**Major Project-I**

**on**

# AI-Powered Football Match Monitoring

***Submitted in Partial fulfillment for the award of degree of Bachelor of Technology in Artificial Intelligence & Data Science***

Submitted to

# Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)

Submitted By:

## Aman Kumar Kushwaha (0131AD211005)

## Ashutosh Pandey (0131AD211015)

## Vishnu Prabhakar (0131AD211065)

Under the Guidance of

**Prof. Pushpendra Kumar**

# Department of Artificial Intelligence & Data Science



**Jai Narain College of Technology,Bhopal**

**Approved by AICTE New Delhi & Govt. of M.P.**

**Affiliated to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)**

**Session: 2024-25**

#### JAI NARAIN COLLEGE OF TECHNOLOGY,BHOPAL

**Approved by AICTE New Delhi & Govt. of M.P. & Affiliated to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE**

# CERTIFICATE

This is to certify that the work embodied in this Project entitled as **“AI-Powered Football Match Monitoring ”** being Submitted by **Aman Kumar Kushwaha (0131AD211005) and Ashutosh Pandey (0131AD211015), Vishnu Prabhakar (0131AD211065)** in partial fulfillment of the requirement for the award of **“Bachelor of Technology” in Artificial Intelligence & Data Science** discipline to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.) during the academic year 2024-25 is a record of bonafide piece of work, carried out under my supervision and guidance in the Department of Artificial Intelligence & Data Science, **Jai Narain College of Technology, Bhopal.**

.

### Approved by

Guided by Head of Department

Prof. Pushpendra Kumar Prof. Ravinder Tanwar

Dean, CSE Principal

Dr. Vivek Dubey Dr. Netra Pal Singh

#### JAI NARAIN COLLEGE OF TECHNOLOGY,BHOPAL

**Approved by AICTE New Delhi & Govt. of M.P. & Affiliated to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)**

**DEPARTMENT ARTIFICIAL INTELLIGENCE & DATA SCIENCE**

# CERTIFICATE OF APPROVAL

This Project **“****AI-Powered Football Match Monitoring”** being submitted by **Aman Kumar Kushwaha (0131AD211005), Ashutosh Pandey (0131AD211015) and Vishnu Prabhakar (0131AD211065)** has been examined by me & hereby approve for the partial fulfillment of the requirement for the award of **“Bachelor of Technology in Artificial Intelligence & Data Science”,** for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but the Project only for the purpose for which it has been submitted.

### INTERNAL EXAMINER EXTERNAL EXAMINER

Date: Date:

# CANDIDATE DECLARATION

We hereby declare that the Project work presented in the report entitled as **“AI-Powered Football Match Monitoring”** submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Bachelor of Technology in Artificial Intelligence & Data Science of **Jai Narain College of Technology, Bhopal** is an authentic record of our own work.

We have not submitted the part and partial of this report for the award of any other degree or diploma.

### Aman Kumar Kushwaha(0131AD211005) Ashutosh Pandey (0131AD211015)

### Vishnu Prabhakar (0131AD211065)

Date:

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

### Guided By:

**Prof. Pushpendra Kumar**

# ACKNOWLEDGMENT

We are heartily thankful to the **Jai Narain College of Technology** for providing us all the facilities and infrastructure to take our work to the final stage.

It is the constant supervision, moral support and proper guidance of our respected Principal **Prof. (Dr) Netra Pal Singh** and Dean, CSE **Prof. (Dr) Vivek Dubey,** who motivated throughout the work.

We express a deep sense of gratitude and respect to our learned guide **Prof. Pushpendra Kumar,** Assistant Professor in the Department of Artificial Intelligence & Data Science, during all phases of our work. Without his enthusiasm and encouragement this dissertation would not have been completed. His valuable knowledge and innovative ideas helped us to take the work to the final stage. He has timely suggested actions and procedures to follow for which we are really grateful and thankful to him.

We express our gratitude to **Prof. Ravinder Tanwar** Head of Artificial Intelligence & Data Science Department for providing all the facilities available in the department for his continuous support, advice, and encouragement during this work and also to help to extend our knowledge and proper guidelines.

Constant help, moral and financial support of our loving parents motivated us to complete the work. We express our heartfelt thanks to all our family members for their cooperation.

We really admire the fond support of our class-mates for their cooperation and constant help. It gives immense pleasure to acknowledge the encouragement and support extended by them. Last but not the least we are extremely thankful to all who have directly or indirectly helped us for the completion of the work.

.

### Aman Kumar Kushwaha (0131AD211005) Ashutosh Pandey (0131AD211015)

### Vishnu Prabhakar (0131AD211065)

**Abstract**

This project focuses on the development of an advanced sports analytics system designed to analyse football matches using video footage from eagle-eye cameras. The system incorporates multiple machine learning and computer vision techniques to automate the extraction of player movements, ball tracking, and key game metrics. The primary components of this project include a Speed and Distance Estimator, Team Assigner, and Player-Ball Assigner.

The Speed and Distance Estimator module calculates the distance covered and speed of players during the game. It processes tracking data, measures distances between sequential player positions, and estimates speed over specific frame intervals. This provides a comprehensive analysis of player performance and movement patterns.

The Team Assigner module utilizes colour-based clustering techniques to identify and classify players into teams based on their uniforms. By analyzing colour data from players’ bounding boxes and applying K-Means clustering, the module effectively distinguishes between teams and assigns players accordingly.

The Player-Ball Assigner module assigns ball possession by determining which player is closest to the ball at any given frame. Using a distance threshold and calculating the proximity between the center of the ball’s bounding box and players’ positions, this module helps track ball possession throughout the match.

Together, these modules provide detailed insights into player movements, team dynamics, and game events, enhancing the capabilities for automated match analysis. The project paves the way for further applications in sports analytics, such as tactical assessments, performance monitoring, and automated highlight generation.

By integrating advanced algorithms and leveraging Python libraries for data analysis and computer vision, this system aims to support coaches, analysts, and fans in better understanding game mechanics through robust and scalable analysis tools.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Table of Contents** | **Page No.** |
| **1** | **Introduction** | 8 |
|  | * 1. 1.1 Objective   2. 1.2 Problem Identification   3. 1.3 Proposed Solution | 8  8  8 |
| **2** | **Software Requirement Specification**   * 1. 2.1 Purpose   2.2 Scope   * 1. 2.3 Feasibility Study      1. 2.3.1 Technical Feasibility   2.3.2 Economic Feasibility | 10 |
|  | 10 |
|  | 10 |
|  | 10 |
|  | 10 |
|  | 10 |
|  |  |
| **3** | **Requirements** | 11  11  11  11 |
|  | 3.1 Hardware Requirement |
|  | 3.2 Software Requirement |
|  |  |
|  | 3.4 Functional Requirements |
| **4** | **System Documentation** | 12  12  13 |
|  |  |
|  |  |
|  | 4.3 System Flow Chart |
|  | 4.4 System Level Class Diagram |
|  |  |
|  |  |
| **7** | **Limitations** | 14 |
| **8** | **Future Enhancement** | 15 |
| **9** | **Conclusion** | 16 |
| **12** | **Appendix – I Source Code** | 17 |

### Introduction

In modern football, understanding the nuances of player performance, team dynamics, and match strategies is crucial for success. Traditional methods of match analysis rely heavily on manual observations and basic statistics, which often lack depth and precision. To address this challenge, advancements in computer vision and machine learning have opened the door to automated, data-driven approaches for match analysis. This project harnesses these technologies to create a comprehensive system capable of tracking player and ball movements, estimating key performance metrics, and visualizing data directly on match footage. By providing detailed insights, this system aims to enhance decision-making for coaches, analysts, and players alike.

### Objective

The primary objective of this project is to develop a robust, automated analysis system that processes video footage of football matches to extract and present critical information. The goals include:

1. Accurately tracking players, referees, and the ball throughout the match.
2. Estimating player speed and distance covered using reliable positional data.
3. Classifying players into their respective teams based on jersey colour.
4. Determining ball possession by measuring player proximity to the ball.
5. Providing clear, annotated visualizations of key performance metrics for enhanced analysis.

### 1.2 Problem identification

1. **Labor-intensive Process**: Manual tracking of players and ball movements is time-consuming and requires significant effort from analysts.
2. **Subjective Biases**: Human analysis is prone to subjective interpretation, leading to inconsistencies in performance evaluations.
3. **Real-time Analysis**: Providing immediate insights during live matches is difficult with traditional methods.
4. **Data Accuracy**: Ensuring precise measurements of distances and speeds is challenging without an automated system.
5. **Team Classification**: Correctly identifying players and distinguishing between teams based on visual cues is complex and error-prone without reliable algorithms.

### 1.3 Proposed Solution

To address the intricate challenges of sports video analysis, we propose a comprehensive solution that leverages advanced computer vision and machine learning techniques.

**1. Video Preprocessing**

* **Frame Extraction:** Extract frames from the input video at a consistent frame rate to ensure smooth analysis.
* **Image Quality Enhancement:** Apply image processing techniques like noise reduction, contrast enhancement, and colour normalization to improve the quality of frames and facilitate accurate object detection and tracking.

1. **Object Detection and Tracking**

* **Robust Object Detection:** Employ state-of-the-art object detection models, such as YOLOv8 or DETR, to reliably detect players, the ball, and other relevant objects in each frame.
* **Advanced Tracking Algorithms:** Utilize sophisticated tracking algorithms like DeepSORT or ByteTrack to maintain object identities across frames, handling occlusions, re-identifications, and complex motion patterns.

**3. Camera Motion Estimation and Compensation**

* **Feature Tracking:** Employ robust feature tracking techniques, such as KLT or Lucas-Kanade, to identify corresponding points between consecutive frames.

**4. Team Assignment and Player Identification**

* **Colour-Based Clustering:** Utilize colour-based clustering techniques (e.g., K-Means) to group players based on their jersey colours.
* **Temporal Consistency:** Enforce temporal consistency in team assignments to handle ambiguous cases and ensure accurate tracking across frames.
* **Player Identification:** Explore techniques like face recognition or unique player attributes (e.g., tattoos, equipment) to identify individual players.

**5. Performance Metric Calculation**

* **Speed and Distance:** Accurately calculate player speeds and distances travelled using frame-to-frame displacement and time intervals.
* **Passing and Shooting Accuracy:** Analyse player actions to estimate passing and shooting accuracy, considering factors like ball velocity, angle, and target.

By combining these techniques and addressing the challenges, we can develop a robust and accurate sports video analysis system that provides valuable insights for coaches, analysts, and fans.

### 2. Software Requirement Specification

### 2.1 Purpose

This project aims to develop a robust sports video analysis system that automates the process of extracting valuable insights from video footage. By employing advanced computer vision and machine learning techniques, the system will accurately track players and the ball, estimate camera movement, and calculate performance metrics like speed, distance, and possession time. This will enable coaches, analysts, and fans to gain deeper insights into player performance, team tactics, and game strategies.

**2.2 Scope**

* **Track Players and Ball:** Accurately track the positions of players and the ball in each frame.
* **Estimate Camera Movement:** Determine camera movement (pan, tilt, zoom) to compensate for perspective changes.
* **Calculate Performance Metrics:** Compute player speed, distance traveled, and other relevant metrics.
* **Analyse Team Tactics:** Identify team formations, passing patterns, and defensive strategies.
* **Generate Visualizations:** Create interactive visualizations to aid in analysis and decision-making.

**2.3 Feasibility Study**

**2.3.1 Technical Feasibility**

**Object Detection and Tracking:**

* **Deep Learning Models:** State-of-the-art deep learning models like YOLOv5, DETR, and DeepSORT can reliably detect and track objects in complex sports scenarios.
* **Real-time Processing:** With advancements in hardware (GPUs) and optimized algorithms, real-time processing of high-resolution video is feasible.

**Camera Motion Estimation:**

* **Optical Flow:** Techniques like Lucas-Kanade optical flow can estimate motion between frames, providing insights into camera movement.

**2.3.2 Economic Feasibility**

* **Cost of Development:** The cost of development will depend on factors such as team size, expertise, and the complexity of the system.
* **Hardware and Software Costs:** The cost of hardware (e.g., GPUs, high-performance computers) and software licenses (e.g., deep learning frameworks) will need to be considered.
* **Potential Revenue:** The system can be commercialized through licensing, subscription models, or consulting services.

### Requirements

A nutshell, to create a multilingual information system for railway passengers and customers, you'd need technologies for: -

### Hardware Requirements

* + - Computer System: Intel i5 or AMD Ryzen 5
    - Graphical Processing Unit: NVIDIA GTX 1650
    - Internet: 5mbps bandwidth
    - Power Supply: 120 WATTs

### Software Requirements

* GPU-accelerated Computing Libraries (CUDA, cuDNN, etc.)
* Visualization Libraries (Matplotlib, Seaborn, etc.)
* Version Control Systems (Git, Mercurial, SVN, etc.)
* Integrated Development Environments (PyCharm, Visual Studio Code, etc.)
* Containerization Tools (Docker, Kubernetes, etc.)
* Documentation Generation Tools (Sphinx, Doxygen, etc.)
* Operating Systems (Linux, Windows, macOS, etc.)

### Functional Requirements

* **Video Input and Processing:** Support various video formats, extract frames, and preprocess for enhanced image quality.
* **Object Detection and Tracking:** Accurately detect and track players, the ball, and other relevant objects, handling occlusions and re-identifications.
* **Camera Motion Estimation:** Estimate camera movement parameters (pan, tilt, zoom) and compensate for motion to obtain accurate object positions and trajectories.
* **Feature Extraction and Analysis:** Extract relevant features like player speed, distance, ball possession, and analyse team tactics and player interactions.
* **Performance Metric Calculation:** Calculate metrics like player speed, acceleration, distance covered, pass accuracy, shot accuracy, and time of possession.
* **Visualization and Reporting:** Generate annotated videos, statistical reports, heat maps, line charts, and provide an interactive user interface.
* **Player Identification:** Identify individual players based on appearance or other unique features.
* **Action Recognition:** Recognize specific player actions like passing, shooting, dribbling, and tackling.
* **Tactical Analysis:** Analyse team formations, passing patterns, and defensive strategies.
* **Integration with External Systems:** Integrate with other sports analysis tools, databases, and coaching platforms.

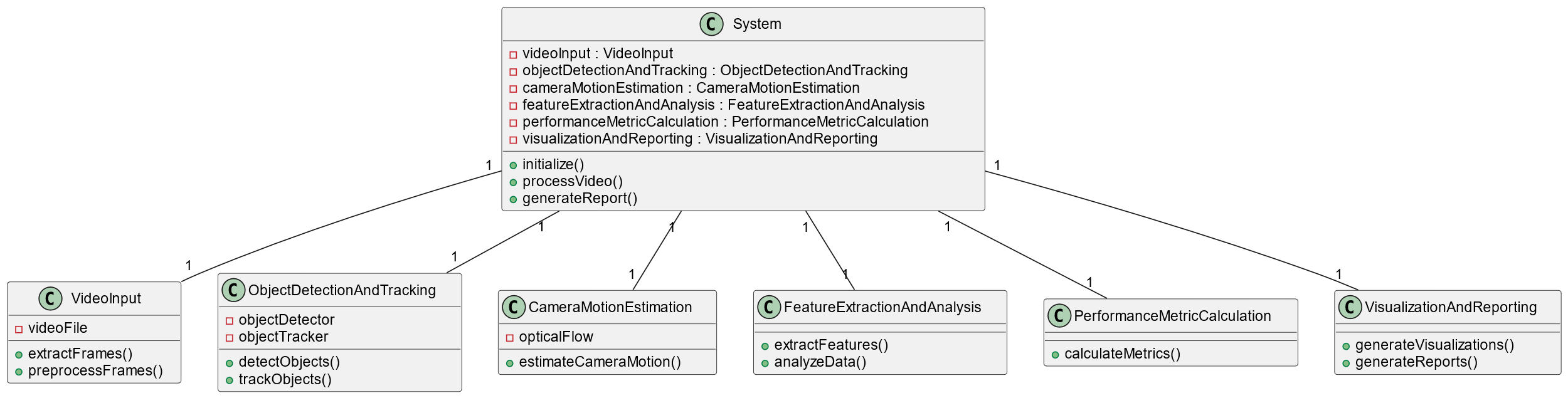
### System Documentation

The sports video analysis system will provide a robust and user-friendly solution for extracting valuable insights from sports footage. By leveraging advanced computer vision and machine learning techniques, the system will enable coaches, analysts, and fans to analyse player performance, team tactics, and game strategies in detail. This will ultimately lead to improved decision-making and a deeper understanding of the game.

### 4.1 System Flow Chart



**4.2 System Level Class Diagram**



### 5. Limitations

### While the sports video analysis system offers significant potential, it is important to acknowledge its limitations:

### Video Quality: The accuracy of the analysis depends on the quality of the input video. Poor video quality, low resolution, or excessive noise can hinder the performance of object detection and tracking algorithms.

### Occlusions: When players or objects are occluded, tracking can become challenging, leading to potential errors in analysis.

### Complex Camera Movements: Extreme camera movements, such as rapid zooming or panning, can impact the accuracy of camera motion estimation and object tracking.

### Lighting Conditions: Varying lighting conditions, especially in indoor or low-light environments, can affect the performance of object detection and tracking algorithms.

### Computational Cost: Real-time analysis of high-resolution videos can be computationally intensive, requiring powerful hardware and efficient algorithms.

### Data Annotation: The development of accurate machine learning models requires large amounts of annotated data, which can be time-consuming and labour-intensive.

### Contextual Understanding: While the system can analyse player movements and ball trajectories, it may struggle to fully understand the tactical context and strategic nuances of the game.

### 6. Future enhancements

While the current system provides a solid foundation for sports video analysis, there are several potential enhancements to explore:

**1. Advanced Machine Learning Techniques:**

* **Deep Learning:** Leverage deep learning models for more accurate object detection, tracking, and pose estimation.
* **Reinforcement Learning:** Train agents to optimize analysis strategies and adapt to different video scenarios.

.

**2. Real-time Processing:**

* Optimize algorithms and hardware to enable real-time analysis of live sports broadcasts.
* Explore cloud-based solutions for scalable and efficient processing.

**3. Multi-Camera Support:**

* Develop algorithms to handle multiple camera views and synchronize data from different perspectives.
* Use camera calibration techniques to accurately align and fuse information from multiple cameras.

**4. User Experience Improvements:**

* Provide a user-friendly interface with intuitive controls and clear visualizations.
* Offer customizable settings and preferences to tailor the system to individual needs.
* Implement robust error handling and user feedback mechanisms.

### 7. Conclusion

The development of a robust sports video analysis system offers significant potential for enhancing the understanding and analysis of sports performance. By leveraging advanced computer vision and machine learning techniques, this system can accurately track players and the ball, estimate camera motion, and calculate performance metrics.

The system's ability to provide detailed insights into player performance, team tactics, and game strategies can revolutionize the way coaches, analysts, and fans approach sports analysis. It can help identify areas for improvement, optimize training regimens, and gain a competitive edge.

While significant progress has been made, there are still opportunities for further development, such as improving the accuracy of object tracking in challenging conditions, enhancing the understanding of player intentions, and integrating with other sports technologies.

By addressing these areas and continuing to innovate, the sports video analysis system can become an indispensable tool for the sports industry, empowering teams and fans alike.

### Source Code

### Main.py

import pickle

import cv2

import numpy as np

from camera\_movement\_estimator import CameraMovementEstimator

from player\_ball\_assigner import PlayerBallAssiginer

from speed\_and\_distance\_estimator import SpeedAndDistanceEstimator

from team\_assignier import TeamAssiginer

from trackers import Tracker

from utils import read\_video, save\_video

from view\_transformer import ViewTransformer

def main():

    # Read video

    print("reading Video")

    video\_frames = read\_video("input\_videos/08fd33\_4.mp4")

    # Initialize the tracker

    print("tracking players")

    tracker = Tracker(model\_path="models/best.pt")

    tracks = tracker.get\_object\_tracks(

        frames=video\_frames, read\_from\_stub=True, stub\_path="stubs/track\_stubs.pkl"

    )

    # Get objects position

    tracker.add\_position\_to\_tracks(tracks)

    # camara movement estimator

    print("estimating camera movement")

    cm\_estimator = CameraMovementEstimator(frame=video\_frames[0])

    camara\_movement\_per\_frame = cm\_estimator.get\_camera\_movement(

        video\_frames, True, "stubs/camera\_movement.pkl"

    )

    cm\_estimator.adjust\_positions\_to\_tracks(tracks, camara\_movement\_per\_frame)

    # View transformer

    print("transforming view")

    vt = ViewTransformer()

    vt.add\_transformed\_position\_to\_tracks(tracks)

    # Interpolate ball positions

    tracks["ball"] = tracker.interpolate\_ball\_positions(tracks["ball"])

    # Speed and distance estimator

    print("estimating speed and distance")

    speed\_distance\_estimator = SpeedAndDistanceEstimator()

    speed\_distance\_estimator.add\_speed\_and\_distance\_to\_tracks(tracks)

    # Initialize the team assignier

    print("assigning teams")

    team\_assignier = TeamAssiginer()

    team\_assignier.assign\_team\_color(

        frame=video\_frames[0],

        player\_detections=tracks["players"][0],

    )

    for frame\_num, player\_track in enumerate(tracks["players"]):

        for player\_id, player\_bbox in player\_track.items():

            bbox = player\_bbox["bbox"]

            team\_id = team\_assignier.get\_player\_team(

                video\_frames[frame\_num], bbox, player\_id

            )

            tracks["players"][frame\_num][player\_id]["team"] = team\_id

            tracks["players"][frame\_num][player\_id]["color"] = (

                team\_assignier.team\_colors[team\_id]

            )

    # Assign ball Aquisition

    print("assigning ball aquisition")

    player\_assigner = PlayerBallAssiginer()

    team\_ball\_control = []

    for frame\_num, player\_track in enumerate(tracks["players"]):

        ball\_box = tracks["ball"][frame\_num][1]["bbox"]

        assigned\_player = player\_assigner.assign\_ball\_to\_player(player\_track, ball\_box)

        if assigned\_player != -1:

            tracks["players"][frame\_num][assigned\_player]["has\_ball"] = True

            team\_ball\_control.append(

                tracks["players"][frame\_num][assigned\_player]["team"]

            )

        else:

            team\_ball\_control.append(team\_ball\_control[-1])

    team\_ball\_control = np.array(team\_ball\_control)

    # Draw annotations

    print("drawing annotations")

    output\_video\_frames = tracker.draw\_annotations(

        video\_frames, team\_ball\_control, tracks, camara\_movement\_per\_frame

    )

    # Draw speed and distance

    print("drawing speed and distance")

    output\_video\_frames = speed\_distance\_estimator.draw\_speed\_and\_distance(

        output\_video\_frames, tracks

    )

    with open("stubs/complete\_tracks.pkl", "wb") as f:

        pickle.dump(tracks, f)

    # Save video

    save\_video(output\_video\_frames, "output\_videos/output\_video\_final.mp4")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

### tracker.py

import os

import pickle

import cv2

import numpy as np

import pandas as pd

import supervision as sv

from ultralytics import YOLO

from utils import get\_bbox\_width, get\_center\_of\_bbox, get\_foot\_position

class Tracker:

    def \_\_init\_\_(self, model\_path):

        self.model = YOLO(model\_path)

        self.tracker = sv.ByteTrack()

    def add\_position\_to\_tracks(self, tracks):

        for objects, object\_track in tracks.items():

            for frame\_num, track in enumerate(object\_track):

                for track\_id, track\_info in track.items():

                    bbox = track\_info["bbox"]

                    if objects == "ball":

                        position = get\_center\_of\_bbox(bbox)

                    else:

                        position = get\_foot\_position(bbox)

                    tracks[objects][frame\_num][track\_id]["position"] = position

    def detect\_frames(self, frames):

        batch\_size = 20

        detections = []

        for i in range(0, len(frames), batch\_size):

            batch\_detections = self.model.predict(frames[i : i + batch\_size], conf=0.1)

            detections += batch\_detections

        return detections  *# List[dict] {'boxes': [[],[],[]], 'conf': [], 'cls\_names': [0,0,0,2,2,1,1,1,1]}*

    def get\_object\_tracks(self, frames, read\_from\_stub=False, stub\_path=None):

        if read\_from\_stub and stub\_path is not None and os.path.exists(stub\_path):

            with open(stub\_path, "rb") as f:

                return pickle.load(f)

        detections = self.detect\_frames(frames)

        tracks = {

            "players": [],

            "referees": [],

            "ball": [],

        }

        for frame\_num, detection in enumerate(detections):

            cls\_name = detection.names  *# {0: 'person', 1: 'car', ....}*

            cls\_name\_inv = {

                v: k for k, v in cls\_name.items()

            }  *# {'person': 0, 'car': 1, ....}*

*# Convert the detections from ultralytics to Supervision format*

            supervision\_detections = sv.Detections.from\_ultralytics(detection)

*# Convert Goalkeeper to player object*

            for object\_ind, class\_id in enumerate(supervision\_detections.class\_id):

                if cls\_name[class\_id] == "goalkeeper":

                    supervision\_detections.class\_id[object\_ind] = cls\_name\_inv["player"]

*# Tracker objects*

            detection\_with\_tracker = self.tracker.update\_with\_detections(

                supervision\_detections

            )

            tracks["players"].append({})

            tracks["referees"].append({})

            tracks["ball"].append({})

            for frame\_detection in detection\_with\_tracker:

                bbox = frame\_detection[0].tolist()

                cls\_id = frame\_detection[3]

                track\_id = frame\_detection[4]

                if cls\_id == cls\_name\_inv["player"]:

                    tracks["players"][frame\_num][track\_id] = {"bbox": bbox}

                if cls\_id == cls\_name\_inv["referee"]:

                    tracks["referees"][frame\_num][track\_id] = {"bbox": bbox}

            for frame\_detection in supervision\_detections:

                bbox = frame\_detection[0].tolist()

                cls\_id = frame\_detection[3]

                if cls\_id == cls\_name\_inv["ball"]:

                    tracks["ball"][frame\_num][1] = {"bbox": bbox}

            if stub\_path is not None:

                with open(stub\_path, "wb") as f:

                    pickle.dump(tracks, f)

        return tracks

    def draw\_triangel(self, frame, bbox, color):

        y = int(bbox[1])

        x, \_ = get\_center\_of\_bbox(bbox=bbox)

        traiangel\_points = np.array([[x, y], [x - 10, y - 20], [x + 10, y - 20]])

        cv2.drawContours(frame, [traiangel\_points], 0, color, cv2.FILLED)

        cv2.drawContours(frame, [traiangel\_points], 0, (0, 0, 0), 2)

        return frame

    def draw\_ellipse(self, frame, bbox, color, track\_id=None):

        y2 = int(bbox[3])  *# want to place ellipse at the bottom*

        x\_center, \_ = get\_center\_of\_bbox(bbox)

        width = get\_bbox\_width(bbox)

*# Draw ellipse*

        cv2.ellipse(

            frame,

            center=(x\_center, y2),

            axes=(int(width), int(0.35 \* width)),

            angle=0.0,

            startAngle=-45,

            endAngle=235,

            color=color,

            thickness=2,

            lineType=cv2.LINE\_4,

        )

*# Draw Rectangel*

        rectangle\_width = 40

        rectangle\_height = 20

        rectangle\_x1 = x\_center - rectangle\_width // 2

        rectangle\_x2 = x\_center + rectangle\_width // 2

        rectangle\_y1 = (y2 - rectangle\_height // 2) + 15

        rectangle\_y2 = (y2 + rectangle\_height // 2) + 15

        if track\_id is not None:

            cv2.rectangle(

                frame,

                pt1=(rectangle\_x1, rectangle\_y1),

                pt2=(rectangle\_x2, rectangle\_y2),

                color=color,

                thickness=cv2.FILLED,

            )

            text\_x1 = rectangle\_x1 + 12

            if track\_id > 99:

                text\_x1 -= 10

            cv2.putText(

                frame,

                str(track\_id),

                (text\_x1, rectangle\_y1 + 15),

                cv2.FONT\_HERSHEY\_SIMPLEX,

                0.6,

                (0, 0, 0),  *# Black*

                2,

            )

        return frame

    def draw\_team\_ball\_control(self, frame, frame\_num, team\_ball\_control):

        overlay = frame.copy()

        cv2.rectangle(overlay, (1350, 850), (1900, 970), (255, 255, 255), -1)

        alpha = 0.4

        cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0, frame)

        team\_ball\_control\_till\_frame = team\_ball\_control[: frame\_num + 1]

*# Number of time each team have the ball*

        team\_1\_num\_frames = team\_ball\_control\_till\_frame[

            team\_ball\_control\_till\_frame == 1

        ].shape[0]

        team\_2\_num\_frames = team\_ball\_control\_till\_frame[

            team\_ball\_control\_till\_frame == 2

        ].shape[0]

        team\_1 = team\_1\_num\_frames / (team\_1\_num\_frames + team\_2\_num\_frames)

        team\_2 = team\_2\_num\_frames / (team\_1\_num\_frames + team\_2\_num\_frames)

        cv2.putText(

            frame,

            f"Team 1 Ball Control: {team\_1\*100:.2f}%",

            (1400, 900),

            cv2.FONT\_HERSHEY\_SIMPLEX,

            1,

            (0, 0, 0),  *# color*

            3,  *# thickness*

        )

        cv2.putText(

            frame,

            f"Team  Ball Control: {team\_2\*100:.2f}%",

            (1400, 950),

            cv2.FONT\_HERSHEY\_SIMPLEX,

            1,

            (0, 0, 0),  *# color*

            3,  *# thickness*

        )

        return frame

    def draw\_camera\_movement(self, frame, frame\_num, camera\_movement):

        overlay = frame.copy()

        cv2.rectangle(overlay, (0, 0), (500, 100), (255, 255, 255), -1)

        alpha = 0.6

        cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0, frame)

        cv2.putText(

            frame,

            f"Camara Movement X: {camera\_movement[frame\_num][0]:.2f}",

            (10, 30),

            cv2.FONT\_HERSHEY\_SIMPLEX,

            1,

            (0, 0, 0),  *# color*

            3,  *# thickness*

        )

        cv2.putText(

            frame,

            f"Camara Movement Y: {camera\_movement[frame\_num][1]:.2f}",

            (10, 60),

            cv2.FONT\_HERSHEY\_SIMPLEX,

            1,

            (0, 0, 0),  *# color*

            3,  *# thickness*

        )

        return frame

    def draw\_annotations(

        self, video\_frames, team\_ball\_control, tracks, camera\_movement

    ):

        output\_video\_frames = []

        for frame\_num, frame in enumerate(video\_frames):

            frame = frame.copy()

            player\_dict = tracks["players"][frame\_num]

            referee\_dict = tracks["referees"][frame\_num]

            ball\_dict = tracks["ball"][frame\_num]

*# Draw players*

            for track\_id, player in player\_dict.items():

                color = player.get("color", (0, 0, 225))

                frame = self.draw\_ellipse(frame, player["bbox"], color, track\_id)

                if player.get("has\_ball", False):

                    self.draw\_triangel(frame, player["bbox"], (0, 0, 255))

*# Draw Referees*

            for \_, referee in referee\_dict.items():

                frame = self.draw\_ellipse(frame, referee["bbox"], (0, 255, 255))

*# Draw ball*

            for \_, ball in ball\_dict.items():

                frame = self.draw\_triangel(frame, ball["bbox"], (0, 255, 0))  *# b.g.r*

*# Draw team ball control*

            frame = self.draw\_team\_ball\_control(frame, frame\_num, team\_ball\_control)

*# Draw camera movement*

            frame = self.draw\_camera\_movement(frame, frame\_num, camera\_movement)

            output\_video\_frames.append(frame)

        return output\_video\_frames

    def interpolate\_ball\_positions(self, ball\_positionss):

        ball\_positionss = [

            frame.get(1, {}).get("bbox", []) for frame in ball\_positionss

### camera\_movement.py

import os

import pickle

import cv2

import numpy as np

from utils import measure\_distance, measure\_xy\_distance

class CameraMovementEstimator:

    def \_\_init\_\_(self, frame) -> None:

        """

        Initialize the CameraMovementEstimator class.

        Args:

            frame: The first frame of the video.

        """

*# Set the minimum distance threshold for considering a camera movement.*

        self.minimum\_distance = 5

*# Convert the first frame to grayscale.*

        first\_frame\_greyscale = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

*# Create a mask for the features to track.*

*# The mask is a binary image where 1 indicates the feature is trackable.*

*# In this case, we are setting the first 20 columns and the pixel at (900, 1050) to 1.*

        mask\_features = np.zeros\_like(first\_frame\_greyscale)

        mask\_features[:, 0:20] = 1

        mask\_features[:, 900:1050] = 1

*# Set the parameters for the goodFeaturesToTrack function.*

*# These parameters specify the maximum number of corners, quality level, minimum distance,*

*# block size, and the mask for the features to track.*

        self.features = dict(

            maxCorners=100,  *# Maximum number of corners to detect.*

            qualityLevel=0.3,  *# Minimum quality level for corner detection.*

            minDistance=3,  *# Minimum distance between corners.*

            blockSize=7,  *# Size of the block for corner detection.*

            mask=mask\_features,  *# Mask for the features to track.*

        )

*# Set the parameters for the calcOpticalFlowPyrLK function.*

*# These parameters specify the window size, maximum level, termination criteria, and the mask for the features to track.*

        self.lk\_params = dict(

            winSize=(15, 15),  *# Window size for Lucas-Kanade optical flow.*

            maxLevel=2,  *# Maximum level of pyramid for Lucas-Kanade optical flow.*

            criteria=(

                cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT,

                10,

                0.03,

            ),  *# Termination criteria for Lucas-Kanade optical flow.*

        )

    def adjust\_positions\_to\_tracks(self, tracks, camera\_movement\_per\_frame):

        for objects, object\_track in tracks.items():

            for frame\_num, track in enumerate(object\_track):

                for track\_id, track\_info in track.items():

                    position = track\_info["position"]

                    camera\_movement = camera\_movement\_per\_frame[frame\_num]

                    adjusted\_position = [

                        position[0] - camera\_movement[0],

                        position[1] - camera\_movement[1],

                    ]

                    tracks[objects][frame\_num][track\_id][

                        "adjusted\_position"

                    ] = adjusted\_position

    def get\_camera\_movement(self, frames, read\_from\_stub=False, stub\_path=None):

*# If we should read from a stub file and the stub path is provided and the file exists,*

*# then read the camera movement from the stub file and return it.*

        if read\_from\_stub and stub\_path is not None and os.path.exists(stub\_path):

            with open(stub\_path, "rb") as f:

*# Load the camera movement from the stub file.*

                camera\_movement = pickle.load(f)

                return camera\_movement

*# Initialize an empty list to hold the camera movement for each frame.*

        camera\_movement = [[0, 0]] \* len(frames)

*# Convert the first frame to grayscale and find the features to track.*

        old\_grey = cv2.cvtColor(frames[0], cv2.COLOR\_BGR2GRAY)

        old\_features = cv2.goodFeaturesToTrack(image=old\_grey, \*\*self.features)

*# Iterate over each frame after the first frame.*

        for frame\_num in range(1, len(frames)):

*# Convert the current frame to grayscale.*

            frame\_gray = cv2.cvtColor(frames[frame\_num], cv2.COLOR\_BGR2GRAY)

*# Use the Lucas-Kanade algorithm to track the features between the current and previous frames.*

            new\_features, \_, \_ = cv2.calcOpticalFlowPyrLK(

                prevImg=old\_grey,

                nextImg=frame\_gray,

                prevPts=old\_features,

                nextPts=None,

                \*\*self.lk\_params,

            )

*# Initialize variables to hold the maximum distance and the camera movement for the frame.*

            max\_distance = 0

            camera\_movement\_x, camera\_movement\_y = 0, 0

*# Iterate over each new feature and old feature pair.*

            for new, old in zip(new\_features, old\_features):

*# Convert the new and old features to a single dimension array.*

                new\_features\_points = new.ravel()

                old\_features\_points = old.ravel()

*# Calculate the distance between the new and old features.*

                distance = measure\_distance(new\_features\_points, old\_features\_points)

*# If the distance is greater than the maximum distance, update the maximum distance and the camera movement for the frame.*

                if distance > max\_distance:

                    max\_distance = distance

                    camera\_movement\_x, camera\_movement\_y = measure\_xy\_distance(

                        old\_features\_points, new\_features\_points

                    )

*# If the maximum distance is greater than the minimum distance, update the camera movement for the frame.*

            if max\_distance > self.minimum\_distance:

                camera\_movement[frame\_num] = [camera\_movement\_x, camera\_movement\_y]

*# Update the old features for the next frame.*

                old\_features = cv2.goodFeaturesToTrack(frame\_gray, \*\*self.features)

*# Update the old grayscale frame for the next frame.*

            old\_grey = frame\_gray.copy()

*# If a stub path is provided, write the camera movement to the stub file.*

        if stub\_path is not None:

            with open(stub\_path, "wb") as f:

*# Write the camera movement to the stub file.*

                pickle.dump(camera\_movement, f)

*# Return the camera movement for each frame.*

        return camera\_movement

        df\_ball\_positions = pd.DataFrame(

            ball\_positionss, columns=["x1", "y1", "x2", "y2"]

        )

        df\_ball\_positions = df\_ball\_positions.interpolate()

        df\_ball\_positions = df\_ball\_positions.bfill()

        return [{1: {"bbox": x}} for x in df\_ball\_positions.to\_numpy().tolist()]