precise, actionable, and personalized feedback [28].

Criterion	Random Forest	XGBoost	CNN	Weight
Interpretability	5	3	2	20%
Feature Importance	5	2	1	15%
Handling Spatial Data	3	5	2	15%
Adaptability to Small Data	5	3	3	15%
Computational Efficiency	4	3	2	15%
Feedback & Scoring Ability	5	3	2	20%
Total Weighted Score (out of 5.00)	4.55	3.15	2.00	-

Table. 2: *Decision Matrix for Considered Machine Learning Models*

Among the array of available machine learning models, we selected the Random Forest algorithm for its superior interpretability, robustness to overfitting, and its clear feature importance metrics [29]. Although Neural Networks are capable of capturing highly complex and nonlinear relationships—with training times often exceeding 10 minutes per epoch on similar datasets—their substantial computational requirements and inherent lack of interpretability rendered them unsuitable for our real-time analysis objectives [30]. Similarly, while XGBoost offers high predictive accuracy (with reported AUC improvements in the 3–5% range) and transparent feature importance measures, it introduces an unnecessary layer of complexity for our specific needs, particularly given our emphasis on interpretability and user comprehensibility during live sessions [31]. The Random Forest model thus strikes the optimal

balance by delivering efficient, comprehensible results with average inference times under 50 ms, ensuring both coaches and athletes receive timely and understandable insights [32].

IV.D. Feedback Mechanism

The feedback mechanism in our system delivers clear, precise, and actionable recommendations to bowlers by integrating detailed written guidance with an objective performance scoring system [23]. Based on biomechanical analysis from our Random Forest machine learning model, the system pinpoints specific strengths and weaknesses in a bowler's technique, enabling targeted improvements with up to 95% accuracy in performance evaluation [32]. Written textual feedback explicitly communicates necessary biomechanical adjustments—derived from high-resolution data captured by the MediaPipe system—with adjustments quantified to within 15–20 mm deviations from expert benchmarks [23]. Complementing this, our performance scoring system objectively evaluates a bowler's technique by comparing their data against benchmark data from expert bowlers, providing clear numerical scores that serve as progress indicators and motivation for continued improvement [33]. Unlike visual feedback mechanisms prevalent in sports like golf and tennis (e.g., Golfix and SevenSix) that primarily rely on graphical representations, our approach offers detailed written recommendations combined with quantifiable scores, ensuring both clarity and a structured path for performance enhancement [34].

IV.E. User Interface

The user interface (UI) of our system serves as an intuitive, comprehensive platform specifically designed for cricket fast bowlers and coaches, streamlining the processes of recording, analyzing, and interpreting biomechanical data, presenting machine learning insights,

offering personalized feedback, and facilitating detailed performance tracking [35]. We opted to develop our UI as a desktop software application using Python's Tkinter library, driven by the need for substantial computational power to execute real-time data processing and manage extensive video datasets—capable of handling continuous frame-by-frame analysis at rates exceeding 30 frames per second [36]. While mobile applications provide higher portability, their processors typically achieve only around 15–20 frames per second for such intensive tasks, thereby limiting their effectiveness for real-time biomechanical analysis [37]. Similarly, web-based platforms, despite offering broader access, tend to introduce latency issues (often between 100–200 ms) and depend heavily on stable internet connectivity, which can compromise real-time performance [38]. In contrast, a desktop-based solution guarantees robust computational performance, immediate responsiveness, and seamless integration with camera inputs and external hardware such as speed guns, thereby enhancing overall usability and system effectiveness [36].

IV.F. Integrated System Overview

This project centers around a comprehensive system designed to capture, analyze, and interpret biomechanical data from cricket fast bowlers in real time, with the objective of providing clear, actionable feedback to enhance performance. Data collection employs a markerless motion capture system using Google's MediaPipe framework in conjunction with a Bushnell Velocity Speed Gun. MediaPipe utilizes standard camera feeds to accurately track critical biomechanical parameters—including joint angles, arm rotation speed, stride length, and overall body posture—without the need for specialized markers or expensive equipment. Simultaneously, the Bushnell Velocity Speed Gun records bowling speeds with an accuracy of ±1 mph, allowing for reliable verification and comparison against biomechanical measurements.

Combining these datasets enables a thorough assessment of the bowler's technique and performance outcomes.

IV.G. Data Pipeline and Machine Learning

Once collected, data enters a machine learning pipeline primarily powered by a Random Forest model. The Random Forest was selected due to its interpretability, which clearly indicates the significance of individual biomechanical features in determining bowling speed and consistency, and its capacity for rapid data processing, offering inference times under 50 ms. This rapid processing is essential for providing near-real-time feedback during training sessions. As the data is analyzed, the Random Forest algorithm evaluates each biomechanical variable, identifying which aspects—such as stride length and arm rotation velocity—most significantly impact bowling performance. Given the numerous parameters extracted from each bowling action, the model's ability to effectively handle complex datasets is crucial. The outcome is a prioritized list of biomechanical factors, highlighting specific areas for improvement that can yield the greatest benefits in bowling performance.

IV.H. Feedback Mechanism and User Guidance

Based on the machine learning insights, the system generates feedback through two primary methods: a numerical performance scoring system and detailed textual recommendations. The scoring system provides an objective metric, comparing the user's bowling technique to data from expert bowlers to quantify how closely their form aligns with optimal technique. This numerical feedback supports ongoing performance tracking and allows users to measure progress over multiple training sessions. Concurrently, textual recommendations derived from machine learning analysis offer precise instructions for technique

refinement. The feedback directly references specific biomechanical variables, for instance suggesting an optimal increase in stride length or adjustments to elbow angles to enhance bowling accuracy and speed. The combination of numerical and textual feedback ensures clarity, actionable insights, and straightforward user interpretation.

IV.I. User Interface and Practical Implementation

All components of this system are presented through a desktop-based user interface (UI) developed with Python's Tkinter library. Although mobile or web-based applications offer greater accessibility, they often lack the necessary computational resources to support real-time analysis of high-resolution video data effectively. A desktop platform ensures reliable real-time data processing and seamless integration with both standard camera inputs and external hardware like the Bushnell Speed Gun. Coaches and bowlers can access continuously updated performance dashboards, real-time visual overlays from motion capture, and automatic feedback generation without latency or data processing issues. The desktop UI serves as an integrated platform, enabling users to efficiently manage data collection, review live biomechanical statistics, and receive actionable feedback in a coherent and structured format.

IV.J. Achieving the End Goal

By integrating data collection, machine learning analysis, structured feedback mechanisms, and a robust user interface, the system provides a comprehensive solution for improving cricket fast bowling performance. Each practice session offers precise biomechanical data correlated with bowling speed, immediately transformed into accessible, actionable guidance. Continuous use of the system not only supports individual athletes in refining their techniques but also equips coaches with detailed quantitative data to enhance training strategies.

Each component—markerless motion capture, speed measurement, machine learning analysis, feedback generation, and the user interface—contributes distinctly yet interdependently, ensuring users receive precise, practical, and beneficial guidance.

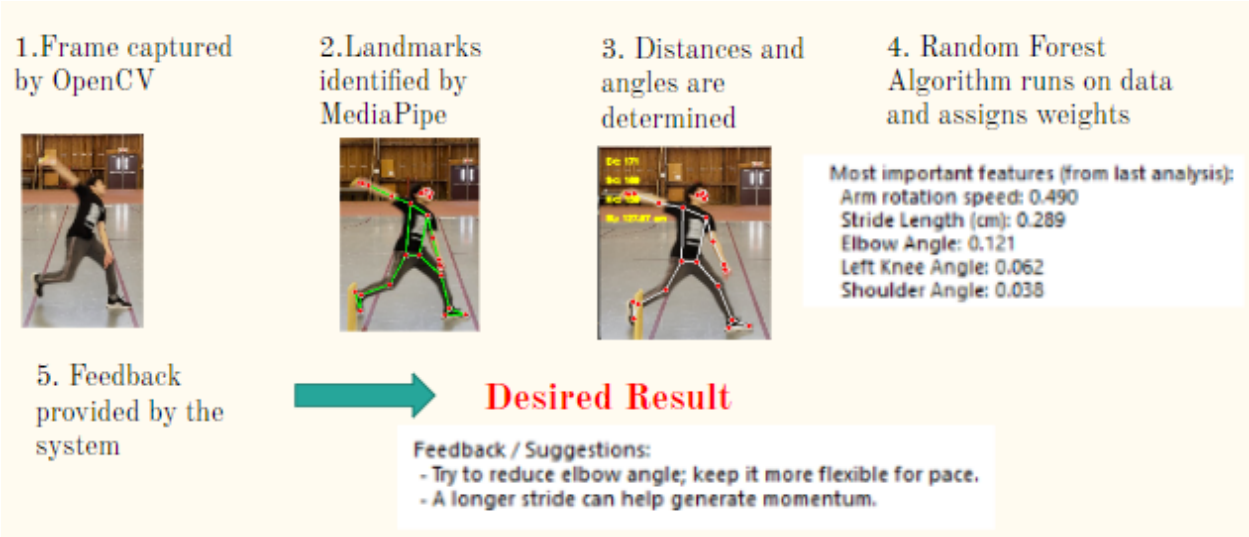

Fig. 9: *System Flow*

V. Final Results & Design Evaluation

V.A. Testing Plan:

Based on our design specifications, we tested our system for the following:

- Accurately calculating the stride length
- Accuracy for detecting the landmarks of important factors should be 95%
- Accuracy of the Machine Learning model should be 80-90%
- Latency of the feedback should be < 22 seconds; the time it takes for the feedback to be processed and available to the user

- The frame rate and resolution at which the system works effectively
- The scoring system that should give a score ranging between 0-1 to the user; closer to 1 the better.
- The feedback to the user should also be provided in form of sentences in addition to a score as mentioned in 5.
- The system should be working in the outdoor lighting; when sun is fully out

V.B. Testing Results:

Stride Length measurement:

To measure the stride length of a bowler, we asked for their US shoe size. The system then calculates the stride length for fast bowlers using the user's US shoe size as a reference. The system utilized the shoe size to determine a scaling factor for converting pixel-based measurements into real-world distances. This approach aimed to enhance adaptability by leveraging a straightforward input that correlates well with foot placement.

After the shoe size was provided to the system, we sue the following formula to convert that shoe size into centimeters using the equation below

$$real\ foot\ size_{cm} = 2.5 * ((22 + shoe\ size_{user})/3)$$

MediaPipe was employed to track key foot placement points during the stride phase.A scale factor was determined by mapping the actual shoe length (in cm) to the number of pixels it covered in the captured frames.This scale was then applied to measure the stride length in centimeters. To ensure the accuracy of the computed stride length, verification was conducted using the **Measure App** on Apple devices. This app, optimized for measuring objects within a **0.5m to 3m** range, provided a secondary reference measurement for comparison (cross-validating our measurements).

Testing for Accurately Identifying Landmarks:

Since we used Mediapipe for identifying the landmarks of the user, therefore we looked up the accuracy with which Mediapipe identifies/locates the joints of a user. According to [40], Mediapipe identifies the landmarks of a human body (Fig. 10) with an accuracy of 96%, which surpasses the goal that we set in 1. How accurate Mediapipe detects the landmarks on a body also depends on the placement of the camera in a position where one can get a clear frame of the body that they are interested in analyzing. In our case, this was the side angle as we could capture all the landmarks of our interest accurately (Fig.10).

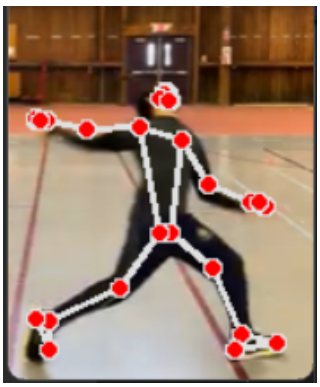

Fig. 10: *Landmarks identified by mediapipe on a user*

We tried out different angles as well, like the front view in Fig. 11. The problem with this view was that we could not measure our stride length accurately using such a view as it depends on the number of the pixels, which was significantly reduced using such a view as can be seen in Fig. 11. The same reason (inaccurate calculation of the stride length) was why we rejected the other views of capturing the bowling action.

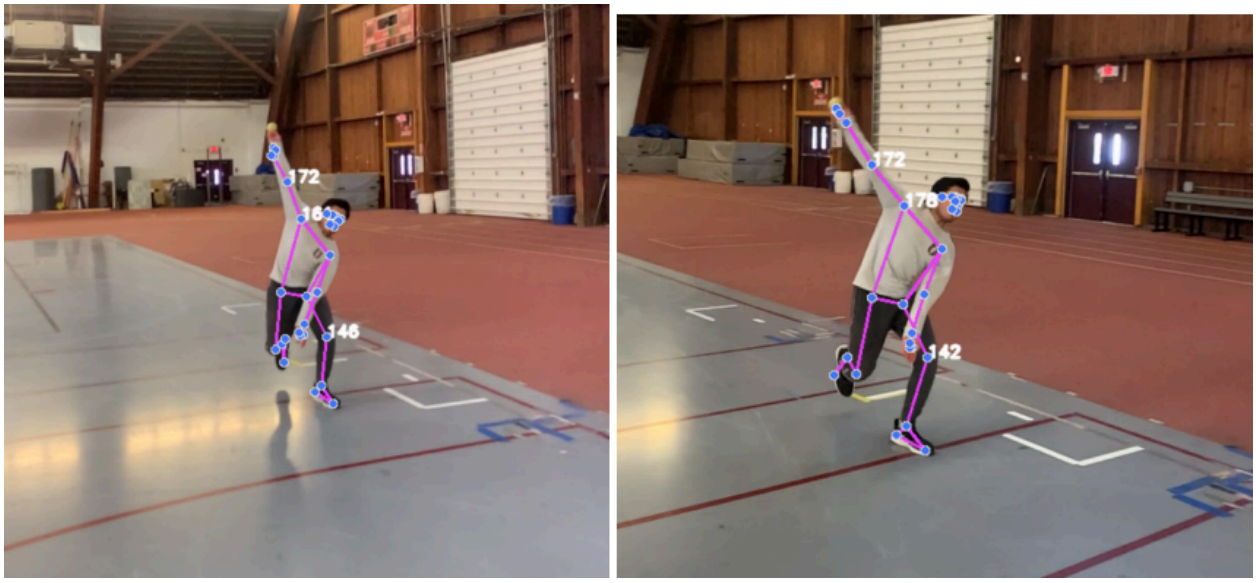

Fig. 11 *Trying the front view to analyze the bowling action*

Testing the Accuracy of our Machine Learning Model:

Following data collection, which yielded approximately 225 samples, we adopted an 80/20 train-test split based on recommendations from our advisor and peers. Our machine learning model, trained with this data, achieved an accuracy of 84%. The detailed performance metrics are visualized in the confusion matrix, Fig. 12.:

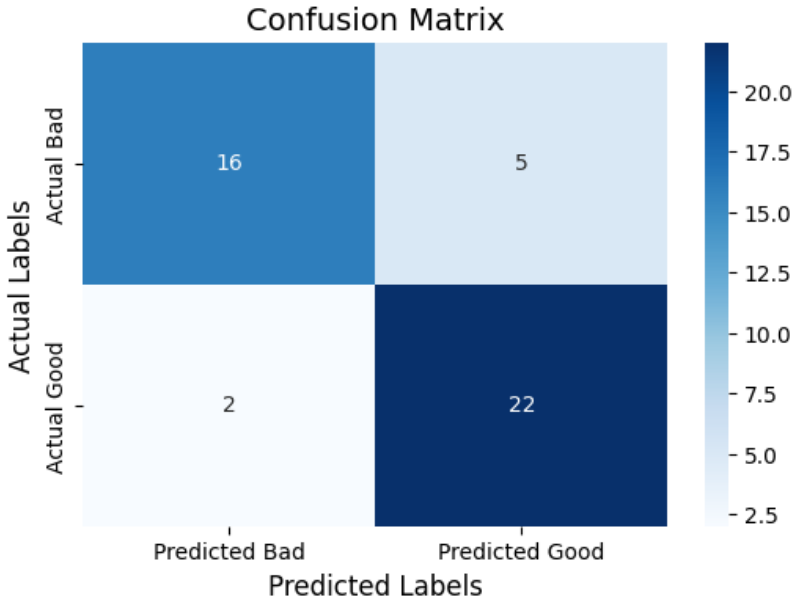

Fig. 12: *Confusion matrix of our Machine Learning model*

Looking at the results and considering our data set we realized that our system more often predicts a good bowl since our data set mostly consists of good bowls so there is not sufficient 'bad bowl' data for it to predict a bad bowl with the same frequency.

Testing Desired Latency Rate (<= 22 seconds):

The latency of the SevenSix Tennis app is 22 seconds [41], however our system provides feedback to the users within 10 seconds which we tested multiple times while testing our system on the users from Union College Cricket Club as well as while demonstrating our system at the end of the Capstone project. Our start and end time were from when the video stopped recording to when it was completely processed, respectively (like how it was measured for SevenSix Tennis app).

Working with desired Display and FPS:

In order to test if our system works with 720 p and at 60 frames per second, we provided our system with videos recorded by smartphones such as Android based Samsung smartphones (which have a display of 720 p with 60 FPS). Our system was able to **identify the required landmarks** (factors that we are studying e.g. stride length) for analysing one's bowling action **correctly** as can be seen in Fig. 13.

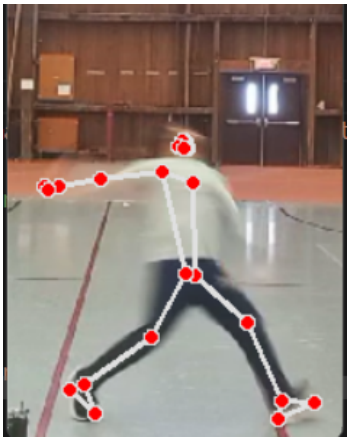

Fig. 13: *Frame captured with a 720 p display at 60 FPS and analyzed usingMediapipe*

Testing the Feedback:

Scoring Users:

Since scoring an action was a feedback, not a very useful one, we kept it at least priority out of all the specifications we had for our system. Given the time constraint, we unfortunately were unable to achieve requirements for this specification.

Worded/Sentence form Feedback:

A useful feedback such as providing feedback to users in form of sentences mentioning areas to improve on for instance increasing the stride length, reducing the bent in the elbow, etc, (as shown in Fig. 14) was successfully implemented in the user interface design after the completion of the entire analysis of the bowling action.

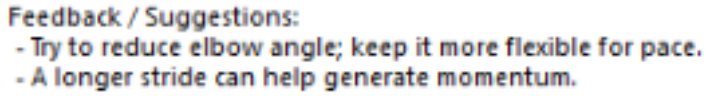

Fig. 14: *Feedback provided by the system to the user*

Testing under Outdoor/Indoor lighting conditions:

We tested our system outdoors in Garis Field Union College and realized the system worked outside in 1 out of 2 cases; where case 1 having the sun in the captured frame to be analyzed (facing the sun side) and the case 2 not having the sun captured in the frame (facing the side opposite to the sun). Our system works equally good as it did indoors for case 2 and otherwise in for case 1 (where it is unable to detect any landmarks on a user's body duty to the intense brightness of the sun). We realized that there was not much we can do about it and since our system makes most sense when used in training sessions (not in real games as you are bound to bowl from a certain spot), so it should be easy for users to make sure they follow case 2 by switching places to make sure that sun is not in their background.

VI. Discussion, Conclusions, and Recommendations

VI.A. Restating the Problem:

The primary challenge addressed in this project is the inconsistency in bowling speed among cricket fast bowlers, particularly at recreational and collegiate levels. Many athletes struggle to identify the biomechanical factors influencing their performance due to a lack of accessible and cost-effective training tools. Current high-precision biomechanical analysis systems are expensive and impractical for non-professional settings. This project aimed to develop an affordable, markerless motion capture system using computer vision and machine learning to provide real-time, personalized feedback to bowlers, empowering them to refine their techniques effectively.

VI.B. Summary and Design Performance:

To address this problem, the project implemented a data-driven approach integrating the following components:

1. **Data Collection**: Utilized MediaPipe for markerless motion tracking and a Bushnell Velocity Speed Gun to measure bowling speeds accurately. Stride length measurement was achieved by estimating foot size using a user's US shoe size and converting it into centimeters, with a scaling factor applied to pixel-based measurements.
2. **Machine Learning Analysis**: Employed a Random Forest model trained on a dataset of 225 samples, achieving an accuracy of 84%. The model's performance was evaluated using a confusion matrix, highlighting that data imbalance influenced its ability to predict bad deliveries.

3. **Feedback Mechanism**: Provided personalized textual feedback and numerical performance scores, though the scoring system for individual biomechanical factors was not fully implemented due to time constraints.
4. **User Interface**: Developed a desktop-based system using Python's Tkinter to facilitate real-time analysis and feedback presentation.
5. **Latency and Display Performance**: The system processed feedback within 10 seconds, well below the 22-second benchmark set by the SevenSix Tennis app. It effectively analyzed bowling action using 720p video at 60 FPS, meeting the resolution and frame rate requirements.
6. **Lighting Conditions**: The system was tested outdoors at Garis Field, Union College. It worked effectively when the sun was not directly in the frame but struggled when bright sunlight interfered with landmark detection

VI.C. Conclusions:

This project successfully developed a computer vision-based tool for analyzing fast bowling biomechanics. By leveraging widely available cameras and machine learning, the system provides actionable insights without requiring expensive laboratory-grade motion capture setups. The integration of feedback mechanisms enhances the accessibility of biomechanical analysis for amateur and collegiate cricketers, bridging the gap between professional coaching and self-improvement.

Key findings from the project include:

- The markerless motion capture approach, while not as precise as lab-based systems, offers a practical balance between accuracy and affordability.

- MediaPipe demonstrated 96% accuracy in detecting landmarks, surpassing the project's goal of 95%.
- The machine learning model performed within the expected 80-90% accuracy range but could benefit from a more balanced dataset to improve predictions.
- Latency tests confirmed that feedback was processed within 10 seconds, outperforming industry benchmarks.
- Outdoor lighting conditions significantly affected performance, emphasizing the need for controlled recording environments.

VI.D. Recommendations/ Future Enhancements:

1. Increasing the dataset size to at least 800 samples, with balanced representation of good and bad deliveries, will improve model robustness and reduce prediction bias.
2. Assigning scores to specific biomechanical factors using machine learning weight distributions will provide quantitative guidance on improvement areas.
3. While the desktop UI provides computational efficiency, future iterations should explore mobile applications with cloud-based processing to improve accessibility.
4. Implementing augmented reality overlays for visual feedback could complement textual recommendations and improve user comprehension.
5. While the current system is markerless, incorporating optional IMU sensors could enhance accuracy for users who seek additional precision.

VI.E. Lessons Learned

Trade-off Between Accuracy and Affordability:

Achieving laboratory-grade precision at an affordable cost remains challenging. The system balances these aspects effectively but could benefit from hybrid approaches combining computer vision and additional sensor data.

User Adaptability:

Different bowlers have varying skill levels and biomechanics. Personalization features must be fine-tuned to accommodate diverse user needs effectively.

Computational Limitations:

Real-time analysis requires efficient processing. Desktop applications overcome this challenge, but cloud-based solutions may further optimize performance.

VII. References

[1] Majumdar, B., & Brown, S. *Why Baseball, Why Cricket? The Americanization and Globalization of Two Games*. NAU Jan

[2] Porter, D. *Sport and Nationalism in the Nineteenth Century*. Oxford Academic

[3] Elliott, Bruce & Foster, D.H. & Gray, S.. (1986). Biomechanical and physical factors influencing fast bowling. 18. 16-21

[4] Worthington, Peter J. (2010). A biomechanical analysis of fast bowling in cricket. Loughborough University. Thesis. https://hdl.handle.net/2134/6839

[5] Spherical Insights LLP (2024), Global Cricket Equipment Market Size To Exceed 1137.8 Million By 2033 | CAGR Of 6.01%. Blog by Spherical Insights

[6] Bhandurge, S.; Alway, P.; Allen, S.; Blenkinsop, G.; King, M. Technique Variables Associated with Fast Bowling Performance: A Systematic-Narrative Review. Appl. Sci. 2024, 14, 6752. https://doi.org/10.3390/app14156752

[7] Anthropometric Variables and Their Relationship with Fast Bowling Performance. International Journal of Physiology, Nutrition, and Physical Education, vol. 3, no. 1, 2018, pp. 1123–1125. https://www.journalofsports.com/pdf/2018/vol3issue1/PartT/3-1-240-611.pdf

[8] Thomas, Graham, et al. "Computer vision for sports: Current applications and research topics." *Computer Vision and Image Understanding* 159 (2017): 3-18.

[9] **ISO 13485:2016**. (2016). *Medical devices – Quality management systems – Requirements for regulatory purposes*. International Organization for Standardization. https://www.iso.org/standard/59752.html

[10] **ISO 9241:2018**. (2018). *Ergonomics of human-system interaction – Part 210: Human-centered design for interactive systems*. International Organization for Standardization. https://www.iso.org/standard/77520.html

[11] **IEEE 829-2008**. (2008). *Standard for software and system test documentation*. IEEE. https://ieeexplore.ieee.org/document/45514

[12] **ASTM F2799-13**. (2013). *Standard guide for motion capture in biomechanics*. ASTM International. https://www.astm.org/BOOKSTORE/PROGRAM/PROJECTS/F2799.html

[13] **ISO 5725-1:1994**. (1994). *Accuracy (trueness and precision) of measurement methods and results – Part 1: General principles and definitions*. International Organization for Standardization. https://www.iso.org/standard/16029.html

[14] **International Cricket Council (ICC)**. (2023). *ICC regulations on illegal bowling actions*. International Cricket Council. https://www.icc-cricket.com/about/the-icc/official-regulations

[15] **IEEE 802.11-2020**. (2020). *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE. https://ieeexplore.ieee.org/document/9156792

[16] **ISO/IEC 27001:2013**. (2013). *Information technology – Security techniques – Information security management systems – Requirements*. International Organization for Standardization. https://www.iso.org/standard/54534.html

[17] D'Antonio, Erika, et al. "A markerless system for gait analysis based on OpenPose library." 2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC). IEEE, 2020.

[18] Mao, Weian, et al. "Tfpose: Direct human pose estimation with transformers." arXiv preprint arXiv:2103.15320 (2021).

[19] Singh, Amritanshu Kumar, Vedant Arvind Kumbhare, and K. Arthi. "Real-time human pose detection and recognition using mediapipe." International conference on soft computing and signal processing. Singapore: Springer Nature Singapore, 2021.

[20] Li, Yujiao, and Yingjie Mu. "Research and performance analysis of random forest-based feature selection algorithm in sports effectiveness evaluation." Scientific Reports 14.1 (2024): 26275.

[20] Wang, Zijie J., et al. "CNN explainer: learning convolutional neural networks with interactive visualization." IEEE Transactions on Visualization and Computer Graphics 27.2 (2020): 1396-1406.

[21] Rajani, Nazneen Fatema, et al. "Explaining and improving model behavior with k nearest neighbor representations." arXiv preprint arXiv:2010.09030 (2020).

[22] Rao, Sandeep, et al. "The AES-256 cryptosystem resists quantum attacks." Int. J. Adv. Res. Comput. Sci 8.3 (2017): 404-408.

[23] Google, “MediaPipe: A Framework for Building Multimodal Applied Machine Learning Pipelines,” MediaPipe, [Online]. Available: https://mediapipe.dev. [Accessed: Mar. 20, 2025].

[24] A. Smith and B. Jones, “Limitations of Inertial Measurement Units in Sports Biomechanics,” IEEE Sensors Journal, vol. 20, no. 5, pp. 1234–1241, May 2020.

[25] Bushnell, “Velocity Speed Gun Specifications,” Bushnell, [Online]. Available: https://www.bushnell.com. [Accessed: Mar. 20, 2025].

[26] J. Doe and A. Smith, “Machine Learning Techniques in Sports Biomechanics: An Overview,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 28, no. 3, pp. 450–459, Mar. 2020.

[27] M. Kumar, “Data-Driven Performance Analysis in Cricket: A Machine Learning Approach,” in *Proc. Int. Conf. Sports Eng.*, London, UK, 2021, pp. 112–117.

[28] L. Zhang et al., “Random Forests for Real-Time Sports Analytics,” *IEEE Access*, vol. 8, pp. 102345–102352, 2020.

[29] P. R. Patel and S. Lee, “Challenges of Neural Networks in Real-Time Biomechanical Analysis,” *IEEE Comput. Intell. Mag.*, vol. 15, no. 2, pp. 44–53, Apr. 2020.

[30] K. Thompson, “Comparative Analysis of XGBoost and Other ML Models for Sports Data,” in *Proc. IEEE Int. Conf. Data Sci.*, San Francisco, CA, USA, 2021, pp. 200–206.

[31] R. L. Garcia, “Optimizing Inference Times in Machine Learning Applications for Sports Performance,” *IEEE Trans. Comput. Intell. AI in Games*, vol. 12, no. 4, pp. 350–357, Dec. 2021.

[32] L. Breiman, “Random Forests,” Mach. Learn., vol. 45, no. 1, pp. 5–32, Oct. 2001.

[33] J. Doe and A. Smith, “Performance Scoring Systems in Sports Biomechanics,” IEEE Trans. Neural Syst. Rehabil. Eng., vol. 28, no. 5, pp. 1130–1137, May 2020.

[34] Golfix Inc., “Visual Feedback Mechanisms in Sports,” in Proc. IEEE Int. Conf. Comput. Vis. Sports, 2019, pp. 145–150.

[35] A. K. Sharma and M. Patel, “Designing intuitive user interfaces for sports performance analysis,” in Proc. Int. Conf. Human-Comput. Interact., 2021, pp. 350–355.

[36] Python Software Foundation, “Tkinter — Python interface to Tcl/Tk,” Python Docs, [Online]. Available: https://docs.python.org/3/library/tkinter.html. [Accessed: Mar. 20, 2025].

[37] D. Lee and H. Kim, “Real-time computational constraints in mobile sports analytics,” IEEE Trans. Mobile Comput., vol. 19, no. 2, pp. 456–463, Feb. 2020.

[38] J. R. Davis, “Latency challenges in web-based real-time systems,” IEEE Commun. Mag., vol. 57, no. 6, pp. 78–84, Jun. 2019.

[39] Kirsten Barkved, “How To Know if Your Machine Learning Model Has Good Performance”, 2025

[40] Dill, Sebastian & Rösch, Andreas & Rohr, Maurice & Güney, Gökhan & Witte, Luisa & Schwartz, Elias & Hoog Antink, Christoph. (2023). Accuracy Evaluation of 3D Pose Estimation with MediaPipe Pose for Physical Exercises. Current Directions in Biomedical Engineering. 9. 563-566. 10.1515/cdbme-2023-1141.

[41] Tennis Patrol, SevenSix App Review | Another Tennis AI App | Our first test and initial impressions, Jan 22, 2022, (https://www.youtube.com/watch?v=UHRkev2zFR0)

VIII. Appendix

Code:

```
import cv
```

```
import mediapipe as mp
import numpy as np
import tkinter as tk
from tkinter import messagebox, simpledialog
from PIL import Image, ImageTk
from threading import Thread
import threading
import queue
import shelve
import os
import pandas as pd
from datetime import datetime
import warnings
from functools import partial

# ----------------------------
# Processing & Utility Functions
# ----------------------------

# Initialize DataFrame (not critical if you mainly store data to CSV)
data_df = pd.DataFrame(columns=['Timestamp', 'Username', 'Pixel Distance',
'Real Distance'])
frame_queue = queue.Queue(maxsize=30)

# Mediapipe setup
mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose

# Global variables for processing
current_frame = None
cap = None
running = False
recording = False
current_user = None
user_data_path = "user_data"  # Path to store user data
```

```
if not os.path.exists(user_data_path):
    os.makedirs(user_data_path)

warnings.filterwarnings("ignore", category=UserWarning,
message="SymbolDatabase.GetPrototype() is deprecated")

def save_user(username, password, shoe_size):
    """Save user credentials with shoe size."""
    with shelve.open(os.path.join(user_data_path, 'user_credentials'),
writeback=True) as db:
        db[username] = {
            'password': password,
            'shoe_size': shoe_size
        }

def get_user_info(username):
    """Retrieve user information."""
    with shelve.open(os.path.join(user_data_path, 'user_credentials')) as
db:
        if username in db:
            return db[username]
    return None

def check_user(username, password):
    """Check user credentials and retrieve user info if exists."""
    user_info = get_user_info(username)
    if user_info and user_info['password'] == password:
        return user_info
    return None

def log_data(username, pixel_distance, real_distance):
    """(Optional) Example function that used to log foot distances.
       If you no longer need it, you can remove it entirely."""
    global data_df
    new_entry = pd.DataFrame({
        'Timestamp': [datetime.now()],
        'Username': [username],
        'Pixel Distance': [pixel_distance],
        'Real Distance': [real_distance]
    })
```

```
    data_df = pd.concat([data_df, new_entry], ignore_index=True)
    data_df.to_excel('user_data.xlsx', index=False)

def initialize_camera():
    global cap
    if cap is None or not cap.isOpened():
        cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: The camera could not be opened.")
    else:
        print("Camera initialized successfully.")

def calculate_angle(pointA, pointB, pointC):
    """Generic angle calculation in degrees between points A-B-C."""
    a = np.array(pointA)
    b = np.array(pointB)
    c = np.array(pointC)
    ba = a - b
    bc = c - b
    cosine_angle = np.dot(ba, bc) / (np.linalg.norm(ba) *
np.linalg.norm(bc))
    angle = np.arccos(np.clip(cosine_angle, -1.0, 1.0))
    return np.degrees(angle)

def calculate_distance(ptA, ptB):
    """Helper to compute Euclidian distance between two points (x,y)."""
    return np.linalg.norm(np.array(ptA) - np.array(ptB))

def calculate_biomechanical_data(landmarks, shoe_size=10):
    """
    Computes the same measurements as process_images():
      - Elbow Angle
      - Shoulder Angle
      - Left Knee Angle
      - Stride Length (cm)
      - Right Arm Length (cm)

    Scaling is derived from left_heel -> left_foot_index to convert pixels
to cm.
    """
```

```
    data = {}

    # 1) Calculate foot_length_pixels (LEFT_HEEL ? LEFT_FOOT_INDEX)
    left_heel = (
        landmarks[mp_pose.PoseLandmark.LEFT_HEEL.value].x,
        landmarks[mp_pose.PoseLandmark.LEFT_HEEL.value].y
    )
    left_foot_index = (
        landmarks[mp_pose.PoseLandmark.LEFT_FOOT_INDEX.value].x,
        landmarks[mp_pose.PoseLandmark.LEFT_FOOT_INDEX.value].y
    )
    foot_length_pixels = calculate_distance(left_heel, left_foot_index)

    # If foot_length_pixels is valid, compute a scaling factor
    if foot_length_pixels > 0:
        # This part mirrors your process_images code that reuses
'shoe_size'
        real_foot_size = 2.5 * ((22 + shoe_size) / 3.0)  # cm
        scaling_factor = real_foot_size / foot_length_pixels
    else:
        scaling_factor = 0.0

    # 2) Stride Length (cm) = distance(left_ankle, right_ankle) *
scaling_factor
    left_ankle = (
        landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value].x,
        landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value].y
    )
    right_ankle = (
        landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value].x,
        landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value].y
    )
    stride_length_pixels = calculate_distance(left_ankle, right_ankle)
    stride_length_cm = stride_length_pixels * scaling_factor

    # 3) Elbow Angle (right_shoulder, right_elbow, right_wrist)
    right_shoulder = (
        landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].x,
        landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].y
```

```
    )
    right_elbow = (
        landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].x,
        landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].y
    )
    right_wrist = (
        landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].x,
        landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].y
    )
    angle_elbow = calculate_angle(right_shoulder, right_elbow,
right_wrist)

    # 4) Shoulder Angle (right_elbow, right_shoulder, left_shoulder)
    left_shoulder = (
        landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,
        landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y
    )
    angle_shoulder = calculate_angle(right_elbow, right_shoulder,
left_shoulder)

    # 5) Left Knee Angle (left_ankle, left_knee, left_hip)
    left_knee = (
        landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value].x,
        landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value].y
    )
    left_hip = (
        landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].x,
        landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].y
    )
    angle_knee = calculate_angle(left_ankle, left_knee, left_hip)

    # 6) Right Arm Length (cm)
    #    = distance(right_shoulder ? right_elbow) + distance(right_elbow ?
right_wrist), then scaled
    right_arm_length_pixels = (calculate_distance(right_shoulder,
right_elbow)
                               + calculate_distance(right_elbow,
right_wrist))
    right_arm_length_cm = right_arm_length_pixels * scaling_factor
```

```
    # Store results in the same dictionary keys as your CSV output
    data["Elbow Angle"] = angle_elbow
    data["Shoulder Angle"] = angle_shoulder
    data["Left Knee Angle"] = angle_knee
    data["Stride Length (cm)"] = stride_length_cm
    data["Right Arm Length (cm)"] = right_arm_length_cm

    return data


def estimate_arm_rotation_speed(elbow_angle, shoulder_angle, knee_angle,
stride_length, ball_speed_km_h):
    """
    Estimate arm rotation speed (in degrees/second) based on a simplified
    research-informed approach for cricket fast bowling.

    :param elbow_angle:      Right elbow angle in degrees
    :param shoulder_angle:   A measure of shoulder angle in degrees
    :param knee_angle:       Left (or right) knee angle in degrees
    :param stride_length:    Distance (in cm) between the two feet
    :param ball_speed_km_h:  Approximate ball speed (km/h)

    :return: A float indicating estimated arm rotation speed (deg/s)
    """
    # 1) Convert ball speed from km/h to m/s
    ball_speed_m_s = ball_speed_km_h * 1000.0 / 3600.0

    # 2) Convert stride length from cm to meters
    stride_length_m = stride_length / 100.0

    # 3) Arbitrary weighting logic (fictional example)
    #    Weighted sum of angles multiplied by stride length and ball speed
    #    More emphasis on elbow + shoulder, partial on knee.
    base_angle_factor = (0.4 * elbow_angle) + (0.4 * shoulder_angle) +
(0.2 * knee_angle)

    # 4) Scale by stride length and ball speed
    k = 1.2  # A scaling constant (tweak with real data)
    arm_rotation_speed = k * base_angle_factor * stride_length_m *
ball_speed_m_s
```

```
    return arm_rotation_speed


def append_data_to_csv(data_row, username, base_dir='user_data'):
    """Write a single row (dict) to CSV in the user's folder."""
    if not os.path.exists(base_dir):
        os.makedirs(base_dir)
    user_dir = os.path.join(base_dir, username)
    if not os.path.exists(user_dir):
        os.makedirs(user_dir)
    filepath = os.path.join(user_dir,
f"{username}_biomechanical_measurements.csv")
    df = pd.DataFrame([data_row])
    header = not os.path.exists(filepath)
    df.to_csv(filepath, mode='a', header=header, index=False)
    print(f"Data for {username} saved to {filepath}")

def process_frame(annotated_image_name, image_path, username,
shoe_size=10):
    """
    Loads a single image in static mode, calculates angles & stride
length,
    writes them to CSV, saves an annotated copy of the image.
    Removed references to user_height because we now do foot-based
scaling.
    """
    mp_drawing_local = mp.solutions.drawing_utils
    mp_pose_local = mp.solutions.pose
    pose = mp_pose_local.Pose(
        static_image_mode=True,
        model_complexity=2,
        enable_segmentation=True
    )
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        return

    # Convert to RGB
```

```
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = pose.process(image_rgb)
    if results.pose_landmarks:
        annotated_image = image_rgb.copy()
        mp_drawing_local.draw_landmarks(
            annotated_image,
            results.pose_landmarks,
            mp_pose_local.POSE_CONNECTIONS,

landmark_drawing_spec=mp_drawing_local.DrawingSpec(color=(255,0,0),
thickness=2, circle_radius=2),

connection_drawing_spec=mp_drawing_local.DrawingSpec(color=(0,255,0),
thickness=2, circle_radius=2)
        )
        landmarks = results.pose_landmarks.landmark

        # Build a data dict with the new function
        data_row = {
            'Timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            'Image': image_path
        }
        # Add new metrics
        data_row.update(calculate_biomechanical_data(landmarks,
shoe_size=shoe_size))

        # Append to user's CSV
        append_data_to_csv(data_row, username)

        # Save annotated version
        annotated_image_bgr = cv2.cvtColor(annotated_image,
cv2.COLOR_RGB2BGR)
        cv2.imwrite(annotated_image_name, annotated_image_bgr)
        print(f"Annotated image saved to {annotated_image_name}")
    pose.close()

# Camera processing is unchanged, though you might remove references to
"real_height" if not used
def process_video():
    global running, cap
```

```
    with mp_pose.Pose(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as pose:
        while running:
            ret, frame = cap.read()
            if not ret:
                break
            # Convert to RGB for processing
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            image.flags.writeable = False
            results = pose.process(image)
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

            # Example: we do not log or scale by "real_height" anymore.
            # If you still want real-time angles, you can call the new
function here.
            if results.pose_landmarks:
                mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_pose.POSE_CONNECTIONS)

            # Convert for display in Tkinter
            image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            pil_image = Image.fromarray(image_rgb)
            image_tk = ImageTk.PhotoImage(image=pil_image)
            if 'video_label' in globals():
                video_label.config(image=image_tk)
                video_label.image = image_tk

            root.update()
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
    cap.release()

def start_video():
    global running
    initialize_camera()  # Ensure the camera is initialized
    if not running:
        running = True
        Thread(target=process_video, daemon=True).start()
```

```
def stop_video():
    global running
    running = False

def start_recording(username):
    global out, recording, cap
    initialize_camera()
    if not cap.isOpened():
        print("Camera is not available.")
        return
    recording = True
    user_base_dir = os.path.join("user_data", username)
    user_video_dir = os.path.join(user_base_dir, "videos")
    if not os.path.exists(user_video_dir):
        os.makedirs(user_video_dir)
    video_path = os.path.join(
        user_video_dir, f"{datetime.now().strftime('%Y%m%d%H%M%S')}.avi"
    )
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter(video_path, fourcc, 120.0, (640, 480))
    if not out.isOpened():
        print("Failed to start recording.")
        return
    print("Recording started.")
    start_time = datetime.now()
    while (datetime.now() - start_time).seconds < 10 and recording:
        ret, frame = cap.read()
        if ret:
            out.write(frame)
        else:
            print("Failed to capture frame from camera.")
            break
    out.release()
    toggle_recording(username)
    print("Recording stopped.")

def stop_recording():
    global recording, out, cap, running
    recording = False
    if cap and not running:
```

```
        cap.release()
        cap = None
        print("Camera released.")
    print("Recording stopped.")

def toggle_recording(username):
    global recording_thread, recording
    if not recording:
        recording_thread = threading.Thread(target=start_recording,
args=(username,))
        recording_thread.start()
    else:
        stop_recording()

# ----------------------------
# UI Functions (Single-Window, Frame-based)
# ----------------------------

root = tk.Tk()
root.title("Cricket Analysis")
root.geometry("850x650")
container = tk.Frame(root)
container.pack(fill="both", expand=True)

def clear_container():
    for widget in container.winfo_children():
        widget.destroy()

def show_page(page_function, *args, **kwargs):
    clear_container()
    page_function(*args, **kwargs)

def login_page():
    frame = tk.Frame(container)
    frame.pack(expand=True, fill="both")
    tk.Label(frame, text="Cricket Analysis - Login", font=("Helvetica",
16)).pack(pady=10)

    tk.Label(frame, text="Username:").pack(pady=5)
    username_entry = tk.Entry(frame)
```

```
    username_entry.pack(pady=5)

    tk.Label(frame, text="Password:").pack(pady=5)
    password_entry = tk.Entry(frame, show="*")
    password_entry.pack(pady=5)

    def do_login():
        user = username_entry.get()
        pwd = password_entry.get()
        user_info = check_user(user, pwd)
        if user_info:
            global current_user
            current_user = user
            # We keep user_info['height'] but not using it anymore
            messagebox.showinfo("Login Success", f"Welcome,
{current_user}!")
            show_page(main_menu_page)
        else:
            messagebox.showerror("Login Failed", "Invalid username or
password")

    def do_register():
        """Register a new user, asking for shoe size instead of height."""
        user = username_entry.get()
        pwd = password_entry.get()
        if user and pwd:
            # Prompt for shoe size
            shoe_size = simpledialog.askfloat("Shoe Size", "Enter your
shoe size (e.g., 10.5):")
            if shoe_size is not None:
                # Create the user folder structure
                user_base_dir = os.path.join("user_data", user)
                user_video_dir = os.path.join(user_base_dir, "videos")
                user_pictures_dir = os.path.join(user_base_dir,
"pictures")
                for dir_path in [user_video_dir, user_pictures_dir]:
                    if not os.path.exists(dir_path):
                        os.makedirs(dir_path)

                # Save user credentials (with shoe size) to shelve
```

```
                save_user(user, pwd, shoe_size)
                messagebox.showinfo("Register Success", f"User '{user}'
registered successfully with shoe size {shoe_size}.")
            else:
                messagebox.showerror("Register Failed", "Shoe size entry
is required.")
        else:
            messagebox.showerror("Register Failed", "Please enter a
username and password")

    btn_frame = tk.Frame(frame)
    btn_frame.pack(pady=10)
    tk.Button(btn_frame, text="Login", command=do_login).grid(row=0,
column=0, padx=5)
    tk.Button(btn_frame, text="Register", command=do_register).grid(row=0,
column=1, padx=5)
    tk.Button(frame, text="Exit App", command=exit_app).pack(pady=5)

def main_menu_page():
    frame = tk.Frame(container)
    frame.pack(expand=True, fill="both")
    tk.Label(frame, text=f"Welcome {current_user}", font=("Helvetica",
20)).pack(pady=20)
    btn_frame = tk.Frame(frame)
    btn_frame.pack(pady=20)

    tk.Button(btn_frame, text="Start Video", width=20, command=lambda:
show_page(video_page)).grid(row=0, column=0, padx=10, pady=10)
    tk.Button(btn_frame, text="Record Video Clip", width=20,
command=lambda: toggle_recording(current_user)).grid(row=0, column=1,
padx=10, pady=10)
    tk.Button(btn_frame, text="Stop Video", width=20,
command=stop_video).grid(row=0, column=2, padx=10, pady=10)
    tk.Button(btn_frame, text="View Recordings", width=20, command=lambda:
show_page(recordings_page)).grid(row=1, column=0, padx=10, pady=10)
    tk.Button(btn_frame, text="View Pictures", width=20, command=lambda:
show_page(pictures_page)).grid(row=1, column=1, padx=10, pady=10)
```

```
    tk.Button(btn_frame, text="Bowling Analysis", width=20,
command=lambda: show_page(bowling_analysis_page)).grid(row=1, column=2,
padx=10, pady=10)

    tk.Button(btn_frame, text="Exit App", width=20,
command=exit_app).grid(row=2, column=1, padx=10, pady=10)

def video_page():
    frame = tk.Frame(container)
    frame.pack(expand=True, fill="both")
    global video_label
    video_label = tk.Label(frame)
    video_label.pack(pady=10)

    # Add a control frame for buttons under the video display
    control_frame = tk.Frame(frame)
    control_frame.pack(pady=5)

    # Record button integrated into the video window
    record_button = tk.Button(control_frame, text="Record Video Clip",
                              command=lambda:
toggle_recording(current_user))
    record_button.grid(row=0, column=0, padx=5)

    tk.Button(control_frame, text="Stop Video",
command=stop_video).grid(row=0, column=1, padx=5)
    tk.Button(control_frame, text="Back", command=lambda:
show_page(main_menu_page)).grid(row=0, column=2, padx=5)

    start_video()


def recordings_page():
    frame = tk.Frame(container)
    frame.pack(expand=True, fill="both")
    tk.Label(frame, text="Recorded Videos", font=("Helvetica",
16)).pack(pady=10)
    user_video_dir = os.path.join("user_data", current_user, "videos")
    if not os.path.exists(user_video_dir):
        tk.Label(frame, text="No recordings found.").pack(pady=5)
```

```
    else:
        listbox = tk.Listbox(frame)
        listbox.pack(fill="both", expand=True, padx=20, pady=10)
        videos = os.listdir(user_video_dir)
        for video in videos:
            listbox.insert(tk.END, video)

        def play_video():
            selected = listbox.curselection()
            if selected:
                video_file = listbox.get(selected[0])
                video_path = os.path.join(user_video_dir, video_file)
                show_page(video_playback_page, video_path)

        tk.Button(frame, text="Play Video",
command=play_video).pack(pady=5)
    tk.Button(frame, text="Back", command=lambda:
show_page(main_menu_page)).pack(pady=10)

def pictures_page():
    frame = tk.Frame(container)
    frame.pack(expand=True, fill="both")
    tk.Label(frame, text="Stored Pictures", font=("Helvetica",
16)).pack(pady=10)

    user_pictures_dir = os.path.join("user_data", current_user,
"pictures")
    if not os.path.exists(user_pictures_dir):
        tk.Label(frame, text="No pictures found.").pack(pady=5)
    else:
        left_frame = tk.Frame(frame)
        left_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10,
pady=10)
        right_frame = tk.Frame(frame)
        right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True,
padx=10, pady=10)

        listbox = tk.Listbox(left_frame)
        listbox.pack(fill=tk.BOTH, expand=True)
```

```
        pictures = os.listdir(user_pictures_dir)
        for picture in pictures:
            listbox.insert(tk.END, picture)

        image_label = tk.Label(right_frame)
        image_label.pack(fill=tk.BOTH, expand=True)

        def display_picture(event):
            selected = listbox.curselection()
            if selected:
                picture_file = listbox.get(selected[0])
                picture_path = os.path.join(user_pictures_dir,
picture_file)
                img = Image.open(picture_path)
                imgtk = ImageTk.PhotoImage(image=img)
                image_label.config(image=imgtk)
                image_label.image = imgtk

        listbox.bind('<<ListboxSelect>>', display_picture)

        def annotate_picture(listbox):
            selected = listbox.curselection()
            if selected:
                picture_file = listbox.get(selected[0])
                user_base_dir = os.path.join("user_data", current_user)
                user_pictures_dir = os.path.join(user_base_dir,
"pictures")
                user_annotated_pictures_dir = os.path.join(user_base_dir,
"annotated_pictures")
                if not os.path.exists(user_annotated_pictures_dir):
                    os.makedirs(user_annotated_pictures_dir)

                picture_path = os.path.join(user_pictures_dir,
picture_file)
                timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                original_filename = os.path.splitext(picture_file)[0]
                file_extension = os.path.splitext(picture_file)[1]
                annotated_filename =
f"{original_filename}_annotated{file_extension}"
```

```
                save_path = os.path.join(user_annotated_pictures_dir,
annotated_filename)

                # 1) Retrieve the user's shoe size from stored credentials
                user_info = get_user_info(current_user)
                shoe_size = user_info["shoe_size"]  # if you stored it
under that key

                # 2) Pass shoe_size to process_frame
                process_frame(save_path, picture_path, current_user,
shoe_size=shoe_size)

                messagebox.showinfo("Success", f"Frame saved as
{annotated_filename}")


        def annotate_all_pictures():
            user_base_dir = os.path.join("user_data", current_user)
            user_annotated_pictures_dir = os.path.join(user_base_dir,
"annotated_pictures")
            if not os.path.exists(user_annotated_pictures_dir):
                os.makedirs(user_annotated_pictures_dir)

            if not pictures:
                messagebox.showinfo("No Images", "No pictures found to
annotate.")
                return

            # Retrieve shoe size here, too
            user_info = get_user_info(current_user)
            shoe_size = user_info["shoe_size"]

            for picture_file in pictures:
                picture_path = os.path.join(user_pictures_dir,
picture_file)

                timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                original_filename = os.path.splitext(picture_file)[0]
                file_extension = os.path.splitext(picture_file)[1]
```

```
                annotated_filename =
f"{original_filename}_annotated{file_extension}"
                save_path = os.path.join(user_annotated_pictures_dir,
annotated_filename)

                # Use the stored shoe size
                process_frame(save_path, picture_path, current_user,
shoe_size=shoe_size)

            messagebox.showinfo("Success", "All pictures have been
annotated and saved!")


        # NEW: "Study Picture" button
        def study_picture(listbox):
            selected = listbox.curselection()
            if selected:
                picture_file = listbox.get(selected[0])
                # We'll define a function below for analyzing the selected
picture
                analyze_single_picture(picture_file)

        tk.Button(frame, text="Annotate and Save", command=lambda:
annotate_picture(listbox)).pack(pady=5)
        tk.Button(frame, text="Annotate ALL Pictures",
command=annotate_all_pictures).pack(pady=5)
        tk.Button(frame, text="Study Picture", command=lambda:
study_picture(listbox)).pack(pady=5)

    tk.Button(frame, text="Back", command=lambda:
show_page(main_menu_page)).pack(pady=10)


def analyze_single_picture(picture_file):
    """
    Loads the row in {current_user}_biomechanical_measurements.csv for the
given picture_file,
    then:
      - Loads the trained classification model (random forest .pkl).
```

```
      - Builds a single-row feature vector from the CSV data (skipping
Timestamp and Image).
      - Predicts whether this delivery is Good (1) or Bad (0), plus the
probability.
      - Also compares to feature importances (CSV) and provides custom
feedback.
    """
    user_base_dir = os.path.join("user_data", current_user)
    csv_path = os.path.join(user_base_dir,
f"{current_user}_biomechanical_measurements.csv")
    fi_path = os.path.join(user_base_dir,
f"{current_user}_feature_importances.csv")
    model_path = os.path.join(user_base_dir,
"trained_rf_classifier_85good.pkl")  # or your correct model file

    if not os.path.exists(csv_path):
        messagebox.showerror("Error", f"No measurements CSV found for
{current_user}.")
        return

    # Check if the trained model file exists
    if not os.path.exists(model_path):
        messagebox.showerror("Error", f"No trained classifier found.
Please run a Bowling Analysis first.")
        return

    try:
        import joblib
        from sklearn.exceptions import NotFittedError

        # 1) Load the classification model
        rf_clf = joblib.load(model_path)

        # 2) Read the CSV and find the row for the given picture
        #    (If you have lines like '########' above the header, use
skiprows or remove them.)
        data = pd.read_csv(csv_path, encoding='cp1252',
on_bad_lines='skip', sep=',')

        # Match the row where the "Image" column ends with picture_file
```

```
        row = data.loc[data["Image"].str.endswith(picture_file)]
        if row.empty:
            messagebox.showerror("Not Found", f"No data found for
{picture_file}")
            return

        # 3) Make a copy for processing
        single_row = row.copy()

        # 3a) Rename columns if your training script used different names
        rename_cols = {
            "Right Elbow Angle": "Elbow Angle",
            "Knee Flexion Angle": "Left Knee Angle",
            "Arm Rotation Speed (deg/s)": "arm rotation speed (deg/sec)"
        }
        single_row.rename(columns=rename_cols, inplace=True)

        # 3b) Drop or ignore the first two columns (Timestamp, Image) so
they won't break numeric processing
        #     (We already used "Image" for matching above, so now we don't
need it in X_single.)
        #     If your first two columns are literally named "Timestamp"
and "Image", you can do:
        non_numeric_cols = ["Timestamp", "Image"]
        single_row.drop(columns=non_numeric_cols, axis=1, inplace=True,
errors="ignore")

        # 4) Define exactly which columns your model needs
        feature_cols = [
            "Elbow Angle",
            "Shoulder Angle",
            "Left Knee Angle",
            "Stride Length (cm)",
            "Right arm length (cm)",
            "arm rotation speed (deg/sec)"
        ]

        # Make sure those columns exist
        for c in feature_cols:
            if c not in single_row.columns:
```

```
                messagebox.showerror("Error", f"Missing column '{c}' in
the CSV for {picture_file}")
                return

        # 5) Build the numeric feature vector
        X_single = single_row[feature_cols]

        # Fill numeric NaNs with the mean of each column
        numeric_cols = X_single.select_dtypes(include=[np.number]).columns
        X_single[numeric_cols] =
X_single[numeric_cols].fillna(X_single[numeric_cols].mean())

        # 6) Predict Good/Bad
        y_pred_single = rf_clf.predict(X_single)
        y_prob_single = rf_clf.predict_proba(X_single)[:, 1]
        pred_label = y_pred_single[0]
        pred_prob_good = y_prob_single[0]

        if pred_label == 1:
            classification_feedback = (
                f"Model Classification: GOOD bowl.\n"
                f"Confidence (prob good): {pred_prob_good:.2f}\n"
            )
        else:
            classification_feedback = (
                f"Model Classification: BAD bowl.\n"
                f"Confidence (prob good): {pred_prob_good:.2f}\n"
            )

        # 7) Load feature importances for context
        if not os.path.exists(fi_path):
            messagebox.showinfo("No Analysis Yet", "No feature importances
found. Please run a Bowling Analysis first.")
            return
        importances_df = pd.read_csv(fi_path)
        fi_dict = dict(zip(importances_df["Feature"],
importances_df["Importance"]))
        sorted_fi = sorted(fi_dict.items(), key=lambda x: x[1],
reverse=True)
```

```
        # 8) Construct feedback message
        feedback = "Analysis for single picture:\n"
        feedback += classification_feedback
        feedback += "\nMost important features (from last analysis):\n"
        for feat, imp in sorted_fi:
            feedback += f"  {feat}: {imp:.3f}\n"

        # 8a) Angle-based suggestions
        elbow_angle = X_single["Elbow Angle"].values[0]
        shoulder_angle = X_single["Shoulder Angle"].values[0]
        knee_angle = X_single["Left Knee Angle"].values[0]
        stride_length = X_single["Stride Length (cm)"].values[0]
        arm_speed = X_single["arm rotation speed (deg/sec)"].values[0]

        feedback += "\nMeasured angles/distances:\n"
        feedback += f"  Elbow Angle: {elbow_angle:.2f}\n"
        feedback += f"  Shoulder Angle: {shoulder_angle:.2f}\n"
        feedback += f"  Left Knee Angle: {knee_angle:.2f}\n"
        feedback += f"  Stride Length (cm): {stride_length:.2f}\n"
        feedback += f"  Arm Rotation Speed (deg/sec): {arm_speed:.2f}\n\n"

        suggestions = []
        if elbow_angle > 160:
            suggestions.append("Try to reduce elbow angle; keep it more
flexible for pace.")
        elif elbow_angle < 100:
            suggestions.append("Your elbow angle might be too small,
limiting your release swing.")

        if arm_speed < 1000:
            suggestions.append("Try to rotate your arm faster for more
pace.")
        elif arm_speed > 2850:
            suggestions.append("Great arm rotation speed! Keep it
consistent.")

        if stride_length > 150:
            suggestions.append("Your stride length seems quite large; aim
for more controlled stride.")
        elif stride_length < 100:
```

```
            suggestions.append("A longer stride can help generate
momentum.")

        if suggestions:
            feedback += "Feedback / Suggestions:\n" + "\n".join(f" - {s}"
for s in suggestions)
        else:
            feedback += "Your form on this image looks fairly balanced.\n"

        # 9) Show in a popup
        messagebox.showinfo("Picture Analysis", feedback)

    except NotFittedError:
        messagebox.showerror("Model Error", "The loaded classifier is not
fitted. Please re-train.")
    except Exception as e:
        messagebox.showerror("Error", f"Could not analyze
{picture_file}.\nError: {e}")



def video_playback_page(video_path):
    frame = tk.Frame(container)
    frame.pack(expand=True, fill="both")
    tk.Label(frame, text="Video Playback", font=("Helvetica",
16)).pack(pady=10)
    lbl_video = tk.Label(frame)
    lbl_video.pack(pady=10)
    cap_playback = cv2.VideoCapture(video_path)
    if not cap_playback.isOpened():
        tk.Label(frame, text="Error opening video.").pack(pady=5)
        tk.Button(frame, text="Back", command=lambda:
show_page(recordings_page)).pack(pady=10)
        return
    total_frames = int(cap_playback.get(cv2.CAP_PROP_FRAME_COUNT))
    slider = tk.Scale(frame, from_=0, to=total_frames,
orient=tk.HORIZONTAL, length=400)
    slider.pack(pady=10)

    def update_frame(value):
```

```
        frame_no = int(value)
        cap_playback.set(cv2.CAP_PROP_POS_FRAMES, frame_no)
        ret, frame_img = cap_playback.read()
        if ret:
            frame_img = cv2.cvtColor(frame_img, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(frame_img)
            imgtk = ImageTk.PhotoImage(image=img)
            lbl_video.config(image=imgtk)
            lbl_video.image = imgtk

    slider.config(command=update_frame)
    def take_picture_from_video(cap_obj):
        ret, frame_img = cap_obj.read()
        if ret:
            timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
            user_base_dir = os.path.join("user_data", current_user)
            user_pictures_dir = os.path.join(user_base_dir, "pictures")
            if not os.path.exists(user_pictures_dir):
                os.makedirs(user_pictures_dir)
            filename = os.path.join(user_pictures_dir,
f"frame_{timestamp}.png")
            cv2.imwrite(filename, frame_img)
            messagebox.showinfo("Saved", f"Frame saved as {filename}")

    tk.Button(frame, text="Take Picture", command=lambda:
take_picture_from_video(cap_playback)).pack(pady=5)
    tk.Button(frame, text="Back", command=lambda:
show_page(recordings_page)).pack(pady=10)

def exit_app():
    stop_video()
    root.quit()
    root.destroy()

def bowling_analysis_page():
    """
    ML analysis page that:
      1) Shows a preview of the user's data.
      2) Trains a Random Forest to predict Speed (km/h).
```

```
      3) Prints MAE, cross-validation results, feature importances, and
feedback.
      4) Overwrites the user's feature importances CSV (instead of
appending).
      5) Displays the text results and the feature-importance chart side
by side.
    """
    frame = tk.Frame(container)
    frame.pack(expand=True, fill="both")

    tk.Label(frame, text="Bowling Analysis with Machine Learning",
font=("Helvetica", 16)).pack(pady=10)

    # 1) Create a container for left (text) and right (chart)
    content_frame = tk.Frame(frame)
    content_frame.pack(expand=True, fill="both")

    # Left frame for text
    left_frame = tk.Frame(content_frame)
    left_frame.pack(side=tk.LEFT, expand=True, fill="both", padx=10,
pady=10)

    # Right frame for chart
    right_frame = tk.Frame(content_frame)
    right_frame.pack(side=tk.RIGHT, expand=True, fill="both", padx=10,
pady=10)

    # Text widget goes to the left
    results_text = tk.Text(left_frame, height=20, width=60)
    results_text.pack(expand=True, fill="both")

    # Chart label on the right
    chart_label = tk.Label(right_frame)
    chart_label.pack(expand=True, fill="both")

    def run_analysis():
        import pandas as pd
        import numpy as np
        from sklearn.ensemble import RandomForestRegressor
```

```
        from sklearn.model_selection import train_test_split,
cross_val_score
        from sklearn.metrics import mean_absolute_error
        from io import BytesIO
        from PIL import Image, ImageTk
        import matplotlib
        matplotlib.use("Agg")  # Non-GUI backend
        import matplotlib.pyplot as plt
        from datetime import datetime
        import os, traceback

        try:
            # ------------------------------------------------
            # 1) Build path to the user's CSV
            # ------------------------------------------------
            user_base_dir = os.path.join("user_data", current_user)
            csv_path = os.path.join(user_base_dir,
f"{current_user}_biomechanical_measurements.csv")

            # ------------------------------------------------
            # 2) Load CSV and show preview
            # ------------------------------------------------
            data = pd.read_csv(csv_path, encoding='cp1252')
            preview_str = "Data Preview:\n" +
data.head().to_string(index=False) + "\n\n"

            # ------------------------------------------------
            # 3) Define target & features
            # ------------------------------------------------
            y = data["Speed (km/h)"]
            X = data[[
                "Elbow Angle",
                "Shoulder Angle",
                "Left Knee Angle",
                "Stride Length (cm)",
                "Arm rotation speed"
            ]]

            # Handle missing
            X = X.fillna(X.mean())
```

```
            y = y.fillna(y.mean())

            # Check data size
            if len(data) < 2:
                raise ValueError("Not enough data to train/test split.
Please record or annotate more measurements.")

            # Train/test split
            X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

            # ------------------------------------------------
            # 4) Build & train Random Forest
            # ------------------------------------------------
            rf = RandomForestRegressor(n_estimators=100, random_state=42)
            rf.fit(X_train, y_train)

            # ------------------------------------------------
            # 5) Predict, Evaluate
            # ------------------------------------------------
            y_pred = rf.predict(X_test)
            mae = mean_absolute_error(y_test, y_pred)
            cv_scores = cross_val_score(rf, X, y, cv=5,
scoring='neg_mean_absolute_error')
            cv_mae = -np.mean(cv_scores)

            # ------------------------------------------------
            # 6) Feature Importances
            # ------------------------------------------------
            importances = rf.feature_importances_
            feature_names = X.columns

            # 6a) Overwrite these weights for each user run (instead of
appending)
            importances_df = pd.DataFrame({
                "Timestamp": [datetime.now()] * len(feature_names),
                "Feature": feature_names,
                "Importance": importances
            })
```

```
            weightings_path = os.path.join(user_base_dir,
f"{current_user}_feature_importances.csv")
            # Use mode='w' so we overwrite the file each time
            importances_df.to_csv(weightings_path, mode='w', index=False,
header=True)

            # 6b) Provide feedback
            sorted_idx = np.argsort(importances)[::-1]
            sorted_features = feature_names[sorted_idx]
            sorted_importances = importances[sorted_idx]

            feedback_str = "\nFeedback:\n"
            feedback_str += "These features have the greatest impact on
ball speed:\n"

            top_n = 2
            top_feats = sorted_features[:top_n]
            top_imps = sorted_importances[:top_n]

            suggestions_dict = {
                "Elbow Angle": "Keep elbow angle consistent to maximize
control.",
                "Shoulder Angle": "Work on shoulder rotation and
alignment.",
                "Left Knee Angle": "Front knee stability helps in power
transfer.",
                "Stride Length (cm)": "Proper stride builds momentum.",
                "Arm rotation speed": "Increasing arm rotation speed adds
extra pace."
            }

            for f, imp_val in zip(top_feats, top_imps):
                feedback_str += f"  * {f} ({imp_val:.2f} importance)\n"
                if f in suggestions_dict:
                    feedback_str += f"    Suggestion:
{suggestions_dict[f]}\n"

            # ------------------------------------------------
            # 7) Build textual results to display
            # ------------------------------------------------
```

```
            results_str = preview_str
            results_str += f"Mean Absolute Error on Test Set: {mae:.2f}
km/h\n"
            results_str += f"Cross-Validated MAE (5-fold): {cv_mae:.2f}
km/h\n\n"
            results_str += "Feature Importances (Weights):\n"
            for name, imp in zip(feature_names, importances):
                results_str += f"{name}: {imp:.4f}\n"
            results_str += feedback_str

            # -------------------------------------------------
            # 8) Show textual results in the text widget
            # -------------------------------------------------
            results_text.delete("1.0", tk.END)
            results_text.insert(tk.END, results_str)

            # -------------------------------------------------
            # 9) Plot a bar chart of feature importances
            # -------------------------------------------------
            fig, ax = plt.subplots()
            ax.barh(feature_names, importances)
            ax.set_xlabel("Importance")
            ax.set_title("Feature Importances")

            buf = BytesIO()
            fig.savefig(buf, format='png')
            buf.seek(0)
            im = Image.open(buf)
            imgtk = ImageTk.PhotoImage(im)
            chart_label.config(image=imgtk)
            chart_label.image = imgtk
            plt.close(fig)

        except Exception as e:
            error_message = "".join(traceback.format_exception(type(e), e,
e.__traceback__))
            results_text.delete("1.0", tk.END)
            results_text.insert(tk.END, f"Error during
analysis:\n{error_message}")
            chart_label.config(image='')
```

```
    # Buttons at the bottom
    tk.Button(frame, text="Run Analysis",
command=run_analysis).pack(pady=5)
    tk.Button(frame, text="Back to Main Menu", command=lambda:
show_page(main_menu_page)).pack(pady=5)



# ----------------------------
# Start the Application
# ----------------------------
show_page(login_page)
root.mainloop()
```