Step1 : Player Detection and Tracking

1. **Training our YOLO model**

From ultranalytics import YOLO, models = YOLO(……), result = models.predict(input video….)

2. Now to **read the video** we use openCV library import cv2

function read_video(…)

cap=cv2.VideoCapture(…video_path…) reads each frame of the video

frames = []

Loop through all the frames and append it in the frame list….. ret,frame = cap.read()

Finally returns the frames list…..

function **save_video**(…output video frames , output video path…)

Specify the format = cv2.fourcc(*xvid) out = cv2.videowrite(…format,(width, height),fps….)

Loop in output video frames for out in output video frames : out.write(frame)

out.release()

3. **Object detection and tracking** for detection we use YOLO and for tracking we use ByteTracker

```
# model = self.YOLO(best fit) ——> from ultranalytics import YOLO
#
Object detection
detect_frames(frames):
        batch size = 20
        detections = []
        for i in range(0,len(frames),batch size):
                train using YOLO
                detection_batch = self.model.predict(frames[i:batch_size+l],confidendece=0.1)
                detections += detection_batch
        return detections
Object tracking
get_object_track(frames):
        detections = detect_frames(frames)
        for frame_num,detection in enumerate(detections):
                cls_name = detection.name —->{0:person,1:ball……..}
                cls_inv = {v:k for k,v in cls_name.items()}
                supervision_detection = sv.Detection.from_ultralytics(detection) ——> import
supervision as sv
                print(supervison_detection)
## batch 1 : array([x1,y1,x2,y2],[..,…,…,…]……)
                        conf = ……,……,…….
                        class = 0,1,….
```

4. **Convert goalkeeper as player**

```
Inside get_object_track(frames):
                detections = detect_frames(frames)
                for frame_num,detection in enumerate(detections):
                        cls_name = detection.name —->{0:person,1:ball……..}
                        cls_inv = {v:k for k,v in cls_name.items()}
                        supervision_detection = sv.Detection.from_ultralytics(detection)
                        for object_ind,class_id in enumerate(supervision_detection.class_id):
                                if cls_name[cls_id]==‘goalkeeper’:
                                        supervision_detection.class_id[object_ind]=cls_inv[‘player’]
```

5. tracker = sv.ByteTracker() **creating tracker object and create a dictionary for efficient tracking**

No track_id for balls

Tracks = {“players”:[{..## frame1…},{..#frame2…},{0:[…bbox…],12:[…bbox…],…},{track_id1: [bbox…],track_id2:…},……],

“referees”:[….same….],

“ball”:[{[…bbox…]},{[…bbox…]}]}

```
Inside get_object_track(frames):
                detections = detect_frames(frames)
                tracks = {
                “players”:[ ],
```

```
                “referees”:[ ],
                “ball”:[ ]
                }
                for frame_num,detection in enumerate(detections):
                        cls_name = detection.name —->{0:person,1:ball……..}
                        cls_inv = {v:k for k,v in cls_name.items()}
                        supervision_detection = sv.Detection.from_ultralytics(detection)

                        for object_ind,class_id in enumerate(supervision_detection.class_id):
                                if cls_name[cls_id]==‘goalkeeper’:
                                        supervision_detection.class_id[object_ind]=cls_inv[‘player’]
                        track objects  detection_with_track=
self.tracker.update_with_detection(supervision_detection)
                        tracks[“players”].apend({ })
                        tracks[“referees”].append({ })
                        tracks[“ball”].append({ })
detection_with_track = {bbox:……,……,class_id:……,track_id:…….}
# here now we are creating the tracks
                        for frame_detection in detection_with_track:
                                bbox = frame_detection[0].toList( )
                                cls_id = frame_detection[3]
                                track_id = frame_detection[4]
                                if cls_id = cls_inv[“players”]: tracks[“players”][frame_num]
[track_id]={“bbox”:bbox}
                                if cls_id = cls_inv[“referees”]: similarly for referees
Since ball occurs only once in one frame
                        for frame_detection in detection_supervision:
                                bbox = frame_detection[0].toList( )
                                cls_id = frame_detection[3]
                                if cls_id = cls_inv[“ball”]: tracks[“ball”][frame_num][1]={“bbox”:bbox}
```

6. **Saving the file** so that we do not need to run the code again again from the next time it will read from the saved file using pickle module and os module.
get_object_track(frames,read_from_stub=False,stub_path=None)
7. **Drawing annotations**:

```
draw_ellipse(frame,bbox,color,track_id):
        x_centre=centre_of_bbox(bbox)
        cv2.ellips(centre=(x_center,y2),axis=(major axis, minor axis),start_angle=,end_angle=)
        if track_id is not None:
                cv2.rectangle(……)
                cv2.put_text(…..)
draw triangle(frame,bbox,color):
        cv2.drawContours(…….)—>triangle points
Inside utils module
# centre of the bbox
X1,x2,y1,y2 = bbox int((x1+x2)/2),int((y1+y2)/2)
# bbox width bbox[1]-bbox[0]
draw_annotation(video_frame,tracks):
        output_video_frames=[ ]
        for frame_num,frame in enumerate(video_frame):
                player_dict = tracks[“players”][frame_num]
                similarly create separate dict for referees and ball
                for track_id,player in player_dict.items( ):
                        frame = self.draw_ellipse(frame,player[bbox],(0,0,255)—>color,track_id)
                for track_id,ref in referee_dict.items( ):
                        frame = self.draw_ellipse(frame,ref[bbox],(0,0,255)—>color,track_id)
                for track_id,ref in referee_dict.items( ):
                        frame = self.draw_triangle(frame,ref[bbox],(0,0,255)—>color,track_id)
                output_videoz-frame.append(frame)
        return output_video_frames.
```

Step2: Team Assignment Using KMean Clustering

```
Class TeamAssigner:
        def __init__( ):
                self.team_color = {}
                self.player_team_dict = {}
        def get_clustering_model(self,image):
                image_2d = image.reshape(-1,3)
                kmean = KMeans(n_clusters=2,kmeans++,n_init=1)
                kmean.fit(image_2d)
                return kmeans
        def get_player_color(self, frame,bbox):
                image = frame[bbox[1]:bbox[3],bbox[0]:bbox[2]]
                top_half_of_image = image[0 : image.shape[0]/2 : ]
                kmeans = self.get_clustering_model(top_half_of_image)
                labels = kmeans.labels_
                clustered_image =  labels.reshape(top_half_of_image[0],top_half_of_image[1])
                corner_cluster =
                non_player_cluster = max(corner_cluster)
                player_cluster = 1-non_player_cluster
                player_color = kmeans.cluster_center[player_cluster]
                return player_color
        def assign_team_color(self,frame,player_detections):
                player_colors = [ ]
                for_,player_detection in player_detections.items( ):
                        bbox = player_detection[‘bbox’]
                        player_color = self.get_player_color(frame,bbox)
                        player_colors.append(player_color)
                kmeans = KMean(n_cluster=2,kmean++)
                kmeans.fit(player_colors)
                self.team_color[1]=kmeans.cluster_center[0]
                self.team_color[2]=kmeans.cluster_center[1]
        def get_player_team(frame,player_bbox,player_id):
                player_color = self.get_player_color(frame,player_bbox)
                team_id = self.kmeans.predict(player_color.reshape(0,-1))[0]
                team_id += 1
                self.player_team_dict[player_id]=team_id
                return team_id

main():
        team_assigner = TeamAssigner( )
        team_assigner.assign_team_color(frame[0],tracks[‘players][0])
        for frame_num,player_track in enumerate(tracks[‘player’])
                for player_id,track in player_track.items( ):
                        team =
team_assigner.get_player_team(video_frames[frame_num],track[‘bbox’],player_id)
                        tracks[‘players][frame_num][player_id][‘team’]=team
                        tracks[‘players][frame_num][player_id]
[‘team_color’]=team_assigner.team_color[team]

trackers.py draw annotation:
                for track_id,player in player_dict.items( ):
                        color = player.get(“team_color”)
                        frame = self.draw_ellipse(frame,player[bbox],color,track_id)
```

Step 3 : Ball Interpolation

```
main():
    tracks["ball"]=tracker.interpolate_ball_postion(tracks["ball"])
Class Tracker:
    def interpolate_ball_position(self,ball_position):
        ball_position = [ x.get(1,{}).get('bbox',[ ]) for x in ball_position]
        df_ball_position = pd.Dataframe(ball_position,[x1,y1,x2,y2])
        df_ball_position = df_ball_position.interpolate( )
        df_ball_position = df_ball_position.bfill( )
        df_ball = [ {1:{"bbox":x}} for x in df_ball_position.tonumpy( ).tolist( )]
        return df_ball
```

Step 4 : Player Ball Assigner and Team Ball Control Perc.

```
main( ):
    player_assigner = PlayerBallAssigner( )
    team_ball_control = [ ]
    for frame_num,player in enumerate(tracks["player"]):
        ball_bbox = tracks["ball"][frame_num][1]["bbox"]
        assigned_player = player_assigner.assign_ball_to_player(ball_bbox,player)
        if assigned_player != -1:
            tracks["player"][frame_num][assigned_player]['has ball']=True
            team_ball_control.append(tracks["players"][frame_num][assigned_player]
["team"])
            team_ball_control.append(team_ball_control[-1])
        team_ball_control = np.array(team_ball_control)
    tracker.drawAnnotation(……,team_ball_control)
```

Created a eucldian distance function in utils as measure distance and another function get_centre_of_the_bbox in utils.

```
Class PlayerBallAssigner( ):
    def __init__( ):
        self.max_player_dist = 70
    def assign_ball_to_player(self, players,ball_bbox):
        ball_position = get_center_of_bbox(ball_bbox)
        min_dist = 999999
        assigned_player = -1
        for player_id,player in players.items( ):
            player_bbox = player["bbox"]
            dist_left = measure_dist(player_bbox[0],player_bbox[-1],ball_position)
            dist_right = measure_dist(player_bbox[2],player_bbox[-1],ball_position)
            dist = min(dist_left, dist_right)
            if dist>max_player_dist:
                if dist<min_dist:
                    min_dist = dist
                    assigned_player = player_id
        return assigned_player
```

Draw annotations in tracker.py:

```
        for track_id,player in player_dict.items( ):
            color = player.get("team_color",team_ball_control)
            frame = self.draw_ellipse(frame,player[bbox],color,track_id)
            if player.get("has_ball",False):
                frame = self.draw_triangle(frame,player[bbox],track_id)
            frame = self.draw_ball_control(self,frame,frame_num,team_ball_control)
```

```
def draw_ball_control(team_ball_control) in tracker
        team_ball_control_till_frame = team_ball_control[:frame_num+1]

team1_num_of_frame=team_ball_control_till_frame[team_ball_control_till_frame==1].shape[0]
team2_num_of_frame=team_ball_control_till_frame[team_ball_control_till_frame==2].shape[0]
team1=team1_num_of_frame/(….+….)
team2=…..
```

Draw rectangle and put text

Step 5: Camera Movement estimator

```
Class camera_movement_estimator:
        def __init__(self, frame ):
                self.minimum_distance = 5
                lk_param = dict(….)
                first_frame_grayscale = cv2.cvtcolor(frame,color….)
                mask_features = np.zeros_like(first_frame_grayscale)
                mask_features[:,0:20] = 1
                mask_features[:,900:1050] = 1
                self.features = dict(……..mask=mask_features)
        def get_camera_movement(self, frame,read_from_stub=Fals,stub_path=None):
                # read stub
                camera_movement = [[0,0]*len(frames)]
                old_gray = cv2.cvtcolor(frames[0],colour…)
                old_features = cv2.goodFeaturestoTrack(old_gray,self.features)
                for frame_num in range(1,len(frames)):
                        frame_gray = cv2.cvtcolor(frames[frame_num],colour…)
                        new_feature =
cv2.calculateopticalflow(old_gray,frame_gray,old_features,lk_param)
                        max_dist = 0
                        camera_movement_x,camera_movement_y = 0,0
                        for i,(new,old) in enumerate(zip(new_feature,old_feature)):
                                new_feature_points = new_feature.ravel( )
                                old_feature_points = …..ravel( )
                                distance = measure_distance(new_feature,old_feature)
                                if distance > max_dist:
                                        max_dist = distance

camera_movement_x,camera_movement_y=measure_xy_distance(old_feature_point,new_feature
_point)
                        if max_dist > self.minimum_distance:
                                camera_movement[frame_num] =
[camera_movement_x,camera_movement_y]
                        old_gray = frame_gray.copy( )
                return camera_movement

        def add_adjust_position_to_tracks(self, tracks,camera_movement_per_frame):
                for object,object_tracks in tracks.items( ):
                        for frame_num,track in enumerate(object_tracks):
                                for track_id,track_info in track.items( ):
                                        position = track_info['position']
                                        cam_mov = camera_movement_per_frame[frame_num]
                                        position_adjusted = (postion[0]-cam_mov[0],postion[1]-
cam_mov[1]
                                        tracks[object][frame_num][track_id]
['position_adjusted']=position_adjusted
main( ):
camera_movement = CameraMovement( )
camera_movement_per_frame = camera_movement.get_camera_movement(video_frames)
camera_movement.add_adjust_position_to_tracks(tracks,camera_movement_per_frame)
```

Step 6 : Perspective Transformation

```
Class ViewTransformer( ):
        def __init__(self):
                court_width = 68
                court_length = 23.32
                self.pixel_vertices = np.array([ ])
                self.target_vertices = np.array([ ])
                self.pixel_vertices = self.pixel_vertices.astype(np.float32)
                self.target_vertices = self.target_vertices.astype(np.float32)
                self.perspective_transformer = cv2.getPerspectiveTransform(self.pixel_vertices,
self.target_vertices)
        def transform_point(self, point):
                p = (point[0],point[1])
                isInside = cv2.pointpolygon(pixel_vertices,p)
                if isInside is None:
                        return None
                reshape_point = point.reshape(-1,1,2)
                tarnsform_point =
cv2.perspectivetransform(reshape_point,self.perspective_transformer)
                return transform_point
        def add_transformed_position_to_track(self, tracks):
                for object,object_tracks in tracks.items( ):
                        for frame_num,track in enumerate(object_tracks):
                                for track_id,track_info in track.items( ):
                                        postion = track_info[‘position_adjusted’]
                                        postion = position.np.array( )
                                        position_transformed = self.transform_point(position)
                                        if position_transformed is not None:
                                                position_transformed= position_transformed.list( )
                                        tracks[object][frame_num][track_id]
[‘position_transformed’]=position_transformed

main( ):
view_transformer = ViewTransformer( )
view_transformer.add_transformed_position_to_track(tracks)
```

Step 7 : Speed Distance Estimator

```
Class SpeedDistanceEstimator( ):
        def __init__( ):
                self.frame_window=5
                self.frame_rate = 24
        def add_speed_distance_to_tracks(self, tracks):
                total_dist = { }
                for object,object_track in tracks.item( ):
                        if object == “ball” or object = “referee”:
                                continue
                        number_of_frame = len(object_track)
                        for frame_num in range(0,number_of_frame,self.frame_window):
                                last_frame = min(frame_num+frame_window,number_of_frames-1)
                                for track_id,_ in object_tracks.items( ):
                                        if track_id not in object_track[last_frame]:
                                                continue
                                        st_pos = object_tracks[frame_num][track_id]
[‘position_transformed”]
                                        end_pos = object_tracks[last_frame][track_id]
[‘position_transformed”]

                                distance = measure_distance(st_pos,end_pos)
                                time = (last_frame-frame_num)/frame_rate
                                speed = distance/time
                                speed/hr = speed*3.6

                                if object not in total_distance:
                                        total_distance[object]={}
                                if track_id not in total_distance[object]:
                                        total_distance[object][track_id]=0
                                total_distance[object][track_id]+=distance
                                for frame_num_batch in range(frmae_num,last_frame):
                                        track[object][frame_num_batch][track_id][“speed”]=speed
                                        track[object][frame_num_batch][track_id]
[“distance”]=distance

        def draw_speed_distance ( ):

main( ):
speed_dist = SpeedDistanceEstimator( )
speed_dist.add_speed_distance_to_tracks(tracks)
```