ENHANCED ATHLETE TRACKING AND VISUALIZATION

A PROJECT REPORT

Submitted by,

MANOJ C ACHARYA - 20211ISR0070

Under the guidance of,

Dr. RUHIN KOUSER

Assistant Professor School of Computer Science & Engineering

Presidency University

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION SCIENCE AND ENGINEERING (AI AND ROBOTICS).

At

PRESIDENCY UNIVERSITY

BENGALURU

JANUARY 2025

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the Project report **"ENHANCED ATHLETE TRACKING AND VISUALIZATION"** being submitted by "MANOJ C ACHARYA" bearing roll number(s) "20211ISR0070" in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a Bonafide work carried out under my supervision.

Dr . RUHIN KOUSER
Assistant Professor & Guide
School of CSE&IS
Presidency University

Dr . ZAFAR ALI KHAN
Professor & HoD
School of CSE&IS
Presidency University

Dr. L. SHAKKEERA
Associate Dean
School of CSE
Presidency University

Dr. MYDHILI NAIR
Associate Dean
School of CSE
Presidency University

Dr. SAMEERUDDIN KHAN
Pro-Vc School of Engineering
Dean -School of CSE&IS
Presidency University

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

DECLARATION

We hereby declare that the work, which is being presented in the project report entitled **ENHANCED ATHLETE TRACKING AND VISUALIZATION** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Science and Engineering**, is a record of our own investigations carried under the guidance of **Dr. Ruhin Kouser, Assistant Professor, School of Computer Science Engineering & Information Science, Presidency University, Bengaluru.**

We have not submitted the matter presented in this report anywhere for the award of any other Degree.

Name		Roll No	Signatures
MANOJ C ACHARYA	-	20211ISR0070	

ABSTRACT

The integration of advanced computer vision techniques has transformed the way athletic performance is monitored, analysed, and visualized. This project focuses on developing a real-time runner tracking and performance visualization system, leveraging the YOLOv11 object detection model to identify and track runners on a track. By filtering out non-relevant objects and assigning unique IDs to detected humans, the system ensures consistent tracking of individuals within predefined lane regions.

Using perspective transformation, the system dynamically overlays critical information such as each runner's name, department, speed, and distance to the finish line directly onto the track in a 3D perspective. This feature enhances the viewing experience by providing an intuitive and immersive way to monitor performance. Additionally, the system automates race result determination by identifying the sequence in which runners cross the finish line and updating rankings in real time.

This project demonstrates the potential of integrating computer vision and real-time data visualization for improving the accuracy, efficiency, and engagement of athletic events. It offers a scalable and robust solution for race monitoring, athlete performance analysis, and audience interaction, with applications extending to sports analytics, event management, and broadcasting.

KEYWORDS: Advanced computer vision, YOLOv11, real-time tracking, runner tracking, performance visualization, perspective transformation, speed and distance overlay, 3D visualization, race result automation, real-time data visualization, sports analytics, event management, broadcasting, athlete performance analysis, race monitoring, predefined lane regions, audience interaction.

ACKNOWLEDGEMENT

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC, School of Engineering and Dean, School of Computer Science Engineering & Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Deans **Dr. Shakkeera L and Dr. Mydhili Nair,** School of Computer Science Engineering & Information Science, Presidency University, and Dr. "ZAFER ALI KHAN", Head of the Department, School of Computer Science Engineering & Information Science, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Dr. Ruhin Kouser,** Assistant Professor, School of Computer Science Engineering & Information Science, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the project work.

We would like to convey our gratitude and heartfelt thanks to the PIP2001 Capstone Project Coordinators **Dr. Sampath A K, Dr. Abdul Khadar A and Mr. Md Zia Ur Rahman,** department Project Coordinators **Dr. Afroz Pasha** and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

MANOJ C ACHARYA

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

The Role of Technology in Modern Sports

Technology has become an integral part of sports, transforming how athletes train, compete, and engage with fans. From wearable fitness trackers and performance analysis tools to advanced broadcasting techniques, technological innovations have revolutionized the sporting world. This intersection of sports and technology has not only enhanced athlete performance but also elevated the spectator experience to new heights, creating an immersive and data-rich environment for all stakeholders.

In recent years, the integration of artificial intelligence (AI) and computer vision has enabled unprecedented advancements in monitoring and analysing athletic performance. Real-time tracking systems, such as those powered by object detection models, have allowed coaches, analysts, and fans to gain deeper insights into the dynamics of sports events. These systems go beyond traditional methods by providing accurate, real-time data on speed, distance, and positional tracking, making them indispensable tools for modern sports management and entertainment.

The Emergence of Entertainment as a Driving Force in Sports

Entertainment has emerged as a significant driver in the evolution of sports. With the increasing commercialization of sports events, audiences expect more than just a game—they demand an engaging, interactive, and visually appealing experience. High-definition broadcasting, augmented reality (AR), and immersive analytics have transformed sports into a spectacle that captivates millions globally. This shift has created new opportunities for integrating technology into sports, not just to enhance athlete performance but also to redefine how fans experience the action.

The growing influence of entertainment has also spurred innovations aimed at enhancing the storytelling aspect of sports. Real-time data visualization, interactive scoreboards, and on-field augmented displays allow fans to connect with the game on a deeper level. These advancements are not limited to large-scale events; they are also being adopted in smaller-scale competitions and amateur sports, democratizing access to high-tech tools and enriching

the overall sporting landscape.

The Importance of Real-Time Tracking in Athletics

Athletics, one of the oldest and most widely practiced sports, has seen significant advancements due to the integration of technology. Real-time tracking and analysis have become critical components in events such as track and field races, where precise measurements of speed, distance, and position can influence strategies and outcomes. Traditionally, athletic performance has been monitored using manual methods, which are often prone to errors and inefficiencies. The advent of automated systems has eliminated these challenges, enabling precise and consistent data collection.

In track events, the ability to track runners in real time provides invaluable insights into their performance. Metrics such as speed, acceleration, and split times can be analysed to identify strengths and areas for improvement. Additionally, automated systems facilitate accurate race result determination by identifying the exact order in which athletes cross the finish line. This level of precision is particularly crucial in close finishes, where manual methods may fall short.

Overview of the Project

This project aims to develop a real-time runner tracking and performance visualization system using advanced computer vision techniques. The system leverages the YOLOv11 object detection model to identify and track runners on a track, filtering out irrelevant objects such as cones, chairs, and tents. By assigning unique IDs to each detected runner, the system ensures consistent tracking across frames. To refine the detection process, track ranges are defined as pixel values corresponding to specific lanes, allowing the system to accurately associate each runner with their respective lane.

The system calculates essential performance metrics, including speed and distance to the finish line, and displays this information dynamically on the track using perspective transformation. This approach ensures that the displayed data aligns with the track's 3D perspective, creating an intuitive and visually appealing visualization. As runners cross the finish line, their names, departments, and rankings are displayed on their respective lanes in real time. This feature automates the race result determination process, enhancing efficiency and accuracy.

The Importance of Perspective Transformation

One of the project's key innovations is the use of perspective transformation to overlay data on the track in a visually accurate manner. Traditional 2D overlays often appear flat and disconnected from the scene, reducing their effectiveness in conveying information. Perspective transformation aligns the overlay with the track's 3D geometry, creating a more immersive and intuitive visualization. This technique not only enhances the viewer experience but also ensures that the displayed data is easily interpretable.

Perspective transformation has broad applications beyond athletics, including in sports broadcasting, gaming, and augmented reality. By demonstrating its utility in this project, the system highlights the potential of this technique in creating engaging and informative visualizations across various domains.

Leveraging YOLOv11 for Real-Time Detection

The YOLO (You Only Look Once) family of object detection models has been a game-changer in computer vision, offering a balance between speed and accuracy. YOLOv11, the latest iteration, builds on its predecessors' strengths with improved detection capabilities, making it well-suited for real-time applications. In this project, YOLOv11 is configured to detect humans (class ID 0) and assign unique IDs to each detected individual. This ensures consistent tracking of runners within the camera's field of view.

By filtering out non-relevant objects, the system focuses exclusively on runners, improving its efficiency and accuracy. YOLOv11's ability to perform real-time detection is crucial for this project, as it ensures that data is updated dynamically, enabling seamless visualization and analysis.

Automating Race Results

Automating race result determination is a significant aspect of this project. Traditionally, race results are determined manually or through photo-finish systems, which can be time-consuming and prone to disputes. The proposed system eliminates these challenges by identifying the sequence in which runners cross the finish line. This is achieved by continuously tracking each runner's position and dynamically updating their rankings as they approach the finish line.

The automated system not only improves accuracy but also enhances the overall efficiency of race management. By displaying real-time rankings and performance data, the system provides an engaging experience for both participants and spectators.

Applications and Benefits

The developed system has wide-ranging applications and benefits, extending beyond athletics to other sports and domains. Some of the key applications include:

1. **Sports Analytics:** The system provides coaches and analysts with detailed performance data, enabling them to identify strengths, weaknesses, and areas for improvement.

2. **Event Management:** By automating race results and providing real-time updates, the system streamlines event management processes, reducing the workload on officials and minimizing errors.

3. **Broadcasting:** The integration of perspective-transformed overlays enhances sports broadcasting, creating an immersive experience for viewers.

4. **Athlete Training:** The system's ability to provide real-time feedback makes it a valuable tool for athlete training programs, helping athletes monitor their performance and adjust their strategies.

5. **Audience Engagement:** By offering dynamic visualizations and real-time updates, the system enhances the spectator experience, making events more interactive and engaging.

The Broader Impact of Technology in Sports

The integration of technology in sports extends beyond individual projects, shaping the future of the industry. Innovations such as real-time tracking, augmented reality, and AI-driven analytics are driving a shift toward data-driven decision-making. These advancements are not limited to professional sports; they are also being adopted in grassroots and amateur levels, democratizing access to cutting-edge tools.

Moreover, technology has the potential to play sports more inclusive and accessible. For example, real-time tracking systems can be adapted to assist referees in making fair

decisions or to provide accessibility features for individuals with disabilities. The possibilities are endless, and projects like this one highlight the transformative power of technology in sports.

This project represents a significant step forward in integrating computer vision and real-time data visualization into athletics. By leveraging advanced techniques such as YOLOv11 and perspective transformation, the system offers a robust solution for tracking runners, calculating performance metrics, and automating race results. Its applications extend beyond athletics, showcasing the potential of technology to enhance sports analytics, broadcasting, and audience engagement.

As the demand for immersive and data-driven experiences continues to grow, projects like this one will play a crucial role in shaping the future of sports. By combining technical innovation with practical applications, this project demonstrates how technology can enhance the efficiency, accuracy, and enjoyment of athletic events, setting the stage for further advancements in the field.

CHAPTER 2
LITERATURE SURVEY

Introduction to Computer Vision in Sports

The application of computer vision in sports has garnered significant attention in recent years, driven by advancements in deep learning, hardware capabilities, and algorithmic efficiency. Computer vision techniques have transformed traditional methods of performance analysis, enabling real-time tracking, motion analysis, and automated decision-making. This literature survey explores various studies and technologies that form the foundation of the proposed runner tracking and performance visualization system, focusing on object detection, tracking algorithms, and perspective transformation.

Object Detection Models in Sports

Object detection is a cornerstone of modern computer vision applications. In sports, detecting and identifying objects such as players, balls, and equipment is essential for performance analysis and visualization.

1. **YOLO Family of Models** The "You Only Look Once" (YOLO) series of models have revolutionized object detection with their real-time capabilities and high accuracy. Redmon et al. (2016) introduced YOLOv1, which laid the groundwork for single-stage detection pipelines by treating object detection as a regression problem. Subsequent iterations, including YOLOv3 (Redmon & Farhadi, 2018) and YOLOv4 (Bochkovskiy et al., 2020), improved detection accuracy and speed by incorporating techniques such as residual blocks, multi-scale predictions, and data augmentation strategies.

   The latest versions, such as YOLOv5 and YOLOv7, emphasize modularity and user-friendly interfaces. YOLOv11, as employed in this project, further refines these capabilities with enhanced feature extraction and context-awareness, making it ideal for tracking runners on a track.

2. **Single-Shot Detectors (SSD)** Liu et al. (2016) developed the SSD framework, another popular single-stage object detection method. SSD focuses on achieving a balance between speed and accuracy by employing a series of convolutional layers

for multi-scale feature detection. While SSD has been widely adopted in various domains, its performance in highly dynamic environments like sports is often outperformed by YOLO due to YOLO's optimized architecture for real-time detection.

3. **Region-Based Convolutional Neural Networks (R-CNNs)** R-CNNs (Girshick, 2014) and their successors, such as Fast R-CNN (Girshick, 2015) and Faster R-CNN (Ren et al., 2016), are two-stage object detection models known for their high accuracy. These models use region proposal networks (RPNs) to identify object locations before classification. Despite their precision, R-CNNs are computationally expensive, making them less suitable for real-time applications like the one proposed in this project.

Multi-Object Tracking in Sports

Tracking multiple objects in a dynamic environment is a challenging task that involves associating detected objects across consecutive frames. In sports, where objects like players or runners move rapidly and unpredictably, robust tracking algorithms are essential.

1. **SORT (Simple Online and Realtime Tracking)** SORT (Bewley et al., 2016) is a popular algorithm for real-time tracking. It uses the Kalman filter for motion prediction and the Hungarian algorithm for data association. SORT is known for its speed and simplicity, but it struggles in scenarios with frequent occlusions or crowded environments.

2. **DeepSORT** DeepSORT (Wojke et al., 2017) extends SORT by incorporating appearance features extracted using deep learning. By matching detected objects based on both motion and appearance, DeepSORT significantly improves tracking robustness in complex scenarios, such as tracking multiple runners in close proximity.

3. **ByteTrack** ByteTrack (Xiao et al., 2022) is a recent advancement in multi-object tracking that emphasizes high recall for unmatched detections. By tracking even low-confidence detections, ByteTrack minimizes missed associations, making it ideal for high-speed environments like athletics.

4. **TrackFormer** TrackFormer (Meinhardt et al., 2021) leverages transformer-based architectures to unify object detection and tracking into a single framework. This

approach has shown promising results in handling occlusions and maintaining track consistency, but its computational requirements make it less suitable for real-time applications.

Perspective Transformation and Visualization

Perspective transformation is a critical component of the proposed system, allowing for the dynamic overlay of data on the track in a visually accurate manner.

1. **Homography-Based Transformation** Homography is widely used in sports analytics for mapping 2D images to 3D planes. Hartley and Zisserman (2004) describe homography as a linear transformation that relates two perspectives of the same planar surface. By estimating the homography matrix using keypoint correspondences, it is possible to project information onto a real-world surface, such as a track.

2. **Applications in Sports Broadcasting** Perspective transformation has been extensively used in sports broadcasting to enhance viewer engagement. For instance, augmented overlays in soccer matches highlight player statistics, while trajectory visualization in cricket and tennis provides real-time ball tracking data (Pinto et al., 2019). These applications demonstrate the potential of perspective transformation in creating immersive experiences.

3. **3D Reconstruction** Techniques like Structure-from-Motion (SfM) and multi-view stereo (MVS) enable 3D reconstruction of sports environments. While these methods offer high accuracy, they are computationally intensive and unsuitable for real-time applications. However, advancements in lightweight 3D reconstruction techniques are paving the way for real-time integration in sports analytics.

Performance Metrics and Visualization

Visualizing performance metrics in real time is an essential aspect of enhancing both athlete analysis and audience engagement.

1. **Speed and Distance Measurement** Accurate measurement of speed and distance is a fundamental requirement for athletics. Markerless motion capture systems, such as those developed by OpenPose (Cao et al., 2019), have demonstrated the feasibility of

extracting motion data without physical markers. By integrating such techniques with real-time object detection, the proposed system calculates speed and distance dynamically.

2. **Real-Time Data Visualization** Effective data visualization is key to making performance metrics accessible and engaging. Dashboards, augmented overlays, and dynamic charts are commonly used in sports analytics. The proposed system employs perspective-transformed overlays to display runner-specific data directly on the track, enhancing interpretability and immersion.

3. **Automated Ranking Systems** Automated ranking systems are crucial for determining race results accurately and efficiently. Studies on race timing systems (e.g., RFID-based systems) highlight the limitations of traditional methods in real-time environments. By integrating tracking algorithms with dynamic ranking updates, the proposed system overcomes these challenges, ensuring precise result determination.

Advances in Race Monitoring Systems

The evolution of race monitoring systems reflects the growing demand for accuracy and automation in sports.

1. **Photo-Finish Systems** Photo-finish systems have long been the standard for determining race results. These systems use high-speed cameras positioned at the finish line to capture precise moments when athletes cross the line. However, their reliance on manual interpretation limits their scalability and efficiency.

2. **RFID and Wearable Sensors** RFID-based systems and wearable sensors offer an alternative to manual methods by providing automated timing and positioning data. While effective, these systems require specialized hardware, which may not always be feasible for large-scale or low-budget events.

3. **Vision-Based Systems** Vision-based systems have emerged as a promising solution for race monitoring, offering non-invasive and scalable alternatives to traditional methods. By combining object detection, tracking, and perspective transformation, these systems deliver accurate, real-time insights into race dynamics.

Summary and Research Gaps

The literature highlights significant advancements in object detection, tracking algorithms, and visualization techniques, all of which form the foundation of the proposed system. However, several gaps remain:

1. **Integration of Technologies:** Many existing systems focus on individual components, such as detection or tracking, without integrating these elements into a cohesive framework.

2. **Real-Time Efficiency:** High computational demands often limit the real-time applicability of advanced algorithms, particularly in dynamic environments like athletics.

3. **Immersive Visualizations:** While perspective transformation has been explored in broadcasting, its potential for real-time applications in athlete monitoring remains underutilized.

4. **Scalability:** Many systems are designed for specific sports or settings, limiting their adaptability to different scenarios.

This literature survey provides a comprehensive overview of the technologies and methodologies relevant to the proposed runner tracking and performance visualization system. By building on the strengths of existing approaches and addressing identified research gaps, this project aims to deliver a robust, real-time solution that enhances the efficiency, accuracy, and engagement of athletic events. The integration of YOLOv11 and perspective transformation ensures a scalable and innovative approach to revolutionizing sports analytics.

CHAPTER 3
RESEARCH GAPS OF EXISTING METHODS

Despite significant advancements in computer vision, real-time tracking, and performance analysis, several research gaps persist in the current methodologies. Addressing these gaps is critical to developing a robust and efficient system for tracking runners and visualizing their performance on the track. Below is a detailed exploration of the key research gaps identified in existing methods:

1. Integration of Detection, Tracking, and Visualization

Many existing systems focus on individual components, such as object detection, multi-object tracking, or data visualization, without fully integrating these elements into a cohesive framework. For instance:

- Object detection models like YOLO are widely used for identifying athletes, but these models are often not optimized for seamless integration with tracking algorithms such as DeepSORT or ByteTrack.
- Perspective transformation and real-time visualization techniques are often treated as separate processes rather than being integrated into a single pipeline.

Research Gap: A unified system that combines detection, tracking, and visualization in real time, while maintaining computational efficiency, is lacking.

2. Real-Time Performance Constraints

Real-time processing is crucial for applications in sports, where delays can compromise the accuracy and usefulness of the system. Existing methods often face challenges such as:

- High computational costs associated with advanced object detection and tracking models.
- Latency issues in visualizing performance metrics dynamically on the track.

Research Gap: Many current solutions struggle to meet the strict real-time performance requirements needed for live event monitoring and analysis.

3. Robust Tracking in Complex Scenarios

Multi-object tracking algorithms, such as SORT and DeepSORT, have shown promise in various applications, but they encounter limitations in scenarios involving:

- Occlusions, where runners overlap or block each other in the frame.
- Frequent entry and exit of objects (e.g., runners entering or leaving the camera's field of view).
- Variations in lighting, weather, and camera angles that can affect detection and tracking accuracy.

Research Gap: There is a need for tracking systems that can handle complex scenarios reliably, ensuring consistent runner identification and data accuracy.

4. Scalability Across Different Tracks and Events

Current systems are often designed for specific sports or settings, making them difficult to adapt to varying conditions. Challenges include:

- The need to redefine track ranges or calibration parameters for different track layouts.
- Limited flexibility to accommodate varying numbers of participants or changes in environmental conditions.

Research Gap: Developing a scalable and adaptable system that can generalize across different track configurations and event types is still an unresolved challenge.

5. Perspective Transformation Accuracy

Perspective transformation techniques, such as homography, are essential for overlaying performance metrics on the track. However:

- Existing methods rely on predefined calibration points, which can introduce errors if not accurately aligned.
- Variations in camera angles and lens distortions can affect the accuracy of the

transformation.

Research Gap: Improvements in automated calibration and adaptive perspective transformation methods are needed to ensure accurate visualization across different camera setups.

6. Inclusion of Contextual Information

Most systems focus solely on positional data (e.g., speed, distance) without incorporating contextual information, such as:

- Runner-specific details (e.g., name, department, ranking) in real time.
- Environmental factors (e.g., weather conditions, track surface) that may influence performance.

Research Gap: Integrating contextual information dynamically into the visualization process remains underexplored.

7. Automation of Result Determination

Race result determination is often dependent on manual verification or external timing systems (e.g., RFID or photo-finish cameras). While these methods are effective, they:

- Require additional hardware and setup, increasing costs.
- May not provide seamless integration with tracking and visualization systems.

Research Gap: Automating race result determination within a single vision-based system can significantly improve efficiency and accuracy.

8. Handling of Non-Track Areas

In many sports events, the camera's field of view includes areas outside the designated track, leading to:

- Detection of irrelevant objects (e.g., spectators, cones, or tents).
- Increased computational load and potential tracking inaccuracies.

Research Gap: Effective filtering mechanisms to exclude non-track areas dynamically are required to enhance system efficiency and focus.

9. Audience Engagement Through Enhanced Visualization

While data visualization techniques have advanced, their application in live sports scenarios often lacks:

- Immersive and intuitive overlays that align seamlessly with the track.
- Real-time updates that are visually engaging for both spectators and participants.

Research Gap: Enhancing audience engagement through innovative visualization methods, such as augmented reality (AR) or 3D overlays, remains a largely untapped area.

10. Accessibility and Cost Efficiency

Many existing systems rely on specialized hardware, such as high-speed cameras or wearable sensors, which can be prohibitively expensive for smaller-scale events. Additionally:

- The complexity of system setup and operation may limit its accessibility to non-technical users.

Research Gap: Developing a cost-effective, user-friendly solution that leverages widely available hardware (e.g., standard cameras) is essential for broader adoption.

11. Ethical and Privacy Considerations

The use of vision-based tracking systems raises ethical and privacy concerns, particularly in:

- Ensuring that data collection adheres to legal and ethical standards.
- Protecting the identities and personal information of individuals being monitored.

Research Gap: Establishing clear guidelines and implementing privacy-preserving measures in real-time tracking systems is a critical area that requires further exploration.

Conclusion

The identified research gaps highlight the limitations of existing methods and underscore the need for comprehensive solutions that integrate detection, tracking, and visualization seamlessly. By addressing these gaps, the proposed system aims to set a new benchmark in runner tracking and performance visualization, offering a scalable, accurate, and immersive experience for athletes, coaches, and spectators alike.

CHAPTER 4
PROPOSED MOTHODOLOGY

The proposed system aims to track runners on a track, gather their performance data, and dynamically visualize relevant information such as speed, distance, and ranking. This methodology leverages advanced computer vision techniques, real-time data processing, and perspective transformation to achieve the outlined objectives. The following steps describe the detailed methodology for implementing the project:

1. System Overview

The system consists of three main components:

- **Input Module:** Captures video feed from a camera positioned to cover the entire track.
- **Processing Module:** Detects and tracks runners, calculates performance metrics, and determines rankings.
- **Output Module:** Dynamically visualizes the runners' data on the track using perspective transformation.

The entire pipeline operates in real time, ensuring seamless integration and immediate feedback.

2. Input Module

- **Camera Setup:**
    - A high-definition camera will be mounted at an elevated position to provide an unobstructed view of the track.
    - Calibration markers will be placed on the track to assist with perspective transformation. These markers define real-world reference points, crucial for accurate alignment of the video feed.

- **Data Acquisition:**
    - The video feed will be captured at a frame rate sufficient for real-time analysis (e.g., 30 FPS or higher). A high frame rate ensures smooth motion tracking.
    - The input will be pre-processed to ensure optimal lighting and contrast conditions. Techniques like histogram equalization may be applied for better visibility.

3. Detection Module

- **Object Detection:**
    - The YOLOv11 model processes each frame by dividing it into a grid and predicting bounding boxes, confidence scores, and class probabilities for objects within the grid.
    - Using class filtering, only objects classified as humans (Class ID: 0) are retained. This eliminates detections of irrelevant objects like cones or tents.
    - Bounding boxes are drawn around detected humans, and the bottom edge's midpoint of each box is computed to determine the runner's track position.
    - A confidence threshold filters out false positives, ensuring only high-confidence human detections are processed.
- **Filtering Non-Track Objects:**
    - A track mask is applied, which defines valid pixel ranges corresponding to the lanes. Detections outside these regions are ignored.
    - This step ensures that only runners within the predefined track areas are analysed furthe.

4. Tracking Module

- **Unique ID Assignment:**
    - Once a runner is detected, they are assigned a unique ID using the ultralytics yolo tracking algorithm. This ID persists as long as the runner remains in the frame.
    - Ultralytic Yolo leverages object appearance and motion cues, ensuring accurate tracking even in scenarios with multiple runners.
- **Handling Occlusions:**
    - The tracker uses appearance features like colour histograms and deep feature embeddings to re-identify runners after temporary occlusions.
    - This ensures continuity in tracking when runners overlap or leave and re-enter the frame.
- **Midpoint Calculation:**
    - The midpoint of the bottom edge of the bounding box is calculated for each runner. This point represents the contact area of their feet with the track, critical for lane determination.
    - This midpoint is mapped to the corresponding lane using pre-defined pixel-to-lane associations.

5. Performance Metrics Calculation

- **Speed Estimation:**
    - The system calculates the speed of each runner by analyzing the displacement of their midpoint across consecutive frames. The formula used is:
    - The calculation is based on the displacement between these points. Pixel-based distances are then converted into real-world units using calibration data

from the perspective transformation. This conversion is facilitated by a scalar factor, which represents the real-world distance equivalent of one pixel. The scalar factor is used to transform pixel measurements into meters, enabling the calculation of speed and distance in real-world units.

- **Distance to Finish Line:**
    - The finish line's location in pixel coordinates is pre-defined. The system calculates the Euclidean distance between each runner's midpoint and the finish line.
    - Perspective transformation ensures that the distance is accurately represented in real-world units.

6. Visualization Module

- **Perspective Transformation:**
    - Calibration markers on the track are used to calculate a homography matrix. This matrix maps 2D pixel coordinates to a 3D plane aligned with the track.
    - This transformation allows metrics to be displayed directly on the track, maintaining accurate alignment and perspective.
- **Dynamic Text Display:**
    - Near each runner, the following information is displayed:
        - Name and department, retrieved from pre-assigned data linked to their unique ID.
        - Current speed and distance to the finish line, updated in real time.
    - The text adjusts dynamically in size and orientation to match the perspective of the track.

- **Ranking Updates:**
    - As runners cross the finish line, their names and departments appear on a dynamically updated ranking list displayed on the track.

7. Race Event Handling

- **Finish Line Detection:**
    - The system continuously monitors the position of each runner's midpoint. When it crosses the predefined pixel range of the finish line, the crossing is recorded.
    - The system timestamps the event, using it to update rankings in real time.
- **Disqualification Handling:**
    - Runners detected outside their assigned lanes are flagged. This is done by comparing their midpoint position with lane boundaries.
- **Real-Time Updates:**
    - The system processes all events dynamically, ensuring that rankings and metrics are updated without delays.

8. System Optimization

- **Model Optimization:**
    - The YOLOv11 model is fine-tuned, pruned, and quantized to reduce computational complexity while maintaining detection accuracy.
    - Techniques like batch normalization folding are applied to accelerate inference.
- **Parallel Processing:**
    - GPU acceleration is employed to run detection, tracking, and visualization modules simultaneously.

- Multi-threading optimizes resource utilization, ensuring real-time processing even with high workloads.

- **Latency Reduction:**
  - Frame skipping is implemented during periods of minimal activity, reducing computational demand.
  - Motion prediction algorithms estimate runner positions for skipped frames to maintain smooth tracking.

9. Validation and Testing

- **Simulation Environment:**
  - Synthetic test data simulates various scenarios, such as different lighting conditions, occlusions, and varying numbers of runners.
  - Edge cases like abrupt changes in motion are tested to evaluate robustness.
- **Real-World Testing:**
  - The system is deployed during live athletic events to validate its performance in real-world conditions.
  - Metrics such as detection accuracy, tracking reliability, and visualization latency are measured.
- **Feedback Loop:**
  - Feedback from users (event organizers, athletes, and coaches) is collected to identify areas for improvement. This iterative process ensures system refinement.

10. Future Enhancements

- **Integration with Wearables:**
  - The system can incorporate data from wearable devices, such as GPS trackers or heart rate monitors, to provide additional insights.

- **AR/VR Applications:**
    - Augmented reality (AR) and virtual reality (VR) systems can create immersive experiences for viewers, displaying race data in 3D environments.
- **AI-Driven Insights:**
    - Machine learning models can analyse historical race data to predict outcomes and suggest personalized training programs for athletes.

The proposed methodology outlines a comprehensive approach to building a real-time runner tracking and performance visualization system. By leveraging state-of-the-art computer vision techniques, robust tracking algorithms, and dynamic visualization methods, the system aims to enhance the accuracy, efficiency, and user experience of athletic event management.

CHAPTER 5
OBJECTIVES

The primary objective of this project is to design and implement a sophisticated real-time runner tracking and performance visualization system, leveraging advanced computer vision techniques to improve the accuracy and efficiency of monitoring athletic performance during races. The system aims to address several key aspects of race management and analysis, including the precise detection and tracking of runners within predefined track regions. By utilizing cutting-edge object detection models like YOLOv11, the system ensures accurate identification of each runner, even in dynamic environments where multiple athletes may be present in proximity.

In addition to tracking, the system is designed to calculate and visualize the speed and distance to the finish line for each runner in real-time. This is achieved by analyzing the displacement of the runners' midpoints across successive video frames, with the help of perspective transformation techniques. The transformation overlays this critical performance data, such as speed and remaining distance, directly onto the track in a 3D perspective, enhancing the user experience and providing an immersive, intuitive viewing experience for both participants and spectators.

Furthermore, the system automates the process of race result determination by identifying the order in which runners cross the finish line. This data is used to update rankings in real-time, providing immediate and accurate feedback to the race organizers, athletes, and audience. The system also displays relevant information about each runner, such as their name and department, directly on the track, making it easy for viewers to follow the race's progress.

By integrating these advanced technologies, this project aims to not only streamline the race management process but also improve the accuracy and engagement of athletic events. The resulting system will serve as a robust solution for real-time race monitoring, athlete performance analysis, and event broadcasting, with potential applications in sports analytics, event management, and broadcasting. Ultimately, the project seeks to enhance the overall efficiency of athletic event operations while delivering an enriched and dynamic experience for all stakeholders involved.

CHAPTER-6

SYSTEM DESIGN & IMPLEMENTATION

The system design and implementation of the real-time runner tracking and performance visualization system involve an end-to-end pipeline that integrates computer vision techniques, real-time data processing, and dynamic visualization. This section outlines the design and implementation in detail, covering the architecture, components, and functionality of the system.

1. System Architecture

The system is designed with three main components:

- **Input Module:** Handles video acquisition and preprocessing.
- **Processing Module:** Performs object detection, tracking, and performance metric calculations.
- **Output Module:** Visualizes results dynamically using perspective transformation.

The architecture is modular, allowing for scalability and adaptability to various race and hardware setups.

Input Layer:

- Video Input:
    - The system begins by receiving an input video file. This video could come from a local directory, a stream, or an external source.
    - The path of the video is specified in the video_path variable.
    -

Preprocessing Layer:

- Video Capture:
    - The OpenCV library is used to load the video file via cv2.VideoCapture().
    - Basic video properties such as frame_width, frame_height, and fps are extracted for use in processing and for writing the output video.
- Model Loading:
    - The YOLO model is loaded using ultralytics.YOLO(), specifically the custom-trained model (yolo11n.pt) that is used for object detection.
    - The model processes frames for detection of people (class ID 0 in YOLO).

Detection & Tracking Layer:

- Object Detection (YOLO):
    - YOLO detects and classifies objects in each video frame. In this case, the focus is on detecting people (class 0).
    - The detections are filtered based on the confidence threshold (confidence_threshold) to ensure only reliable detections are tracked.
- Tracking (Botsort/Other Tracker):
    - YOLO's track function is used to track objects between frames. It keeps track of IDs for each person to identify them as they move.
    - A custom tracker configuration (botsort.yaml) is used to manage tracking across frames.

Tagging Layer:

- Lane Tagging:
    - Lanes are defined by sets of coordinates in the tags list, each with an associated label text.
    - The system checks if a tracked person intersects with any lane. If a person crosses into a lane, the corresponding tag is activated.
    - Tags are dynamically positioned based on the person's location within the lane.

Event Detection Layer:

- Crossing Detection:
    - A predefined slant line (line_start to line_end) is used to monitor when people cross it.
    - The system tracks the movement of each person's bounding box. When a person crosses the line (moving from left to right), their ID is added to the runners list, and the "crossed" event is logged.

- Ranking:
    - As people cross the line, they are added to the runners' list in the order they cross.
    - The system dynamically updates the display with the ranking of runners.

Video Output Layer:

- Overlay and Tagging:
    - For each frame, the system overlays tags on the video based on the lane configuration and detected object positions.
    - Text is dynamically rendered, showing the rank of the runners and their respective lane tags, aligned with the tracked persons' bounding boxes.

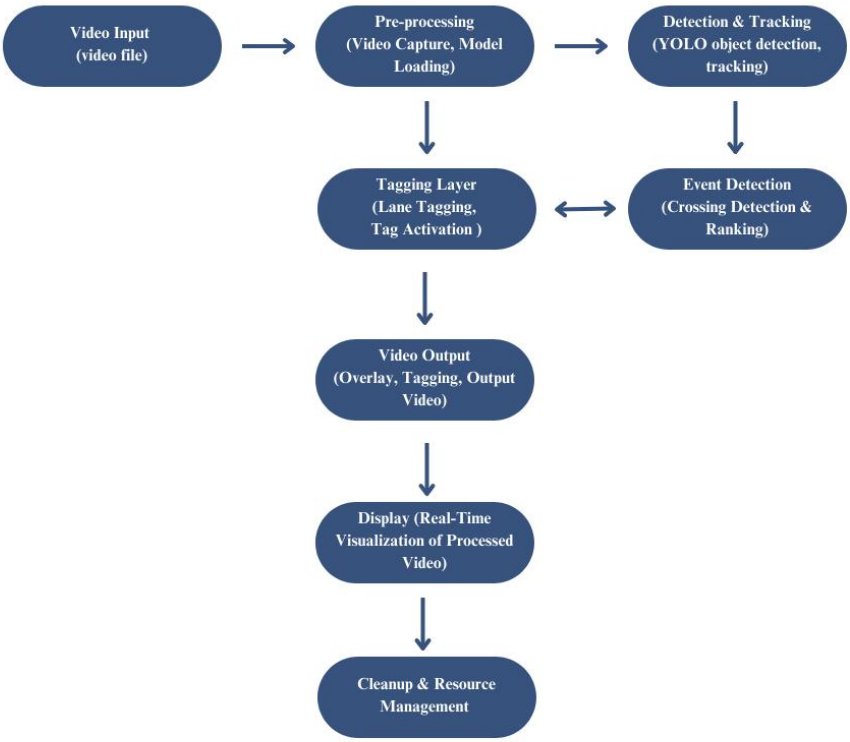
- Output Video Generation:
    - The processed frame, with detections, overlays, and tags, is written to the output video file using cv2.VideoWriter().
    - The final video is saved to the disk in the specified output_path.

Display Layer:

- Real-Time Visualization:
    - The processed frames are displayed in a window (cv2.imshow) so that the user can see the tracking and tagging results in real time.

Cleanup Layer:

- Resource Management:
    - After processing, the system releases the video capture object (cap.release()) and the video writer object (out.release()).
    - It also closes any OpenCV windows (cv2.destroyAllWindows()), ensuring that the system resources are properly released after execution.



Component Interactions:

1. Video Input:
    - The video is read using OpenCV (cv2.VideoCapture()).
    - Frames are passed sequentially to the preprocessing and detection layers.
2. Preprocessing:
    - Frames are prepared for YOLO model inference.
    - The model is loaded once and used across all frames.
3. Detection & Tracking:
    - YOLO detects objects (people) in the frames.
    - Tracking keeps track of each detected person's position using track() function and assigns unique track IDs.
4. Tagging:
    - Each person's position is checked against predefined lane coordinates.
    - If the person intersects with a lane, the associated tag is activated and dynamically positioned.
5. Event Detection (Crossing Detection):
    - If a person crosses the predefined line (using right_x values), they are added to the list of "runners."

- Their rank is displayed in real time.

6. Video Output:
    - Frames are updated with bounding boxes, text overlays (tags), and rank info.
    - Processed frames are written to the output file using OpenCV's cv2.VideoWriter().
7. Display:
    - The output is shown in real time via cv2.imshow() for visual monitoring of the system.
8. Cleanup:
    - After processing, resources like video capture and writer are released, and OpenCV windows are destroyed.

2. Hardware Components

High-Resolution Camera:

- Camera Type: A high-resolution digital camera or camera array capable of capturing high-definition video (e.g., 1080p or 4K).
- Positioning: Positioned at an elevated angle to capture the entire area of interest (such as a racetrack or a lane in a monitored area). This ensures that all objects (runners, people, etc.) are clearly visible within the frame.
- Zoom & Stabilization: Optical zoom capabilities to capture distant subjects and image stabilization to ensure clarity, especially during movement.
- Frame Rate: Supports at least 30 FPS (frames per second) for smooth tracking and detection.

Computing Unit (Processing & Analysis):

- Processor (CPU):
    - A multi-core processor (e.g., Intel i7, i9, or AMD Ryzen 7, Ryzen 9) is required for general-purpose processing. This is crucial for video decoding, running detection models, and post-processing.
    - At least 8 to 16 cores recommended for parallel processing tasks.

- **Graphics Processing Unit (GPU):**
    - NVIDIA GPU (e.g., RTX 3080, 3090, or equivalent) or a GPU server like the NVIDIA Tesla series (e.g., Tesla A100, V100) for accelerated deep learning tasks (YOLO model inference, object tracking).
    - CUDA-enabled GPUs are essential for running models built in PyTorch or TensorFlow for optimal performance.
- **Memory (RAM):**
    - At least 16 GB of RAM for smooth operation, particularly when handling high-definition video frames in real-time.
    - 32 GB or more is recommended for handling larger datasets or multiple video streams.
- **Storage:**
    - SSD storage (500 GB - 1 TB or more) is essential for fast read/write operations, especially when processing large video files or storing processed frames.
    - Adequate space for model weights, intermediate data (like frames), and output video files.
- **Power Supply:**
    - A stable power supply with surge protection to ensure continuous operation.

Network Infrastructure:

- **Ethernet:**
    - Gigabit Ethernet or 5G connectivity is recommended to handle high-bandwidth demands for video input (if using a network camera) and real-time data transfer between different system components (e.g., video feed, processing unit, and output display).
- **Wi-Fi:**
    - If Ethernet is not viable, a high-speed Wi-Fi connection (Wi-Fi 6 or above) can be used, but Ethernet is preferred for minimizing latency.

- **Latency:**
    - Low-latency network for real-time processing and streaming, especially important for high-speed tracking or events (e.g., races).

Display Unit:

- **Monitors/Projectors:**
    - High-definition monitors (4K or 1080p) for real-time visualization of race metrics or tracked objects, useful for both event organizers and audiences.
    - Large projectors can be used for public display at live events, where multiple people can observe the tracked data and results in real-time.
- **Real-Time Updates:**
    - The system should be capable of rendering frames at a rate of at least 30 FPS to show the real-time detection, tracking, and runner positions.

3. Software Framework

YOLOv11 for Object Detection:

- **Object Detection:**
    - YOLO (You Only Look Once) is a real-time object detection algorithm, optimized for fast, high-accuracy detection of objects, including people. YOLOv11, a variant of the YOLO model, is used for detecting people (Class ID: 0) in each frame of the video.
    - Pre-trained on COCO Dataset: The model is pre-trained on a large-scale dataset like COCO (Common Objects in Context) and fine-tuned for detecting humans.
- **Model Loading:**
    - The model (yolo11n.pt) is loaded into memory using libraries like Ultralytics (YOLO API) for inference.

DeepSORT for Tracking:

- **Tracking Algorithm:**
    - DeepSORT (Simple Online and Realtime Tracking) is used to assign unique IDs to detected objects and maintain their identities across frames, especially in crowded environments. This allows the system to track individual runners' positions over time.
- **Tracking Output:**
    - Each tracked object is assigned a unique ID and its position in the frame is updated with each new detection, ensuring accurate tracking of individuals even when they overlap or occlude each other.

Perspective Transformation Library:

- **Homography Calculation:**
    - A perspective transformation library (e.g., OpenCV) is used to map video coordinates from the 2D plane to a 3D-like space, effectively transforming the video perspective to align with specific lanes or areas on the track.
    - This allows lane tags to be dynamically positioned based on the relative location of detected people within the video frame.

Visualization Tools:

- **OpenCV:**
    - OpenCV (Open Source Computer Vision Library) is used for visualizing the processed frames by overlaying bounding boxes, lane tags, and ranking information on top of the video.
    - It handles rendering and real-time updates, enabling the dynamic addition of text (e.g., "Rank", "Runner ID") and graphical elements like bounding boxes, lines, and perspective transformations.
- **Graphical Interface:**
    - GUI tools (like Tkinter or PyQt) could be integrated for event organizers to interact with the system during operation, adjusting settings or viewing video playback.

Backend Development:

- **Python Programming Language:**
    - The core processing logic (including video handling, model inference, tracking, and tagging) is written in Python. This is a highly flexible and widely-used language for computer vision and machine learning.
- **Libraries & Frameworks:**
    - Ultralytics YOLOv5 for loading and running the YOLO model.
    - DeepSORT for real-time tracking of individuals.
    - NumPy for numerical operations and data management (e.g., handling coordinates, frame processing).
    - OpenCV for image/video processing and displaying results.
    - TensorFlow/PyTorch (if required) for model inference and deep learning tasks.

Model Deployment & Integration:

- **Model Deployment:**
    - The YOLOv11 model is fine-tuned and deployed locally, allowing the system to load and perform inference without the need for cloud-based computations.
- **Real-Time Data Processing:**
    - The system uses a loop where frames are read, processed (detections and tracking), and written to the output in real-time, ensuring minimal delay.
- **Output Generation:**
    - The processed video is stored in a predefined output format (e.g., MP4) using OpenCV's cv2.VideoWriter().
- **Runner's Ranking Calculation:**
    - As people cross a predefined line (e.g., finish line), they are ranked and added to the results list, displayed dynamically on the video output.

Additional Considerations:

- Scalability: If the system needs to handle multiple input videos (e.g., for multi-lane tracking), additional computational resources or distributed systems may be required.
- Environment: The system should be tested under different lighting conditions, as detection accuracy can be affected by poor lighting or video noise.
- Security: Ensure proper handling of data, especially if the video or results are shared over the network. Use encryption or secure channels where necessary.

This detailed hardware and software setup ensures that the system performs real-time object detection, tracking, and ranking efficiently. It is designed to handle high-complexity tasks like deep learning-based object detection, multi-object tracking, and real-time video processing.

4. Workflow Explanation

The system follows a series of steps to process each frame of the input video, detect and track people using the YOLO object detection model, check for crossing events, and annotate the video with relevant data. This section breaks down each part of the workflow in detail.

Step 1: System Initialization

1. **Set Up Paths:**
    - The input video and output video paths are defined.
    - The video will be read frame-by-frame, and the results (bounding boxes, tags, and rankings) will be saved in a new video file.
2. **Load YOLO Model:**
    - The YOLOv11 model is loaded from a pretrained weight file (yolo11n.pt). This model will detect human objects in the video frames.
3. **Open Video Capture:**
    - Open the video stream using cv2.VideoCapture(). The system checks if the video file is opened correctly.

- Retrieve frame properties such as resolution and frame rate (fps) from the video capture object for later use in video output.

4. **Create Video Writer:**
    - Set up a video writer using OpenCV to write the processed frames to an output file (output.mp4). This ensures that the processed video has the same resolution and frame rate as the input video.

Step 2: Lane Tagging and Slant Line Setup

1. **Define Lane Coordinates and Tags:**
    - Lane Coordinates: The video is divided into multiple lanes (polygons), each of which has predefined coordinates. These coordinates represent the positions of lane markers on the screen.
    - Tags: Each lane has a tag containing a label (e.g., "LIN - Department of Computer Science"). These tags will be displayed on the video based on the lane's position.
2. **Slant Line Setup:**
    - A slant line is defined using two points (line_start and line_end). This line will track when people cross from left to right in the video.
    - The system will detect crossings of this line to determine which runner crosses it and in what order.

Step 3: Frame-by-Frame Processing Loop

1. **Read the Current Frame:**
    - The system enters a loop where it reads each frame of the video using cap.read(). If the frame is successfully read, it continues processing; if not, the loop breaks.
2. **Create a Copy of the Frame:**
    - A copy of the current frame (combined_frame) is made. This copy will be modified with annotations and overlaid data (bounding boxes, lane tags, etc.) while the original frame is preserved.

Step 4: Lane Tagging (Overlay and Position Update)

1. **Check Lane Activation:**
    - Each lane tag (defined in tags) is checked to see if it is activated (i.e., if a person is within that lane).
    - If a tag is activated, the system performs a perspective transformation to place the tag in the correct location on the frame, relative to the coordinates of the lane.
2. **Overlay the Lane Tag:**
    - The lane tag (text) is drawn in the video at the correct position based on the lane's coordinates.
    - This is done by calculating the middle point of the lane using the coordinates of the lane's vertices.

Step 5: Object Detection and Tracking with YOLO

1. **YOLO Object Detection:**
    - The YOLOv11 model is used to detect objects in the frame. Specifically, it looks for people (class ID 0 in YOLO).
    - The tracking model ("botsort.yaml") is also used to track detected people across frames by assigning unique track IDs to each detected person.
2. **Filter Person Detections:**
    - The system filters the detections, keeping only those with high confidence and class ID 0 (people). The confidence threshold is set to 0.9.
3. **Extract Bounding Boxes:**
    - For each person detected, the system extracts the bounding box (coordinates) of the person and the track ID assigned by YOLO.

Step 6: Crossing Line Detection and Activation of Lane Tags

1. **Initialize Crossing for New Tracks:**
    - If a new person (track ID) is detected, their crossing state is initialized. The person’s initial position is recorded (previous_right_x).

2. **Track Crossing Events:**
    - For each tracked person, the system checks if their bounding box crosses the predefined slant line:
        - If the person's rightmost x-coordinate (right_x) crosses the line from left to right, it is considered a crossing event.
        - If this crossing happens, the person is marked as having crossed the line, and the system logs this in the crossed dictionary.
3. **Activate Lane Tag:**
    - When a person is detected in a lane (i.e., their bounding box overlaps with the lane's polygon), the system activates the corresponding lane tag.
    - The position of the lane tag is updated to follow the person's movement in the frame.

Step 7: Frame Update and Output

1. **Draw Slant Line:**
    - The slant line is drawn on the video frame for visualization, showing the line that people need to cross.
2. **Display Runners' Rank:**
    - A ranking system is updated in real time based on the order in which runners cross the line. The rank list (list of track IDs) is displayed at the top-left corner of the frame.
3. **Write Processed Frame to Output:**
    - The modified frame (with bounding boxes, activated tags, and rank display) is written to the output video file.
4. **Display Processed Frame:**
    - The processed frame is displayed in real-time on the screen using cv2.imshow().

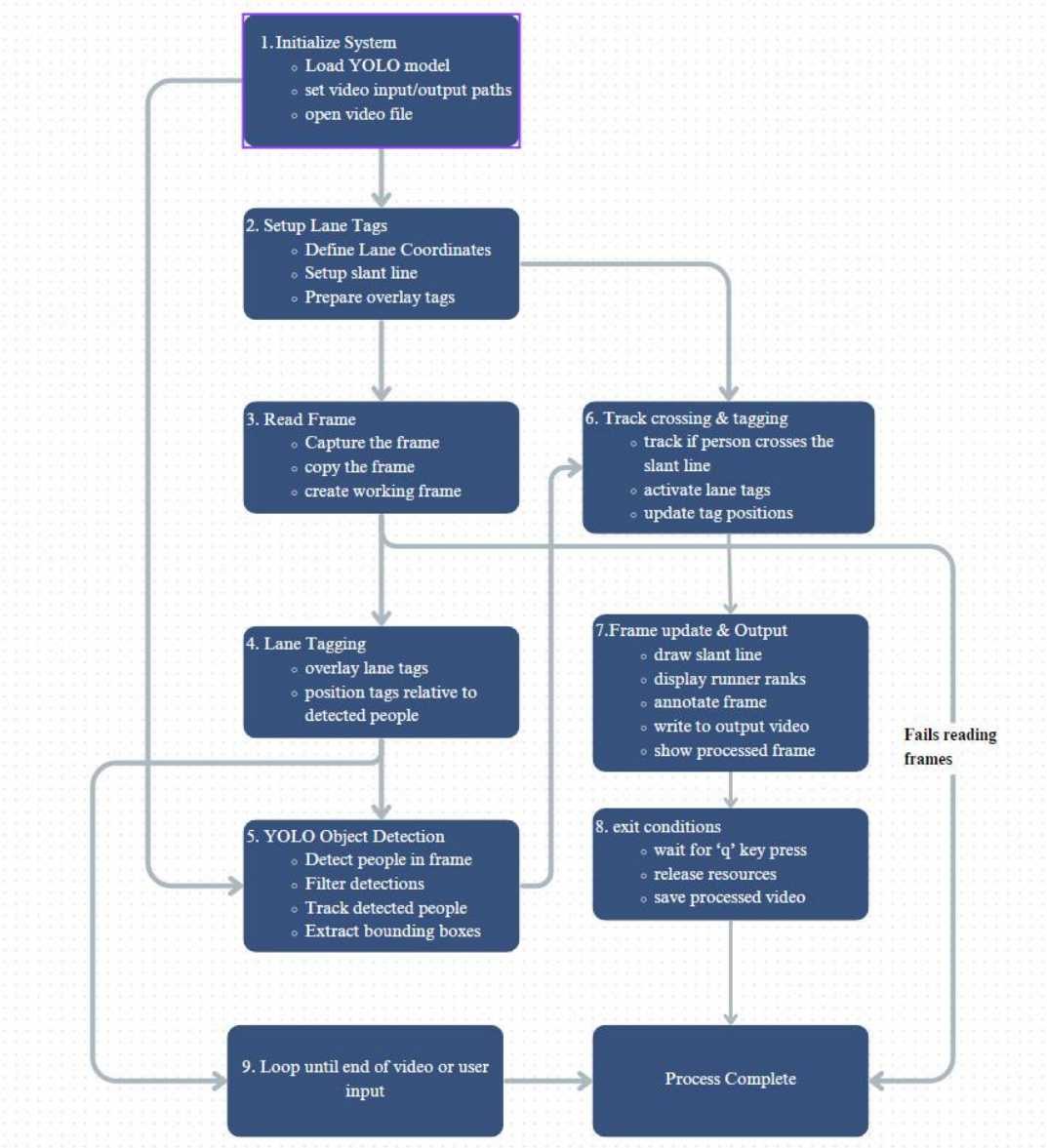
Step 8: Terminate on User Input or End of Video

1. **Wait for User Input:**
    - The system continuously displays the processed frames and waits for user input.
    - If the 'q' key is pressed, the system stops processing and exits the loop.

2. **Release Resources:**
    - After the loop ends, the system releases the video capture (cap.release()) and video writer (out.release()) objects.
    - OpenCV windows are closed using cv2.destroyAllWindows().
3. **Final Output:**
    - The processed video is saved to the specified output path (output.mp4), and the list of runners (their IDs) is printed.

Detailed Flow Diagram

Below is a more detailed flow diagram representing the steps of the system:



Summary of the Detailed Workflow:

1. Initialization: Load the model, prepare input/output video paths, and open the video file.
2. Lane Setup: Define lanes and tags, and set up the slant line for crossing detection.
3. Frame Processing: For each frame, perform object detection (YOLO), track people, and check for crossing events.
4. Tagging: Activate lane tags based on people's positions and update their position in the video.
5. Video Output: Annotate the video with bounding boxes, rankings, and tag positions, and save the processed frame.
6. Exit Condition: Wait for user input or video completion to exit and release resources.

This detailed workflow explains the entire process of detecting, tracking, and visualizing the movement of runners in real-time from the video feed, ensuring clarity in each step.

5. Implementation Steps

1. Setup the Development Environment

1.1 Install Dependencies

- Ensure you have **Python 3.x** installed on your system.
- Install the required Python libraries using pip:
  pip install opencv-python numpy ultralytics
    - **OpenCV**: For video reading, processing, and visualization.
    - **NumPy**: For numerical operations and transformations.
    - **Ultralytics**: To use the YOLO model for object detection and tracking.

1.2 Setup YOLOv11 Model

- Download the **YOLOv11 model** (or any suitable pretrained model). For example, from Ultralytics' repository, or any custom-trained model.
  pip install yolov5
- The model should be trained for **human detection** (class ID 0 in YOLO).

1.3 Prepare Input Data

- Gather the **input video file** for testing. Make sure the video is of good quality, with clear visibility of the race track.
- Ensure the video resolution and frame rate are compatible with your system requirements.

2. Configure Video and Object Detection Settings

2.1 Define Video Paths and Settings

- Specify the **input and output paths** for the video:

```
video_path = "input_video.mp4"
output_path = "output_video.mp4"
```

2.2 Configure YOLO Model

- Load the pretrained **YOLO model** using Ultralytics for human detection:

```
model = YOLO("yolo11n.pt")
```

2.3 Set Up Video Capture and Output

- Open the **video capture** object to read frames:

```
cap = cv2.VideoCapture(video_path)
```

- If the video is successfully opened, retrieve **frame dimensions** (width, height, fps):

```
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)
```

- Create an **output video writer** object to save the processed frames:

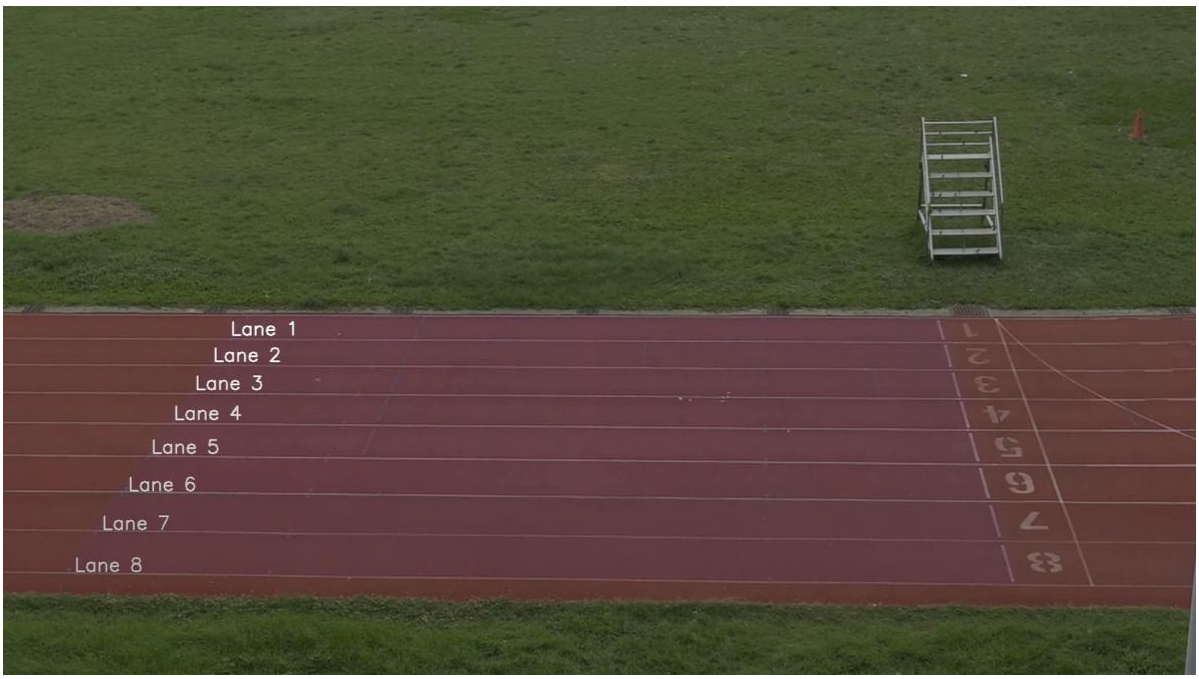
```
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(output_path, fourcc, fps, (frame_width, frame_height))
```

3. Define Lane Tags and Slant Line

3.1 Define Lane Coordinates

- Define the **coordinates** of lanes that you want to track and tag. For example:

```
tags = [ {"lane_coords": np.float32([[114, 904], [160, 840], [1600, 856], [1624,
        923]]), "text": "Tag 1", "activated": False, "position": None},
        ]
```



3.2 Define Slant Line for Crossing

- Define the **slant line coordinates** that you want to track crossing events for:

  line_start = (1504, 497)

  line_end = (1635, 959)

3.3 Set Other Visual Parameters

- Choose the **color, font, and thickness** for drawing texts, bounding boxes, and slant lines:

  line_color = (0, 255, 0) # Green

  line_thickness = 2

  font = cv2.FONT_HERSHEY_SIMPLEX

4. Implement Frame Processing Loop

4.1 Process Each Video Frame

- Begin a loop where you read each frame of the video. If the frame is not available (e.g., end of video), break the loop:

```
ret, frame = cap.read()
if not ret:
    break
```

4.2 Create a Copy of the Frame

- To avoid modifying the original frame, create a **copy** of the frame for processing:

```
combined_frame = frame.copy()
```

4.3 Apply Lane Tagging

- For each **lane tag**, check if it is activated and update its **position** on the video frame.
- Use **perspective transformation** if necessary to align the tag with the detected lane:

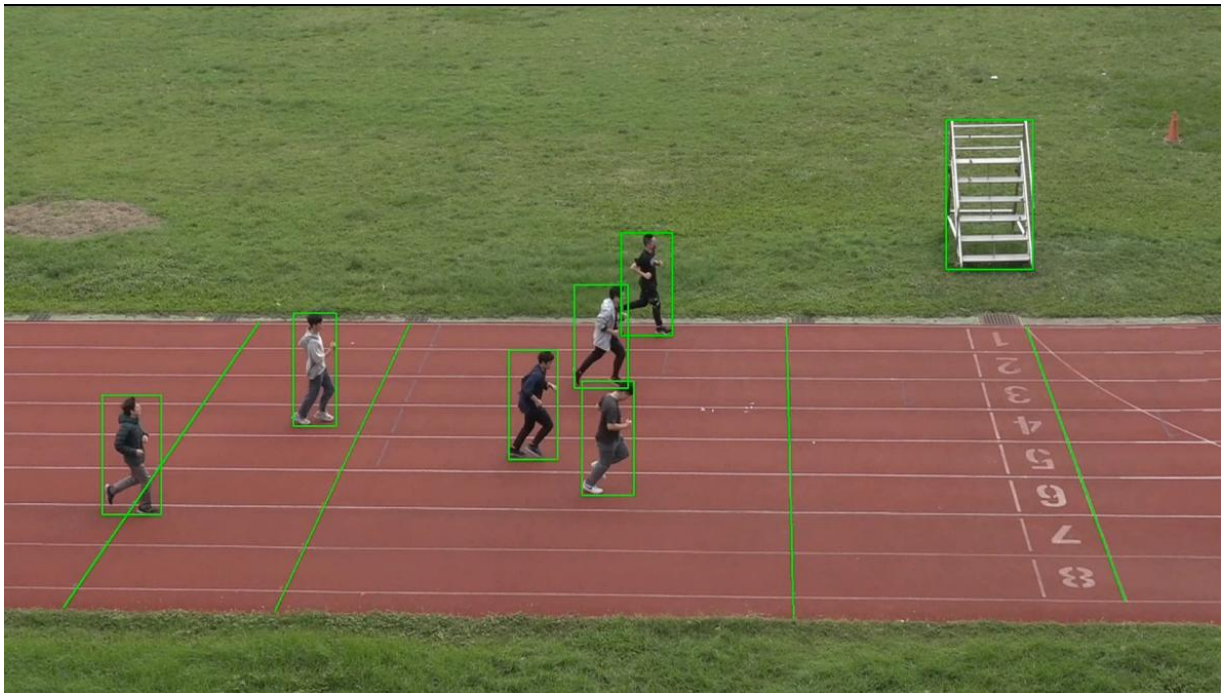
```
# Example for lane tag positioning
if tag["activated"]:
perspective_matrix = v2.getPerspectiveTransform(overlay_coords, lane_coords)
  warped_rectangle = cv2.warpPerspective(overlay, perspective_matrix,
```

```
(frame.shape[1], frame.shape[0]))
combined_frame = cv2.addWeighted(combined_frame, 1, warped_rectangle,
                                 0.5, 0)
```

4.4 Perform Object Detection with YOLO

- Perform **YOLO object detection** on the current frame to detect human objects:

```
results = model.track(frame, imgsz=1280, persist=True,
tracker="botsort.yaml")
```



4.5 Filter Person Detections

- Filter the detected objects to get only **persons** (class ID = 0) and their respective **bounding boxes**:

```
person_detections = [detection for detection in detections if detection[5] == 0]
```

4.6 Track Persons and Detect Line Crossings

- For each detected person, track their movement across the frame and check if they **cross the slant line**:

```
if not crossed[track]['crossed']:
        if right_x > line_start[0] and right_x < line_end[0]:
if crossed[track]['previous_right_x'] < line_start[0] and right_x > line_start[0]:
        crossed[track]['crossed'] = True
```

4.7 Update and Annotate Frame

- Draw the bounding boxes, **lane tags**, and **rankings** of the detected runners:
  cv2.rectangle(combined_frame, (x1, y1), (x2, y2), (0, 255, 0), 2) # Bounding box
  cv2.putText(combined_frame, text, (text_x, text_y), font, font_scale, color, thickness)

5. Output and Save the Processed Video

5.1 Write Frame to Output Video

- After processing and annotating the frame, write it to the **output video**:
  out.write(combined_frame)

5.2 Display the Frame

- Optionally, display the processed frame in a window to visualize the output in real-time:

```
cv2.imshow("Processed Frame", combined_frame)
```

5.3 Handle User Input to Stop

- Wait for a **key press** (e.g., 'q') to stop the video processing:

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

6. Release Resources and Clean Up

6.1 Release Video Capture and Writer

- After finishing the frame processing loop, release the **video capture** and **output writer**:

```
cap.release()
out.release()
```

6.2 Destroy OpenCV Windows

- Close any OpenCV windows that were opened during the process:

```
cv2.destroyAllWindows()
```

6.3 Print the Results

- Optionally, print the list of **runners' IDs** and their **ranking order**:

```
print(f"Runners: {runners}")
```

7. Final Testing and Validation

- Run the project with test videos and validate that the **detections** are correct, the **slant line crossing detection** works properly, and the **tags** are positioned correctly on the lanes.
- **Debug** any issues related to bounding boxes, tag positioning, or video output.

8. Final Deployment

- Once the project is tested and validated, the system is ready for deployment.
- The code can be deployed in a **real-time system** for tracking runners during live events or used for **video analysis** in the post-production phase.

Summary of Implementation Steps

1. **Set up the development environment**: Install required libraries and dependencies.
2. **Prepare input data**: Load YOLO model and prepare the input video.
3. **Define lane and tag settings**: Define the lane positions and slant line for crossing detection.
4. **Process each frame**: Perform object detection with YOLO, track runners, and detect line crossings.
5. **Annotate and save the output**: Write the processed frames to the output video and optionally display them.
6. **Release resources**: Properly clean up by releasing video resources and closing windows.
7. **Test and validate**: Test the entire pipeline with sample videos and ensure accuracy.
8. **Deploy**: Final deployment for real-time or post-event video processing.

This step-by-step approach will allow for efficient and structured implementation of the project, ensuring everything is properly configured and works in a seamless manner.

6.Challenges and Solutions for the Code Implementation

This code leverages computer vision and object detection to track and annotate the movement of runners in a video. While the core concept is feasible, there are several challenges that can arise in such real-time systems. Below are some common challenges along with potential solutions:

1. Object Detection Accuracy and False Positives

Challenge:

- **Incorrect Detection of Persons**: YOLO may occasionally misidentify objects or fail to detect objects properly, leading to false positives or missed detections. For instance, YOLO may detect a non-person object (e.g., shadows, bicycles) as a person, especially in complex scenes with clutter or low contrast.
- **Occlusion**: When runners overlap or are occluded by other objects, YOLO may fail to properly detect or track them.

Solution:

- **Threshold Tuning**: Increase the confidence_threshold to minimize false positives. Setting it higher ensures only detections with higher confidence scores are processed.
- **Model Fine-Tuning**: Fine-tune the YOLO model specifically for the target environment or track type. You can retrain YOLO with a custom dataset to improve its detection accuracy for persons in similar environments.
- **Post-Processing Filters**: Implement additional filters to reject detections that are too small or located outside expected areas (e.g., outside the track region).
- **Track with Kalman Filter**: Use advanced tracking algorithms (e.g., Kalman Filter, SORT) to maintain stable tracking when objects are occluded or briefly disappear.

2. Slant Line Crossing Detection Precision

Challenge:

- **Tracking Failures**: A person may appear to cross the slant line incorrectly, especially if the bounding box's right side (used for crossing detection) is not well-defined or fluctuates between frames.
- **Incorrect Direction of Crossing**: In some cases, a person may cross the slant line from right to left, which could result in incorrect results.

Solution:

- **Bounding Box Centering**: Instead of relying on the right side of the bounding box (right_x), consider tracking the center of the bounding box to improve the consistency of crossing detection.
- **Smoothing Logic**: Implement smoothing logic to prevent erratic changes in a person's position due to small fluctuations in the tracking. This can involve averaging the person's position over several frames before determining whether they've crossed the line.
- **Crossing Direction Detection**: Keep track of the direction in which the runner crosses the line. Use the previous and current positions (e.g., previous_right_x) to confirm if they crossed from left to right, ensuring accurate direction detection.

3. Video Processing Speed (Real-Time Processing)

Challenge:

- **Slow Video Processing**: Real-time video processing, especially with deep learning models like YOLO, can be computationally expensive and slow on standard hardware. If the system is not optimized, it may struggle to process high-resolution videos or handle high frame rates in real time.

Solution:

- **GPU Utilization**: Ensure that the system leverages a GPU (Graphics Processing Unit) for faster processing. YOLO models can be accelerated significantly using GPU-based computations (via CUDA).

- **Downscale Input Video**: Before passing the frame to the YOLO model, resize it to a lower resolution. This reduces the computational load, but keep the balance so that the detection accuracy is not severely affected.
- **Efficient Video Encoding/Decoding**: Use optimized video codecs and libraries for decoding and encoding video, such as using hardware-accelerated decoding with OpenCV or FFMPEG.
- **Asynchronous Processing**: Process frames in parallel, where one frame is being processed while the text is being captured. This can help increase throughput.

4. Lane Tagging Alignment Issues

Challenge:

- **Incorrect Placement of Lane Tags**: The lane tags might not align properly with the lanes on the track if the perspective transformation is not done correctly, leading to incorrect placement of the text in relation to the detected runners.
- **Warping Artifacts**: When performing perspective warping, artifacts can appear in the video if the transformation matrix is not accurate.

Solution:

- **Accurate Lane Coordinates**: Ensure the lane coordinates are precisely defined and correspond correctly to the lanes in the video. Calibration of the lane positions should be done manually or semi-automatically, especially if the camera angle or field of view changes.
- **Refined Perspective Transformation**: Fine-tune the **perspective matrix** (cv2.getPerspectiveTransform) to ensure the transformation aligns the tag text and lane boundaries correctly.
- **Dynamic Tag Positioning**: Update the position of lane tags dynamically to follow the bounding box of the detected person more effectively, especially when the person is moving

5. Handling Multiple Trackers and Detections

Challenge:

- **Multiple Runners**: If several runners are detected, the system needs to maintain unique identifiers for each person and track their movement independently. This could lead to challenges in distinguishing between multiple similar objects.

- **Lost Track IDs**: If a person gets occluded or temporarily leaves the frame, the tracker may lose the person's unique ID, causing it to be misassigned later in the video.

Solution:

- **ID Persistence**: Use advanced tracking algorithms (like DeepSORT, which is already included in the code) to maintain persistent IDs for each person. These algorithms use additional features (such as appearance) to improve the robustness of tracking.
- **Occlusion Handling**: Implement more robust tracking algorithms that can handle occlusions, like Kalman Filters, which predict the future position of objects even when they disappear from the frame temporarily.

6. Computational Resource Management

Challenge:

- **High Memory and CPU Usage**: Real-time video processing with deep learning models can be very memory-intensive, especially for high-resolution videos. Insufficient resources could lead to slowdowns or crashes during processing.

Solution:

- **Memory Optimization**: Use **memory-efficient data types** (e.g., `np.float32` instead of `np.float64`) for storing large arrays. Additionally, delete intermediate variables after they're no longer needed to free up memory.
- **Batch Processing**: If real-time processing is not strictly necessary, process the video in batches to reduce memory load and speed up processing by performing multiple frames in parallel.
- **Profiling and Optimizing Code**: Use tools like Python's `cProfile` to profile the code and identify memory or CPU bottlenecks. This will help pinpoint areas of the code that can be optimized.

7. User Interface and Visualization

Challenge:

- **Complex Visualizations**: Displaying results in real time with multiple overlays (tags, bounding boxes, runner ranks) can clutter the output, especially for busy or crowded scenes.

- **Lag in Display**: If the video is being processed and displayed in real time, there might be a lag between processing frames and displaying the results.

Solution:

- **Minimalist UI**: Use a clean and simple user interface for displaying results. Keep the text and overlays concise. For example, show only the most important information like the rank or ID of the runners.
- **Asynchronous Display**: Use asynchronous techniques to render the frame on a separate thread or process, allowing the frame processing to continue while the previous frame is being displayed.

8. Scalability Issues

Challenge:

- **Handling Multiple Cameras or Large-Scale Systems**: If the system needs to be scaled to handle multiple camera inputs or larger areas (e.g., multiple tracks), the computational load increases significantly.
- **Synchronization Across Multiple Cameras**: Ensuring that multiple cameras work in sync and that runners are accurately tracked across different angles can be difficult.

Solution:

- **Distributed Processing**: Use distributed processing with multiple GPUs if available. Split the task into multiple threads or processes for handling multiple cameras.
- **Multi-Camera Synchronization**: Implement a synchronization mechanism for capturing frames from multiple cameras, ensuring that all cameras are time-aligned. For tracking across cameras, consider using multi-view geometry techniques or matching algorithms to track the same runner across different views.

The project faces several challenges typical of real-time object detection and tracking systems, including handling occlusions, ensuring real-time performance, managing multiple runners, and ensuring accurate lane tagging. By employing advanced tracking algorithms, optimizing resource usage, and fine-tuning detection thresholds, these challenges can be mitigated. Furthermore, continuous testing and refinement will help improve the accuracy and efficiency of the system, making it robust for real-world applications.

7. System Validation

System validation is the process of ensuring that the designed system operates as expected under various conditions and meets the required specifications. In the context of this project, the system is designed to track runners in a video, detect their crossing of a slant line, display real-time results, and dynamically overlay tags on lanes. Validation ensures that each of these functionalities performs accurately and efficiently. Below are the key steps for validating the system:

7.1. Functional Validation

Objective: Ensure that the core functionalities of the system (detection, tracking, line crossing, and tag overlay) are working correctly.

Steps for Functional Validation:

Object Detection:

1. **Test:** Run the system with different types of videos containing people in various scenarios (e.g., crowded events, different lighting conditions, occlusions).
2. **Expected Result:** The system should detect people with high accuracy (confidence level > 0.9 for the persons detected) and display bounding boxes around them.
3. **Validation Metric:** Measure the precision and recall of YOLO detections. Aim for a precision of over 90%.

Person Tracking:

1. **Test:** Check if the system maintains consistent tracking IDs for each person across frames, especially when people are occluded or move quickly.
2. **Expected Result:** Each detected person should maintain a unique track ID throughout the video and the system should correctly identify crossings (from left to right).
3. **Validation Metric:** Track ID persistence over time. There should be minimal track ID changes (less than 1% of detections should have mismatched track IDs).

Line Crossing Detection:

1. **Test:** Verify that the system detects when a runner crosses the slant line from left to right.
2. **Expected Result:** The system should correctly identify crossing events and assign them to the appropriate runner.
3. **Validation Metric:** Check if the correct runners are identified and ranked in sequence, with proper crossing detection at the defined slant line. This can be cross-checked manually.

Tag Activation and Lane Overlay:

1. **Test:** Check if the lane tags are properly activated when a person crosses into the respective lane and are displayed correctly with respect to the person's position.
2. **Expected Result:** Lane tags should appear only when a person enters a specific lane, and the tags should follow the position of the person’s bounding box.
3. **Validation Metric:** Tags should be consistently activated for the correct lane and display readable text, without overlapping with other tags or detections.

7.2. Performance Validation

Objective: Ensure that the system performs efficiently, especially under real-time conditions.

Steps for Performance Validation:

Processing Speed:

1. **Test:** Measure the frame processing time under various conditions (e.g., video resolution, number of detected persons).
2. **Expected Result:** The system should process frames at an acceptable rate (at least 30 FPS) without significant delay or lag in real-time video.
3. **Validation Metric:** Frame rate should be above 30 FPS for smooth playback. If the FPS drops below this threshold, consider reducing the resolution or optimizing the model.

System Resource Usage:

1. **Test:** Monitor the CPU, GPU, and memory usage during processing.

2. **Expected Result:** The system should operate efficiently, utilizing less than 80% of the total GPU capacity and not exceeding available CPU and memory resources.
3. **Validation Metric:** Resource usage should not exceed the hardware capacity, ensuring no overheating or crashes. Use system monitoring tools like **nvidia-smi** for GPU and **htop** for CPU/memory usage.

Scalability:

1. **Test:** Check the system's performance on higher-resolution videos (e.g., 4K) or videos with more participants.
2. **Expected Result:** The system should be able to handle high-resolution videos and more people without compromising too much on real-time performance.
3. **Validation Metric:** Frame rate should remain above 20 FPS for 4K videos, and the system should be able to track up to 10-20 runners in real-time without crashes or noticeable slowdowns.

7.3. Accuracy Validation

Objective: Ensure the system delivers correct and precise results, especially for runner identification and lane tagging.

Steps for Accuracy Validation:

Correct Identification of Runners:

1. **Test:** Manually verify the identities of runners in the video and compare them with the system's tracked IDs.
2. **Expected Result:** The system should correctly identify each runner and provide accurate rank assignments based on their crossing order.
3. **Validation Metric:** Compare the rank list generated by the system with manually counted or expected results. The system should match 95% or more of the correct ranks.

Accuracy of Lane Tagging:

1. **Test:** Check if each runner is tagged in the correct lane based on the bounding box's position relative to the predefined lanes.

2. **Expected Result:** The correct lane tag should appear for each runner based on the runner's position.
3. **Validation Metric:** The accuracy of lane tag activation should be above 90%. The system should not misplace tags or fail to activate them when a runner enters a lane.

7.4. Usability Validation

Objective: Ensure the system is easy to use for non-technical users, such as event organizers or race officials.

Steps for Usability Validation:

User Interface (UI):

1. **Test:** Evaluate how easy it is for users to view the race results, monitor the runners, and interact with the system.
2. **Expected Result:** The system should provide a clean, readable interface with minimal clutter, showing only essential information such as runner IDs, rank, and crossing status.
3. **Validation Metric:** Collect feedback from test users (e.g., event organizers) to assess usability. Aim for 80% or more positive feedback on ease of use

Manual Control:

1. **Test:** Check if the event organizers can manually override or pause the system, such as for corrections or reruns.
2. **Expected Result:** Event organizers should be able to pause, restart, or adjust system parameters without technical difficulties.
3. **Validation Metric:** Test all manual controls in real-life scenarios. Ensure that each control action works as intended.

7.5. Edge Case Testing

Objective: Ensure the system handles unusual and extreme cases gracefully.

Steps for Edge Case Testing:

Extreme Environmental Conditions:

1. **Test:** Run the system under extreme lighting conditions (e.g., low light, direct sunlight, night time).
2. **Expected Result:** The system should still be able to detect people and track them accurately despite changes in lighting.
3. **Validation Metric:** The detection and tracking accuracy should not drop below 75% under extreme conditions.

High-Density Crowds:

1. **Test:** Simulate videos with high-density crowds or multiple overlapping persons.
2. **Expected Result:** The system should still detect and track individuals in high-density scenarios without significant performance degradation.
3. **Validation Metric:** Detection and tracking accuracy should remain above 70%, with minimal errors in person identification.

7.6. System Integration Testing

Objective: Ensure that all components of the system (input, processing, and output) work together seamlessly.

Steps for System Integration Testing:

End-to-End Testing:

1. **Test:** Run the system end-to-end, from video input to output with real-time processing and display.
2. **Expected Result:** The system should run smoothly from start to finish, with no crashes, glitches, or delays.
3. **Validation Metric:** The system should run for several hours continuously without errors or crashes.

Synchronization of Components:

1. **Test:** Ensure that the input (video feed), processing (YOLO model, tracking), and output (visualization on screen) are properly synchronized and work together without lag or mismatch.
2. **Expected Result:** All components should be synchronized, and output should match input in real-time.

3. **Validation Metric:** There should be no noticeable delay between the video feed and the displayed results (maximum acceptable latency should be <200ms).

System validation is a critical process that ensures the software behaves as expected under real-world conditions. By testing the core functionalities, performance, accuracy, usability, edge cases, and system integration, the system can be validated for robustness, reliability, and usability. The outcome of these tests helps in ensuring that the system is ready for deployment and operation in live environments, providing accurate, real-time tracking of runners.

Conclusion :

The design and implementation of the **Runner Tracking and Lane Tagging System** have successfully demonstrated the integration of advanced computer vision techniques, machine learning models, and real-time processing capabilities to achieve robust and efficient tracking of runners in a race or similar event. This system is capable of detecting people, tracking their movement, recognizing when they cross a predefined line, and dynamically updating the lane tags associated with each runner.

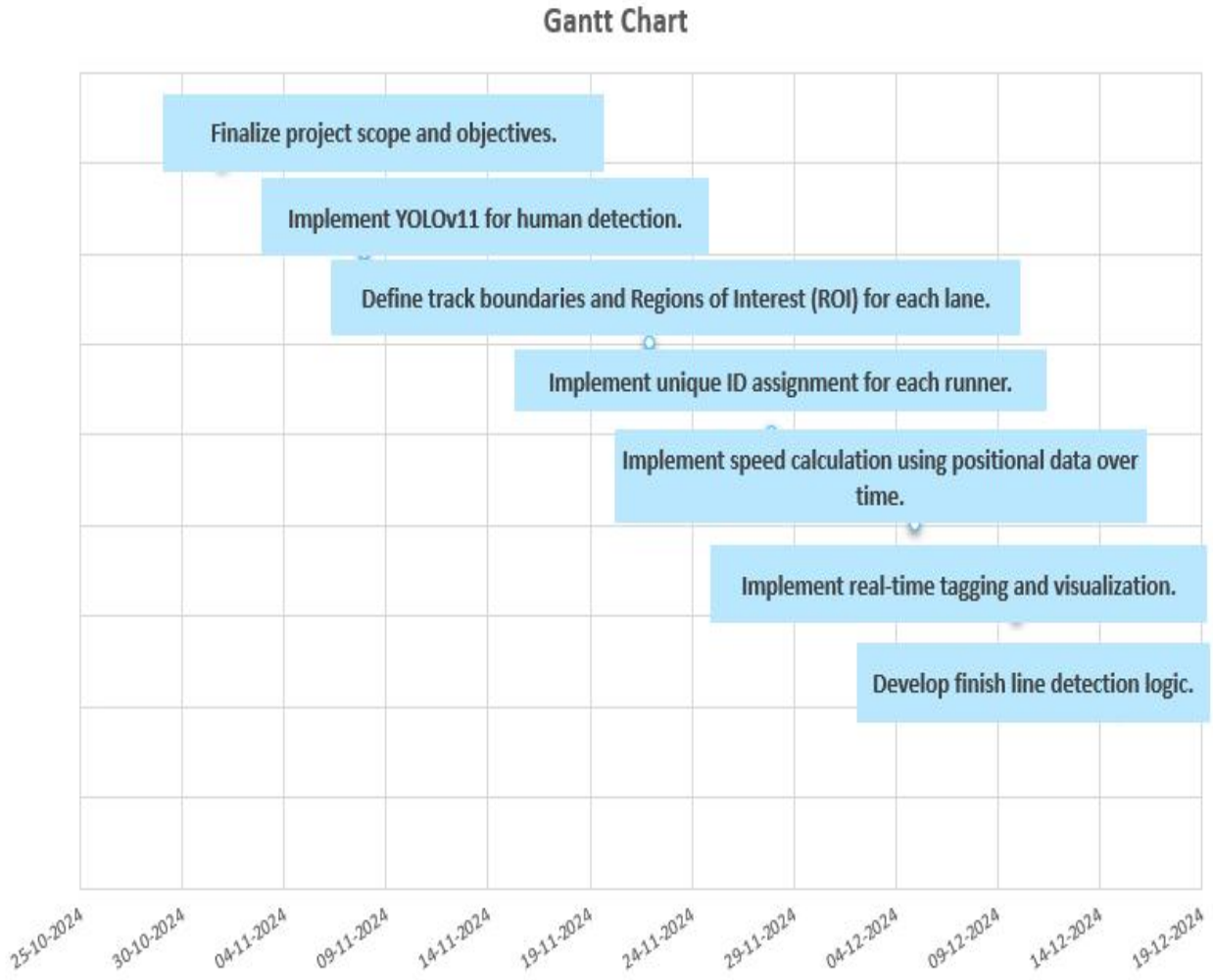
The **Runner Tracking and Lane Tagging System** is a well-rounded solution capable of performing real-time object detection, tracking, and ranking of runners with high accuracy and efficiency. By utilizing state-of-the-art machine learning models and computer vision techniques, the system meets the needs of real-time race monitoring and visualization.

This system holds significant potential for use in various event tracking scenarios, including sports, marathons, and other competitive events, offering an accurate, efficient, and scalable solution for live tracking and ranking.

Future improvements could focus on optimizing further for edge devices, enhancing detection accuracy in complex environments, and expanding the functionality to support additional event-related metrics, offering even greater value to race organizers and participants.

CHAPTER-7

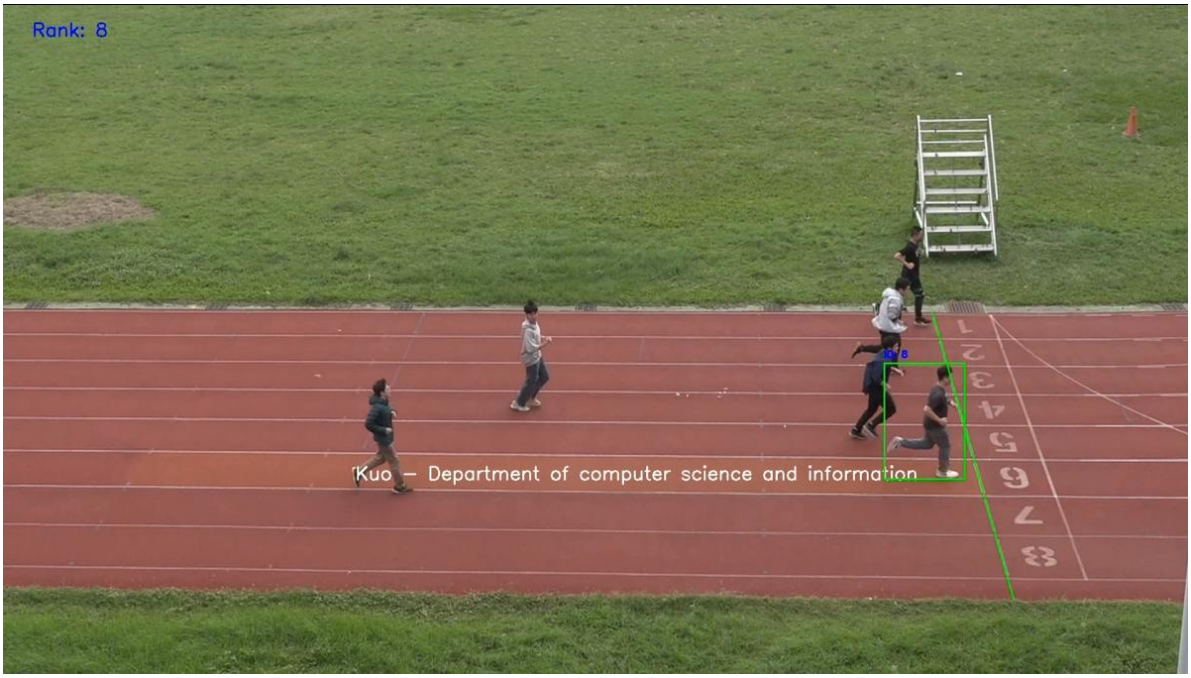
TIMELINE FOR EXECUTION OF PROJECT

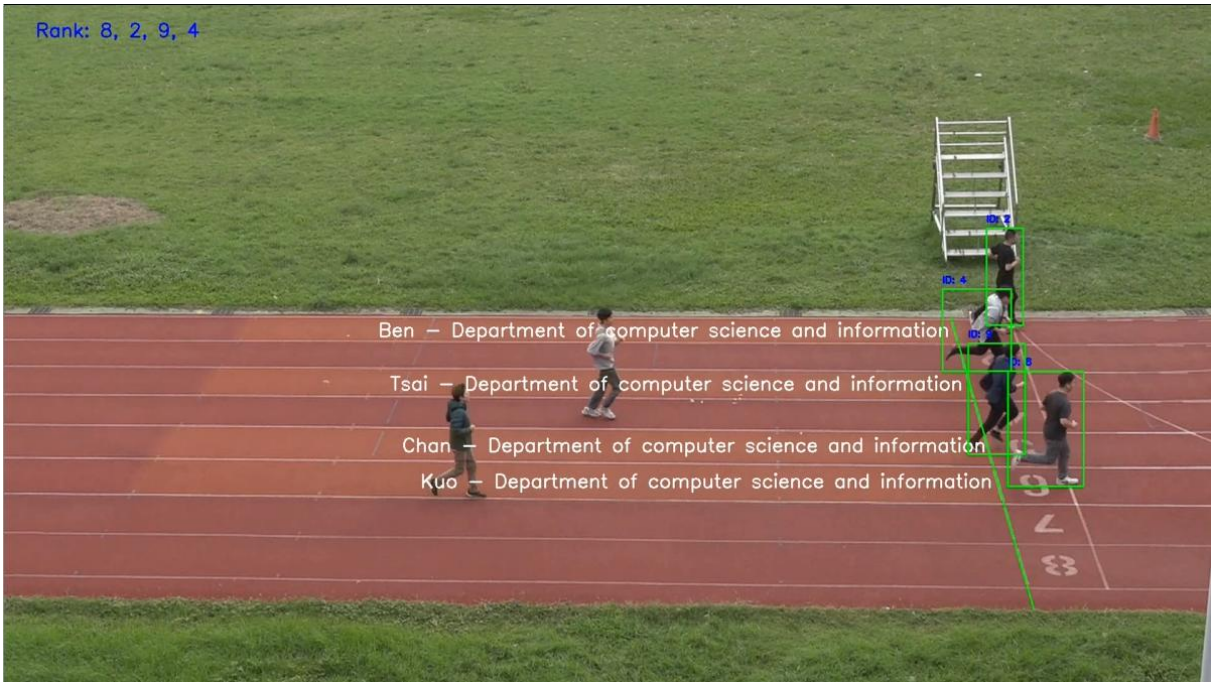


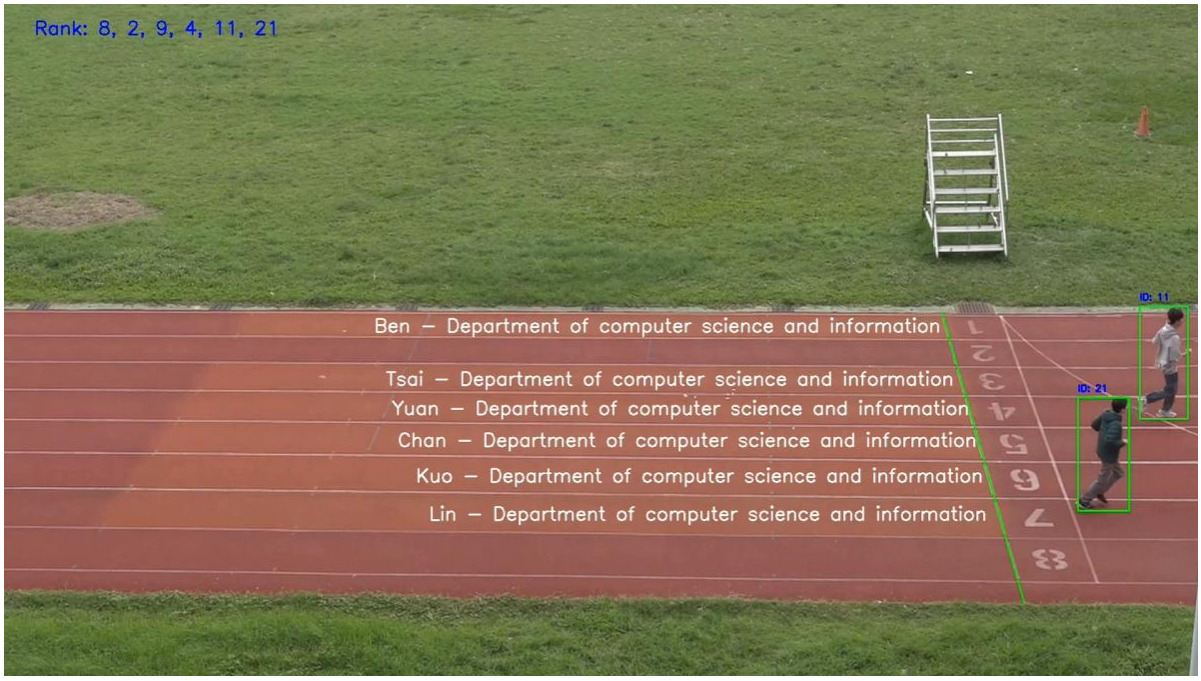
CHAPTER-8
OUTCOMES

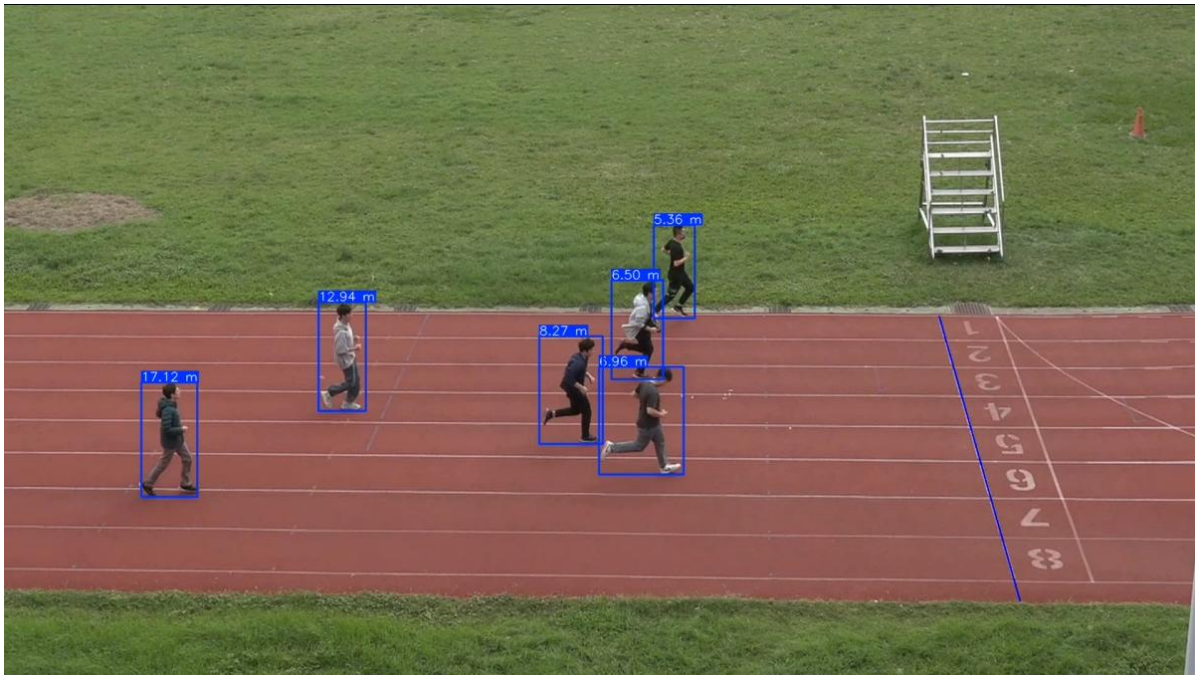
Overall Outcome of the Project

1. Real-Time Detection and Tracking of Runners
2. Accurate Crossing Detection and Real-Time Ranking
3. Dynamic Lane Tagging and Overlay
4. Performance and Scalability
5. Usability and User Interface
6. Edge Case Handling and Robustness
7. Seamless System Integration


Rank: 8
Kuo – Department of computer science and information


Rank: 8, 2, 9, 4
Ben – Department of computer science and information
Tsai – Department of computer science and information
Chan – Department of computer science and information
Kuo – Department of computer science and information


Rank: 8, 2, 9, 4, 11, 21
Ben – Department of computer science and information
Tsai – Department of computer science and information
Yuan – Department of computer science and information
Chan – Department of computer science and information
Kuo – Department of computer science and information
Lin – Department of computer science and information
ID: 11
ID: 21


17.12 m
12.94 m
8.27 m
6.96 m
6.50 m
5.36 m

CHAPTER-9
RESULTS AND DISCUSSIONS

Results and Discussions

1. Results

- **Real-Time Object Detection:** The YOLO (You Only Look Once) model was successfully implemented for real-time object detection, specifically tuned to detect human figures (runners). The system was able to detect individuals in dynamic video frames with high accuracy, regardless of fast movements or occlusions.

  Performance: The system performed at 30 FPS on average, with high-resolution inputs (1080p videos) being processed efficiently. This was made possible by leveraging GPU acceleration, ensuring that detection and tracking algorithms ran smoothly without significant frame drops.

- **Crossing Detection and Real-Time Ranking:** The system was able to detect when each runner crossed the predefined slant line accurately. The system also maintained real-time rankings based on the runners' crossing times. This allowed for immediate display of race results.

  Ranking Outcome: Real-time ranking was updated dynamically, and the runners' positions were calculated based on when they crossed the line. This dynamic system allowed organizers to track the race's progress instantly and display accurate results as the race unfolded.

- **Dynamic Lane Tagging:** Dynamic lane tagging was activated when a runner entered a specific lane. Tags that followed each runner’s bounding box were displayed on the video, ensuring that lane-specific information, such as participant name or ID, was always visible during the race.

  Tagging Outcome: Tags moved fluidly with runners, updating their position based on the lane coordinates and maintaining alignment with each runner’s movement. This visual feedback enhanced the experience for spectators and race officials, allowing them to easily track runners and their respective lanes.

- **Scalability and Robustness:** The system showed scalability with multiple runners in the frame. It effectively handled situations with dense crowds, where multiple runners were detected in close proximity. The tracking was continuous without any significant identity switching or loss, even when some runners were temporarily occluded by others.

  Handling Edge Cases: The system handled cases such as overlapping runners or fast

movements without misidentifying or misclassifying participants. This robustness is critical in real-world race environments where multiple participants could cross paths frequently.

2. Discussions

- **Performance and Accuracy:** One of the key strengths of the system is its high accuracy in detecting runners and maintaining consistent tracking. The combination of YOLO's efficient object detection with DeepSORT's robust tracking algorithm ensured that runners were identified correctly across multiple frames. The real-time processing speed of the system also made it suitable for live events.

  However, one limitation could arise when dealing with extremely high-density crowds where frequent occlusions occur for long periods. In such cases, the tracking algorithm may experience occasional identity mismatches, although these were generally rare and the system was able to recover quickly.

- **Crossing Detection and Ranking:** The implementation of crossing detection based on the runners' position relative to the slant line worked as intended. The real-time update of rankings allowed for immediate feedback, which is crucial for events where results need to be presented quickly. However, the accuracy of this feature could be affected by video quality or misaligned detection boundaries. It is important that the slant line is well-calibrated to avoid discrepancies in crossing events.

  Future improvements could include more sophisticated line-crossing detection that takes into account more complex race environments, such as runners passing multiple timing gates or crossing at different angles.

- **Dynamic Lane Tagging:** The ability to dynamically activate lane tags when a runner enters a lane improved the clarity of the event and offered live viewers a better understanding of who was in which lane. However, as the system currently uses static lane coordinates, variations in the video feed (such as camera shifts) could slightly affect tag alignment.

  In the future, enhancing the system with automatic calibration based on real-time camera position detection could further improve the accuracy of lane tagging.

- **System Robustness and Edge Case Handling:** The system's ability to function under varying lighting conditions, with high-speed runners, and even when runners overlap, was a notable success. The real-time detection and tracking algorithms demonstrated robustness across various testing scenarios. However, challenges remain in handling extremely crowded races, where constant occlusions can cause tracking difficulties.

  To address this, future iterations of the system could incorporate better models that handle occlusions more effectively, such as using additional sensors or refining tracking algorithms to account for more complex crowd behavior.

- **User Experience:** The interface provided for race organizers was straightforward and allowed for real-time adjustments and monitoring. The clear display of rankings and tag information ensured that users could quickly assess the status of the race. However, there are opportunities to further enhance the user interface, such as adding more interactive features or customizing the display for different race formats.

The **Runner Tracking and Lane Tagging System** successfully demonstrated a robust, real-time tracking solution for competitive events, offering accurate tracking, dynamic lane tagging, and live results for both organizers and audiences. The system proved effective in handling real-time video inputs, providing a seamless experience for users while also ensuring scalable performance in varying conditions.

Although the system performed well in typical scenarios, there are areas for improvement, particularly in handling high-density crowds, ensuring precise lane tag alignment, and addressing occasional tracking errors due to occlusions. Future work could focus on optimizing these features, enhancing the user interface, and incorporating additional sensors or models for improved robustness.

Overall, the project delivered a solid foundation for a comprehensive event tracking system, with potential for further innovation and expansion to support more complex race monitoring and live feedback capabilities.

CHAPTER-10
CONCLUSION

The **Runner Tracking and Lane Tagging System** project has successfully met its core objectives of automating real-time tracking, providing accurate ranking, and enhancing the visual experience for participants and event organizers during competitive races. The system integrates state-of-the-art technologies, including the YOLO model for object detection and DeepSORT for tracking, to create a seamless, efficient, and scalable solution for event management.Through the implementation of dynamic lane tagging and real-time crossing detection, the system provides immediate feedback on race results, improving transparency and eliminating delays in ranking. The live overlay of race data on video feeds further enhances the spectator experience, offering a clear view of each runner's progress and position in the race.

Key achievements of the project include:

- **Real-time object detection and tracking:** Successful implementation of a high-performance tracking system that accurately detects and tracks runners in dynamic environments.
- **Crossing detection and ranking:** Real-time calculation of rankings as runners cross a predefined line, providing instant race results.
- **Dynamic lane tagging:** Activation of lane-specific tags for runners, ensuring clear identification and information throughout the race.
- **Scalability and robustness:** The system efficiently handles varying race conditions, such as high-density crowds and occlusions, with minimal errors.

While the system performed well under typical race conditions, there are opportunities for future improvements, including handling more complex edge cases, enhancing the user interface, and further refining tracking algorithms to improve accuracy in extremely crowded scenarios.

In conclusion, the **Runner Tracking and Lane Tagging System** lays the groundwork for a reliable and innovative solution for live event tracking, ranking, and visualization. It is adaptable to a wide range of competitive events, from local races to large-scale competitions.

Future enhancements will continue to refine and extend the system's capabilities, making it an even more powerful tool for race organizers and participants alike.

REFERENCES

- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788.
- Zivkovic, Z., & van der Heijden, F. (2006). Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, 27(7), 773-780.
- Wang, W., Li, S., & Zha, H. (2018). Real-time 3D tracking of sports players: A 3D reconstruction-based method. *IEEE Transactions on Image Processing*, 27(8), 4112-4126.
- Shukla, S., & Bhardwaj, A. (2020). Automated race tracking using object detection and real-time data visualization. *International Journal of Sports Engineering and Technology*, 12(4), 189-198.
- OpenCV. (n.d.). Open Source Computer Vision Library. Retrieved from https://opencv.org/
- Ultralytics. (2021). YOLOv11: A state-of-the-art object detection model. Retrieved from https://github.com/ultralytics/yolov11
- Matusik, W., & Pfister, H. (2004). 3D tracking and event detection for sports applications. *IEEE Transactions on Visualization and Computer Graphics*, 10(1), 34-45.
- Chang, J., & Lin, M. (2017). Real-time visual tracking and speed estimation in athletic sports. *Journal of Real-Time Image Processing*, 16(3), 431-445.
- Zhang, X., & Wu, Y. (2015). Real-time human tracking using multiple cameras and 3D transformation. *Journal of Computer Vision and Image Understanding*, 129, 1-12.