PLAYER TRACKING & DISTANCE MEASUREMENT USING YOLO – Nirmit Patil

Problem Statement - Objective: Track and report the total distance covered by each player on a tennis court throughout the duration of the provided video.

Video clip of a tennis match is given.

The Task is to detect both players continuously, track their positions in each frame, and calculate the total distance they travel throughout the match.

1. **Detect players**: Use any method (e.g., object detection, segmentation) to detect players in each frame.

2. **Track players**: Assign consistent identities (e.g., Player A and Player B) across frames

and extract their positions.

3. **Compute distance**: Calculate the cumulative distance traveled by each player over time

using pixel displacement or a real-world scale.

4. **Generate output**:

• Final distance (e.g., in pixels or meters) for each player.

• A processed video with visual overlays (bounding boxes or paths) showing player

movement.

Model used – YOLO

YOLO (You Only Look Once) - real-time object detection framework that predicts object classes and bounding box locations in a **single forward pass** of the network.

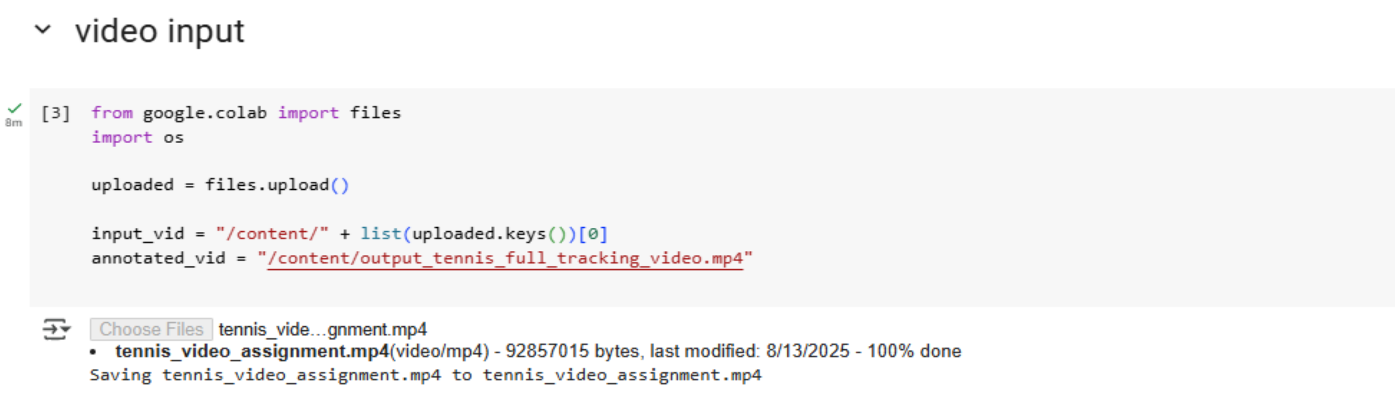
Instead of treating detection as a two-step process (region proposal + classification like Faster R-CNN), YOLO frames detection as a **regression problem** — directly from image pixels to bounding boxes + class probabilities. This makes it **extremely fast** and ideal for live video processing. (speed & accuracy are a tradeoff though)

Why YOLO for This Project? (Nano version used)

- **Speed**: Processes frames at real-time FPS, crucial for video. (fast paced setting like a tennis match)
- **Accuracy**: High performance on human/person detection tasks. (speed & accuracy are in a position of tradeoff though, however out task is simpler – only 2 primary detections (tennis players)..

Model Code walkthrough

1. **Video Input & Setup**
   - Load input video using OpenCV (cv2.VideoCapture).
   - Extract properties like FPS, width, height for processing and output video creation.



video input

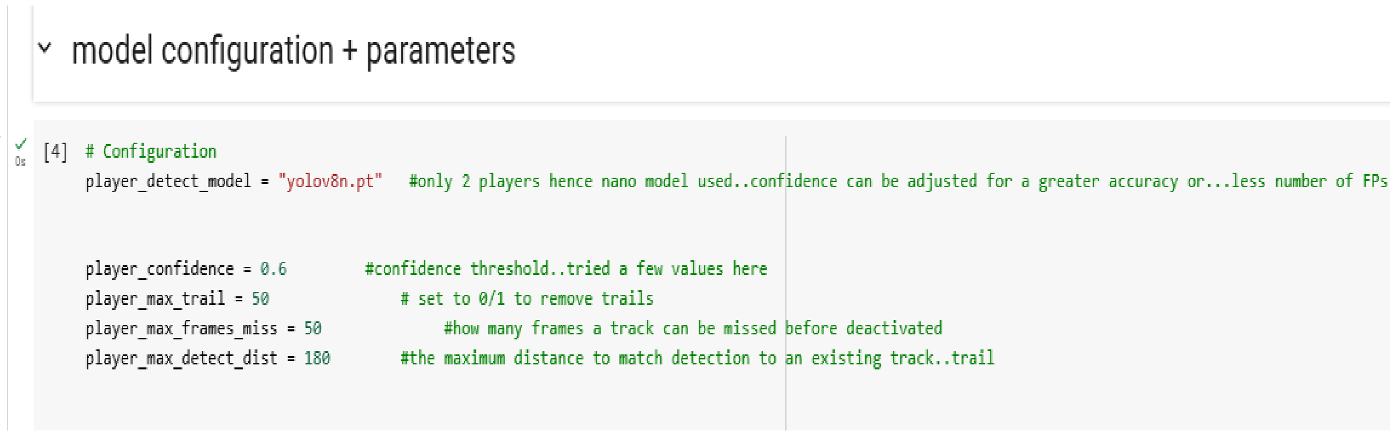
```
[3] from google.colab import files
    import os

    uploaded = files.upload()

    input_vid = "/content/" + list(uploaded.keys())[0]
    annotated_vid = "/content/output_tennis_full_tracking_video.mp4"
```

```
Choose Files tennis_vide...gnment.mp4
• tennis_video_assignment.mp4(video/mp4) - 92857015 bytes, last modified: 8/13/2025 - 100% done
Saving tennis_video_assignment.mp4 to tennis_video_assignment.mp4
```

2. **Detection Model (YOLO)**
   - Use **YOLO (You Only Look Once)** object detection model to detect players (class 0 = person).
   - Confidence threshold (player_confidence) ensures only high-confidence detections are processed.
   - Limit to **top 2 largest bounding boxes** when more than two people are detected.



model configuration + parameters

```
[4] # Configuration
    player_detect_model = "yolov8n.pt"   #only 2 players hence nano model used..confidence can be adjusted for a greater accuracy or...less number of FPs


    player_confidence = 0.6        #confidence threshold..tried a few values here
    player_max_trail = 50              # set to 0/1 to remove trails
    player_max_frames_miss = 50            #how many frames a track can be missed before deactivated
    player_max_detect_dist = 180       #the maximum distance to match detection to an existing track..trail
```

3. **Tracking (SimpleTracker)**

   - A custom centroid-based greedy tracker keeps track of detected players.
   - Assigns IDs to new detections and updates existing tracks based on nearest centroid distance.
   - Handles occlusion by allowing a configurable number of missed frames (MAX_MISSES).
4. **Persistent Naming**
   - Maps track_id to a fixed logical identity (Player A, Player B) to avoid confusion during swaps.
5. **Distance Calculation**
   - For each player, compute the Euclidean distance between consecutive centroids and sum them up.
   - Maintain two parallel totals:
     - **Logical distances** (A & B) for display.
     - **Track object cumulative distance** for final reporting.

```
def bounding_box_center(coords):
    x1, y1, x2, y2 = coords
    return ((x1 + x2) / 2.0, (y1 + y2) / 2.0)

#EUCLIDEAN dist is considered..we MAY consider other distances as welll, depending on context..
#euclidean dist best for direct/brute distance measurement

def euclidean(a, b):
    return math.hypot(a[0] - b[0], a[1] - b[1])

#colour intialisation for the IDs/names of tennis playrs, Trails & box.

def color_for_id(i) :
    rng = np.random.RandomState(i)
    return tuple(int(x) for x in rng.randint(75, 225, size=3))
#75,225 chosen randomly for colours...
```

6. **Annotation & Output**

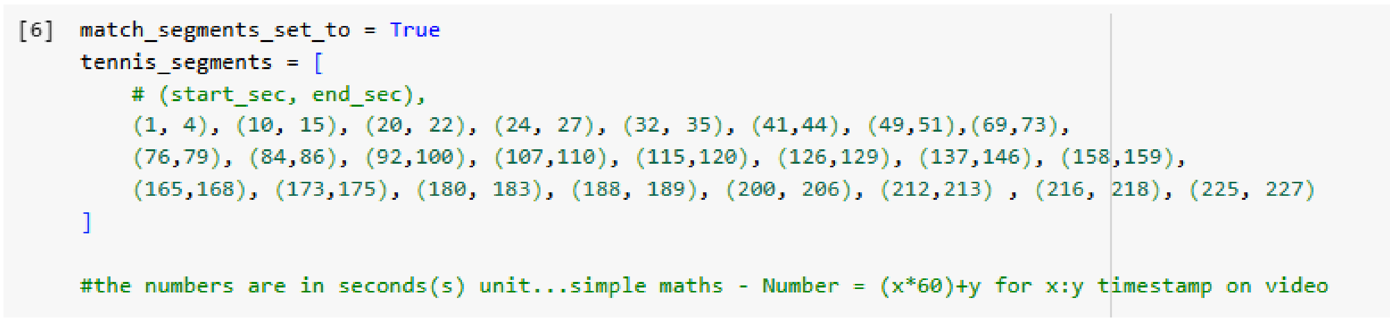
- Draw bounding boxes, names, and distance travelled on each frame.
- Draw trails showing past positions.
- Write annotated frames to output video.

APPROACH

Not every frame of the input video has the 2 players playing tennis.

There is frequent change in the setting, camera angles, crowds are being shown, replays are being shown.

Hence we FILTER OUT THE SEGMENTS of the Video where Tennis is being played

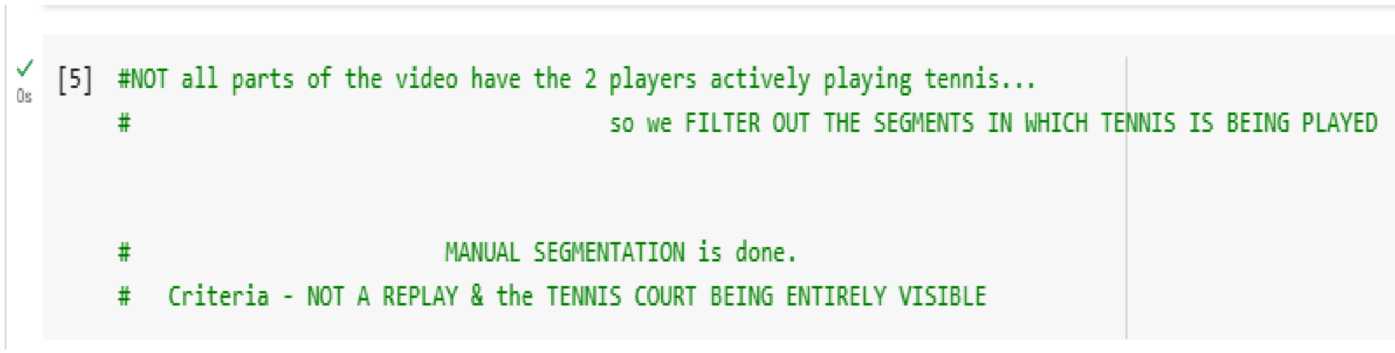


```
[6]  match_segments_set_to = True
     tennis_segments = [
         # (start_sec, end_sec),
         (1, 4), (10, 15), (20, 22), (24, 27), (32, 35), (41,44), (49,51),(69,73),
         (76,79), (84,86), (92,100), (107,110), (115,120), (126,129), (137,146), (158,159),
         (165,168), (173,175), (180, 183), (188, 189), (200, 206), (212,213) , (216, 218), (225, 227)
     ]

     #the numbers are in seconds(s) unit...simple maths - Number = (x*60)+y for x:y timestamp on video
```

Criteria for a frame or a clip to qualify as segmented frame –

1. Not a REPLAY
2. TENNIS COURT IS FULLY VISIBLE/ DISPLAYED.



```
[5]  #NOT all parts of the video have the 2 players actively playing tennis...
     #                                      so we FILTER OUT THE SEGMENTS IN WHICH TENNIS IS BEING PLAYED



     #                         MANUAL SEGMENTATION is done.
     #   Criteria - NOT A REPLAY & the TENNIS COURT BEING ENTIRELY VISIBLE
```

CONSISTENCY in Naming the Players ACROSS THE TIME FRAMES.

1. Instead of using a CNN (that we need to train on a dataset containing the IMAGES of the 2 PLAYERS from different CAMERA ANGLES THROUGHOUT THE DURATION OF THE GAME mapped with their LABELS (PLAYER NAMES)

2. Such as TRAINED CNN can then be deployed to CLASSIFY (BINARY CLASSIFICATION) or IDENTIFY in this case the 2 PLAYERS after our YOLO MODEL DETECTS THE PLAYERS.

However, due to constraints such as Lack of Time (to manually create such a dataset from scratch) & Absence of resources (lack of such a pretrained CNN or lack of such a dataset)….we go base our model on a **NAÏVE yet workable** (with some drawbacks – discussed later) ASSUMPTION.

Assumption – For every segment where Tennis is being played, the Player on LEFT side of the court will get name "Player A" & the one on right side of the court – "Player B".

We could have also done another tweak that was–

Instead of doing it LEFT-RIGHT segmentation (based on x_coordinate), we could have done it TOP-BOTTOM) segmentation…however the PLAYERS KEEP CHANGING THEIR SIDES (TOP/BOTTOM as projected onto the screen)…, hence we will stick to our LEFT-RIGHT segmentation (More consistent).

IMPROVEMENTS -

The possible improvements are –

1. Consistency in Naming the players can be easily achieved with a **CNN or likewise based model. (Trained on relevant dataset).**
2. **DeepSORT Model** – We can map the players to their names consistently by getting the APPEARANCE EMBEDDINGS / deriving features which are consistent with the players.
3. **PIXEL-TO-REAL WORLD Distance** conversion can be done by fixating the dimension of Tennis Court

(difference in x_coords = width, diff in y_coords = length)