Image Analysis & Computer Vision

Visual analysis in sport events: Tennis

2nd Semester of 2023/2024 A.Y.

Mihai-Viorel Grecu
Paolo Riva
Michelangelo Stasi

Visual Analysis in sports: Why the interest?

- Improved performance through statistics analysis
- Significant positive economic impact
- Higher quality communication approach of sports analysis
- Improved understanding of the sport for viewers

Visual Analysis in Tennis: Motivation?

- 4^{th} most popular sport: 1 billion fans
- Staggering increase in popularity over 1 year: 43% - 49%

Visual Analysis in Tennis: Goal of the project

- Trajectory computation of a tennis player through a monocular video
- Ball trajectory computation and statistics

Visual Analysis in Tennis: Goal of the project

Trajectory computation of a tennis player through a monocular video

1. Homography identification from the field to the image
2. Human Pose estimation implementation
3. Dynamics of the feet of the tennis players identification
4. Step points collection

State of the art

State of the art

Homography + Tennis player trajectory tracking + Ball tracking

HOMOGRAPHY IDENTIFICATION AND COMPUTATION

OPENCV LIBRARY

HUMAN POSE ESTIMATION

OPEN POSE

HUMAN DETECTION

+

HUMAN POSE ESTIMATION

MEDIAPIPE POSE

TENNIS BALL TRACKING

YOU ONLY LOOK ONCE

TRACKNET

TRACE

State of the art

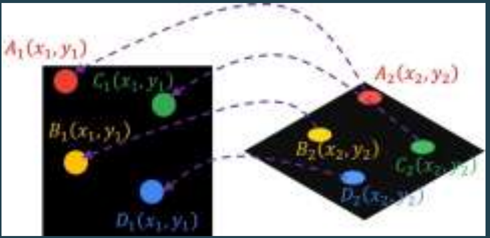
Homography

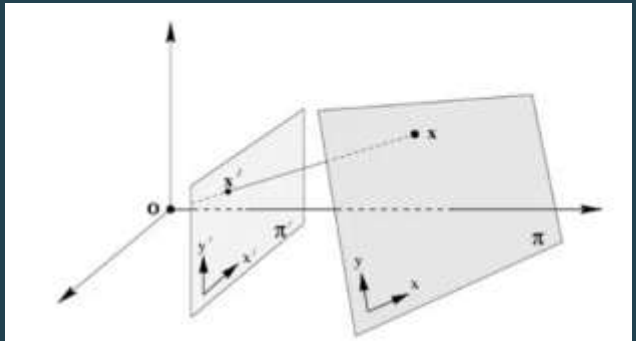
HOMOGRAPHY IDENTIFICATION AND COMPUTATION

OPENCV LIBRARY

OpenCV

$$s\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$





$X' = $ *projection of X in a homogenous coordinate plane*
−> same information, but in transformed perspective

$$\sum_i \left(x_i' - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 + \left(y_i' - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2$$

Back projection error minimized through:
cv2FindHomography *function* $\Rightarrow$
finds H between the source and destination plane

State of the art

Tennis player trajectory tracking

HUMAN POSE ESTIMATION
OPEN POSE

- **Pulls features from the entire frame**
- **Confidence maps for each of the 27 body parts of the human pose skeleton**
- **Body part association**
- **Human pose skeleton assemble**

State of the art

Tennis player trajectory tracking

HUMAN POSE ESTIMATION
OPEN POSE

HUMAN DETECTION
+
HUMAN POSE ESTIMATION
MEDIAPIPE POSE

- **Bounding box computation for human**
- **32 Landmarks prediction (single shot approach) + linking**
- **Pose refinement**
- **Temporal filtering to smooth out jitter or noise**
- **Mapping of the keypoints**

State of the art

Ball trajectory tracking

TENNIS BALL TRACKING

YOU ONLY LOOK ONCE

YOLOv8

- **Bounding boxes prediction**
- **Class probabilities computation**

In a single shot

- **Use of a threshold to keep the highest confidence box**

State of the art

Ball trajectory tracking

TENNIS BALL TRACKING

YOU ONLY LOOK ONCE

YOLOv8

TRACE + TRACKNET

- Probability-like detection heatmap for object tracking
- Upsampling to recover the information loss = pixel-wise prediciton
- Use of a threshold to keep the highest confidence box

Our Implementation

Our Implementation

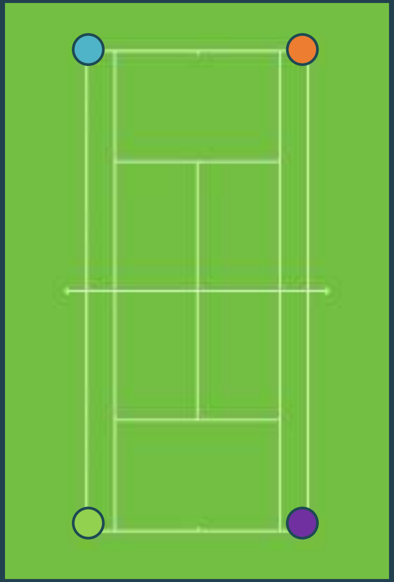
Homography Computation

Automatic Homography Computation

Hough Transform and Probability Hough Transform

First approach :

- Selecting points manually
- Map selected points with real field vertices

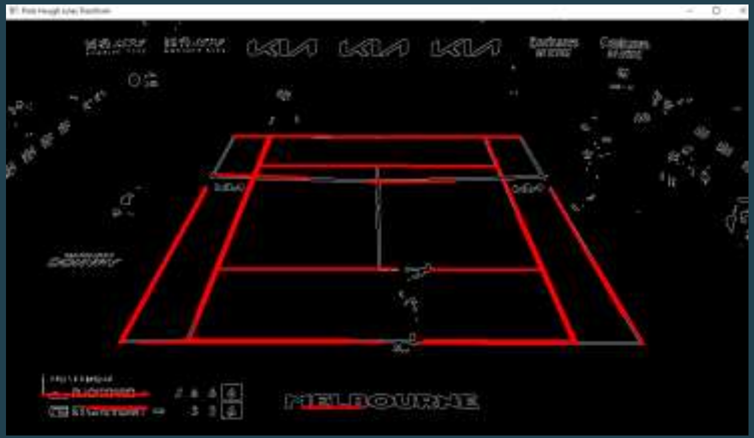


Automatic Homography Computation

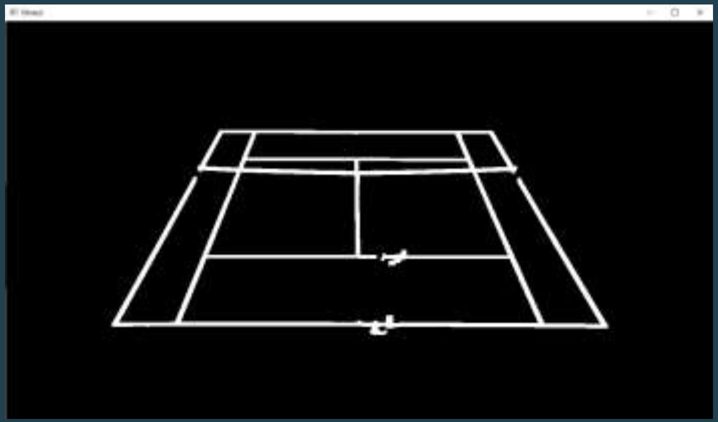
Hough Transform and Probability Hough Transform

Second approach :

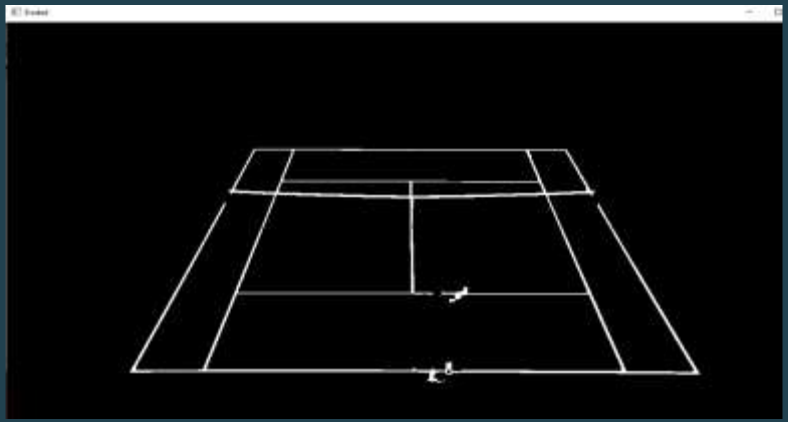
- Selecting points ~~manually~~ automatically
- Map selected points with real field vertices



Probability Hough Transform



Dilating lines:
The lines with most intersections get a fill mask command



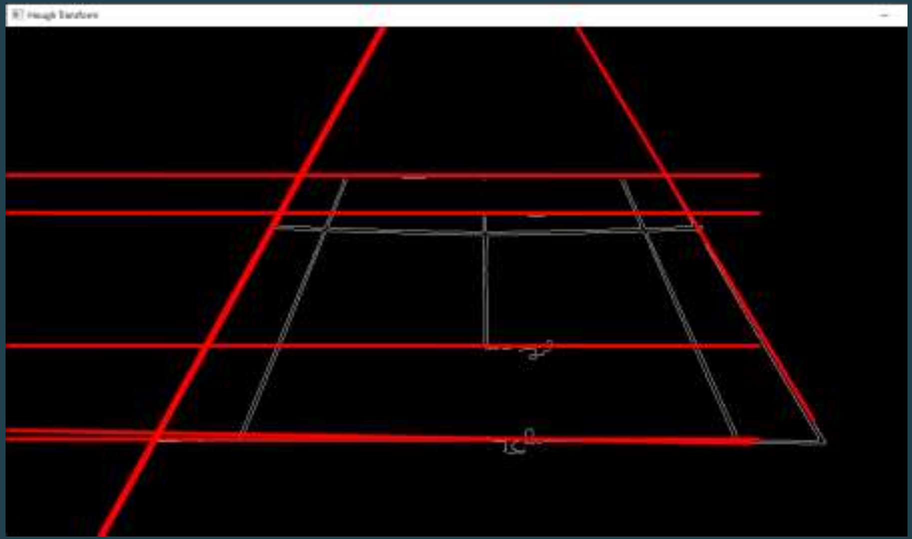
Eroding Lines

Automatic Homography Computation

Hough Transform and Probability Hough Transform

Second approach :

- Selecting points ~~manually~~ automatically
- Map selected points with real field vertices



Hough Transform:
Threshold – 300 px

Find intersections

Our Implementation

Human Detection & Human Pose Estimation

Our implementation

Human detection & Human Pose Estimation

HUMAN POSE ESTIMATION
OPEN POSE

HUMAN DETECTION
+
HUMAN POSE ESTIMATION
MEDIAPIPE POSE

ACCURACY

TIME

Human Pose Estimation - OpenPose

Implementation of the human pose estimation method in a straight-forward, out-of-the-box ready approach.

On the other hand, in terms of computational time and noise seen on the output, the model was performing poorly especially in high noise – high movement scenarios, which made the implementation not the most suitable for our case.

Human Pose Estimation - MediaPipe

Built-in model more accurate and faster than OpenPose model, even if more complex.
Two problems :

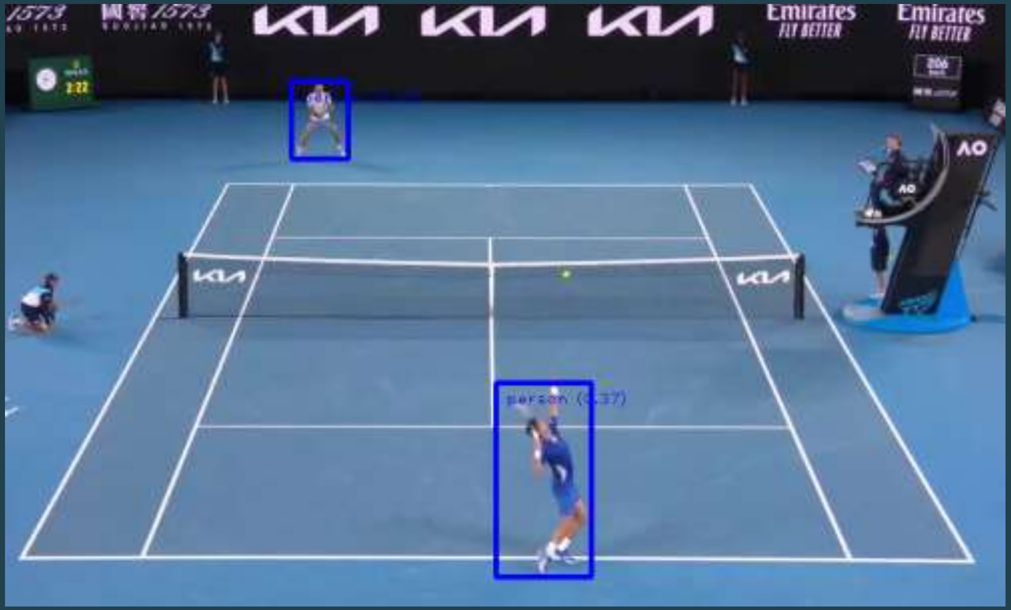
- The multi-object detection is possible but really slow if treated in a standard way;
- For efficiency, the estimation has to be done on specific regions of the frame

Human Detection - MediaPipe

Less effort demanding than the Human Pose Estimation. Pre-trained model implemented through MediaPipe library called *EfficientDet_Lite0*. Who is the player at the top of the frame and who is the one at the bottom ?

- Discrimant given by the center of the frame wrt to its height

Noise : storing the previous detection, which is updated when the distance with the new detection is below a certain threshold. Regions on which to compute Human Pose Estimation given by the extreme points reached. Each player has its own area.



Human Pose Estimation - MediaPipe

To determine whether a player was moving or not, we check if the two feet are static or not.
At each frame for each foot we check if the previous **rectified** position is at a distance greater than a certain threshold; if it is, then the foot is moving, otherwise it is static.

- Previous position updated every 5 frames
- Threshold of 5 pixels

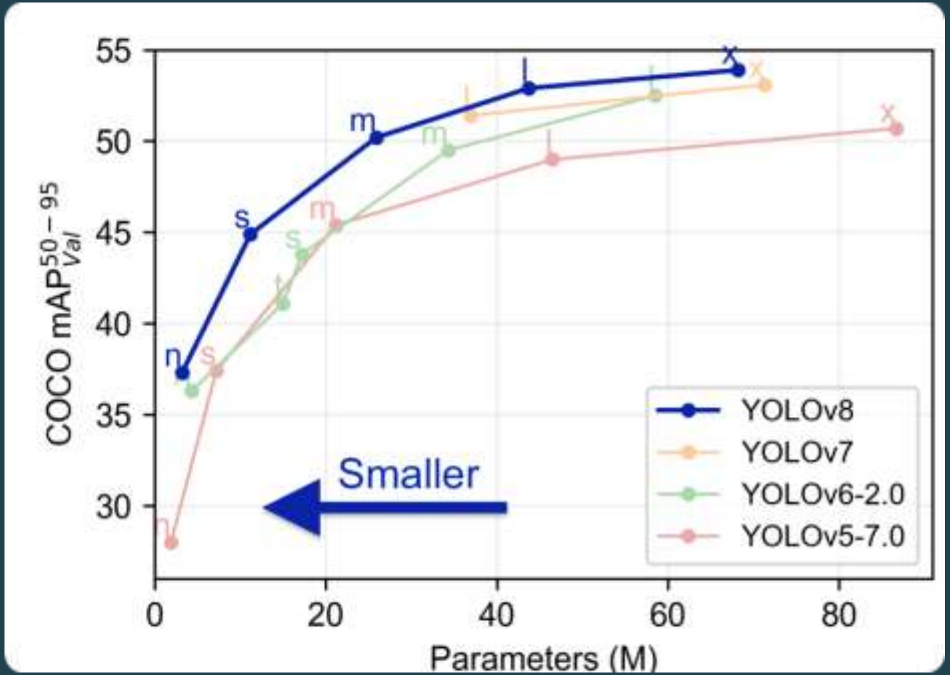
Our Implementation

Ball Detection

Ball Detection: YOLOv8

Early-stage model testing

- Most used solution for object detection
- Many versions available in terms of suitability
  - No Hardware or Software restrictions → *yolov8x.pt*
- Selection of the object through label filtering → *sports_ball*



Performance Test required in our scenario

Ball trajectory detection

YoloV8x Performance Test

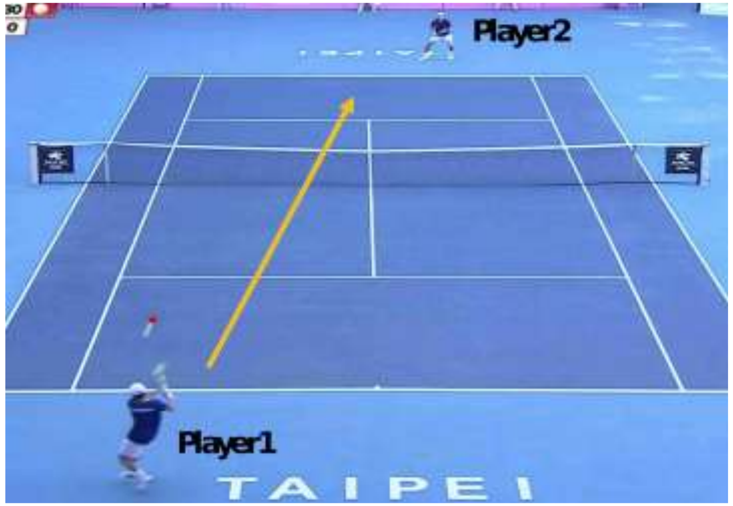

```
0: 384x640 7 persons, 1 chair, 1690.3ms
Speed: 1.0ms preprocess, 1690.3ms inference, 2.0ms postprocess per image
Total Frames: 382
Frames where a tennis ball was detected: 68
Percentage of frames where a tennis ball was detected: 17.80%
```

Ball trajectory detection

TrackNet Model implementation

YOLOv8x

TrackNet Model



- ~ 80% accuracy
- Multi-frame detection
- Trained specifically for our scenario
- Trajectory detection

Ball trajectory detection

Tennis Ball detection

TrackNet Model

TRACE interface

```python
class BallDetector:
    def __init__(self, save_state, out_channels=2):
    def detect_ball(self, frame):
        # Save frame dimensions
        if self.video_width is None:
            self.video_width = frame.shape[1]
            self.video_height = frame.shape[0]
        self.last_frame = self.before_last_frame
        self.before_last_frame = self.current_frame
        self.current_frame = frame.copy()

        # detect only in 3 frames were given
        if self.last_frame is not None:
            # combine the frames into 1 input tensor
            frames = combine_three_frames(self.current_frame, self.before_last_frame, self.last_frame,
                                          self.model_input_width, self.model_input_height)
            frames = (torch.from_numpy(frames) / 255).to(self.device)
            # Inference (forward pass)
            x, y = self.detector.inference(frames)
            if x is not None:
                # Rescale the indices to fit frame dimensions
                x = int(x * (self.video_width / self.model_input_width))
                y = int(y * (self.video_height / self.model_input_height))

                # Check distance from previous location and remove outliers
                if self.xy_coordinates[-1][0] is not None:
                    if np.linalg.norm(np.array([x,y]) - self.xy_coordinates[-1]) > self.threshold_dist:
                        x, y = None, None
            self.xy_coordinates = np.append(self.xy_coordinates, np.array([[x, y]]), axis=0)
```

Ball trajectory estimation

Frame Computation

Scene from TrackNet

Computed by ***processBallTrajectory*:**

- Default 0.5 threshold on heatmap
- Error detection:
  - In-sequence (200px thr.)
  - Out-of-sequence (100px thr.)
- Marking of points to-interpolate

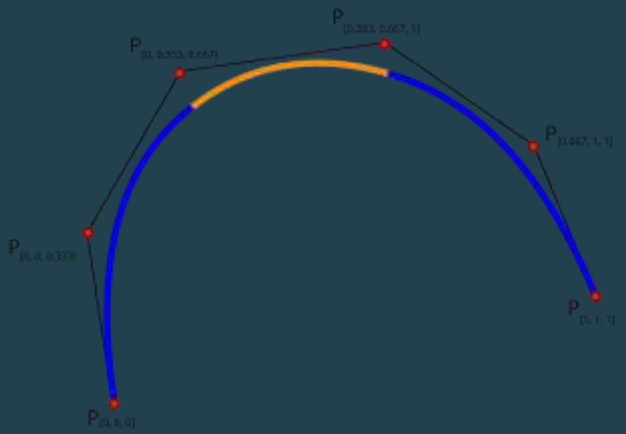
Ball trajectory estimation

Interpolation

Interpolation Domain

Existing ball positions

Interpolation results



***Interpolate missing values*:**

- **Spline Interpolation**
- **Dinamic window to use known ball positions around players**
- **Spline realisticly represents ball behaviour here**

Ball trajectory estimation

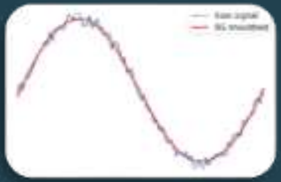
Interpolation Result

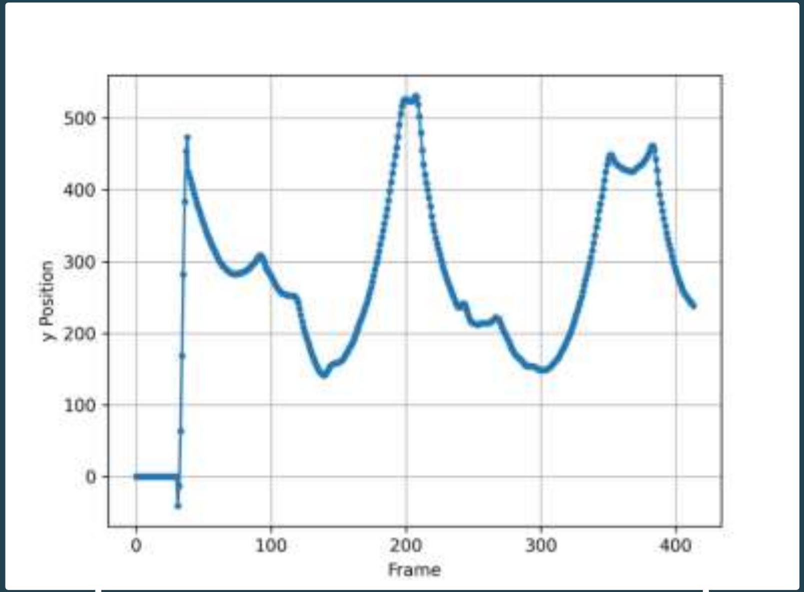
Our Implementation

Racket Hit Detection

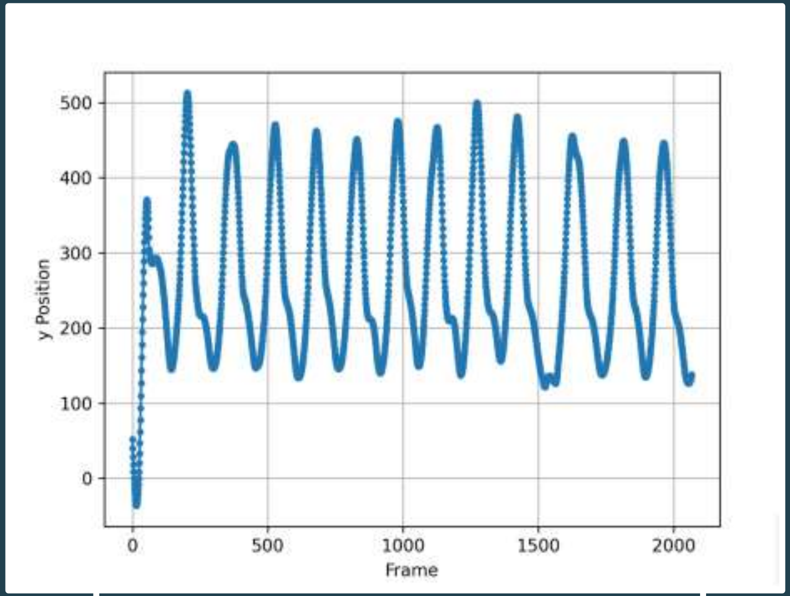
Racket Hit Detection

Filtering Ball Trajectory: Savitzky – Golay Filter





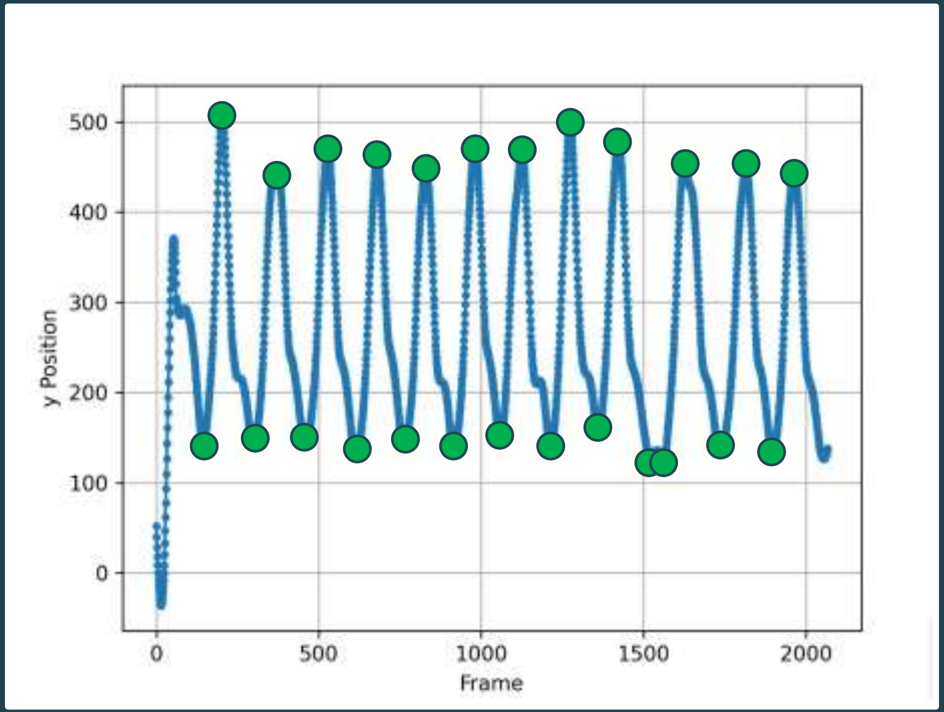
Unfiltered Position



Filtered Result

Racket Hit Detection

Filtering Ball Trajectory: Savitzky – Golay Filter



● *Detected gradient sign changes*

Outer loop

```python
for i in velocity_changes:
    append_flag = False
    min_player_distance = float('inf')
    max_wrist_velocity_sum = 0
    frame_ball_closest_to_player = i

    for j in range(max(0, i - window_around_shot), min(len(ball_positions_array), i + window_around_shot)):
        radiuses = [height_values_top[j] * radius_multiplier, height_values_bot[j] * radius_multiplier]
        if j >= len(ball_positions_array):
            break
        ball_pos = ball_positions_array[j]
        if np.array_equal(ball_pos, np.array([0, 0])):
            break
```

detect_racket_hits

```python
if (dist_right_top < max(radiuses[0], default_minimum_radius) or
    dist_left_top < max(radiuses[0], default_minimum_radius) or
    dist_right_bot < max(radiuses[1], default_minimum_radius) or
    dist_left_bot < max(radiuses[1], default_minimum_radius)):

    if j > 0:
        wrist_velocity_right_top = np.linalg.norm(np.array(rightwrist_positions_top[j]) - np.array(rightwrist_positions_top[j - 1]))
        wrist_velocity_left_top = np.linalg.norm(np.array(leftwrist_positions_top[j]) - np.array(leftwrist_positions_top[j - 1]))
        wrist_velocity_right_bot = np.linalg.norm(np.array(rightwrist_positions_bot[j]) - np.array(rightwrist_positions_bot[j - 1]))
        wrist_velocity_left_bot = np.linalg.norm(np.array(leftwrist_positions_bot[j]) - np.array(leftwrist_positions_bot[j - 1]))

        wrist_velocity_sum = wrist_velocity_right_top + wrist_velocity_left_top + wrist_velocity_right_bot + wrist_velocity_left_bot

        if wrist_velocity_sum > max_wrist_velocity_sum:
            max_wrist_velocity_sum = wrist_velocity_sum
            frame_ball_closest_to_player = j

    append_flag = True
```

Inner Loop

Our Implementation

Statistics and Performance

Statistics – Distance Travelled & Player's Trajectory

Computed starting from the moving feet detection :

- List of static centers
- Distance as the length of the graph containing all the static centers (in pixels)
- Conversion in meters using ratio computed when computing homography

```
if len(stationary_points_bot)!=0 :
        dist_bot = 0
        cv2.circle(rectified_image, stationary_points_bot[0], 2, (125, 125, 125), cv2.FILLED)
        for z in range(1,len(stationary_points_bot)):
            cv2.circle(rectified_image, stationary_points_bot[z], 2, (125, 125, 125),cv2.FILLED)
            cv2.line(rectified_image, stationary_points_bot[z-1],stationary_points_bot[z], (125,125,125), 3)
            dist_bot += np.linalg.norm(np.array(stationary_points_bot[z]) - np.array(stationary_points_bot[z-1]))/ratiopxpermtr
    dist_bot = np.trunc(dist_bot)
    cv2.putText(frame, "Bottom Player Distance : " + str(dist_bot)+" m", (50,80), cv2.FONT_HERSHEY_SIMPLEX,0.5,(70,150,255), 1)

    if len(stationary_points_top)!=0 :
        dist_top = 0
        cv2.circle(rectified_image, stationary_points_top[0], 2, (125, 125, 125), cv2.FILLED)
        for z in range(1,len(stationary_points_top)):
            cv2.circle(rectified_image, stationary_points_top[z], 2, (125, 125, 125), cv2.FILLED)
            cv2.line(rectified_image, stationary_points_top[z-1],stationary_points_top[z], (125,125,125), 3)
            dist_top += np.linalg.norm(np.array(stationary_points_top[z]) - np.array(stationary_points_top[z-1]))/ratiopxpermtr
    dist_top= np.trunc(dist_top)
```

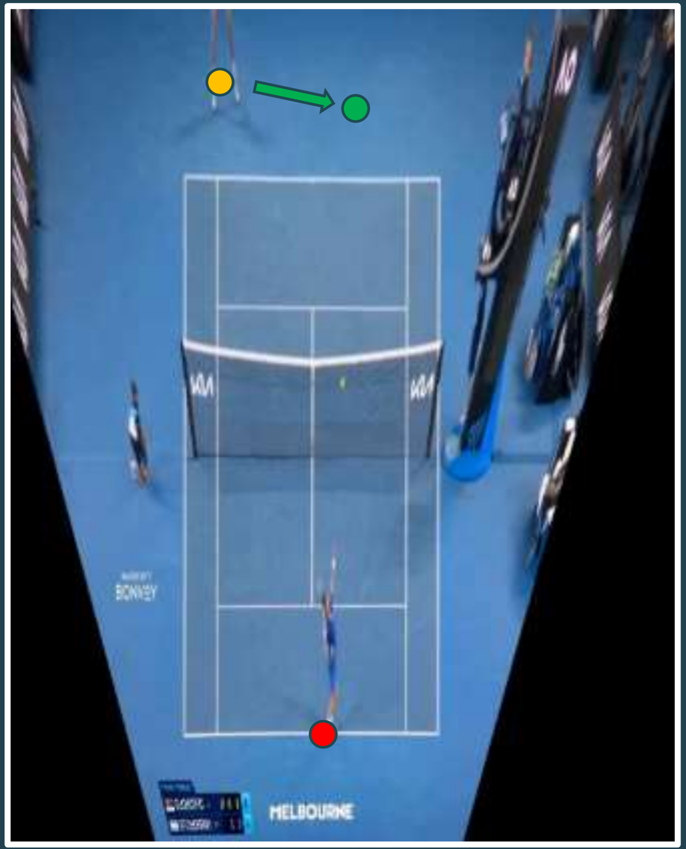
Statistics – Distance Travelled & Player's Trajectory

Trajectory Detected

Distance covered

Statistics

Average Ball Speed between hits




```python
# Pixel distance between the real centers in the transformed perspective (real)
distance = np.linalg.norm(center_start - center_end)
print(f"center_start: {center_start}, center_end: {center_end}")
print(f"Distance: {distance}")

# Time difference in seconds
time_difference = (hit_end - hit_start) / frame_rate
print(f"time_difference: {time_difference}")

# Evaluate average speed of exchange
if time_difference != 0:
    speed = distance / time_difference
else:
    speed = 0
print(f"Speed: {speed}\n")

speeds.append(speed)
```

compute_avg_ballspeed

Computation between starting hit and receiving hit

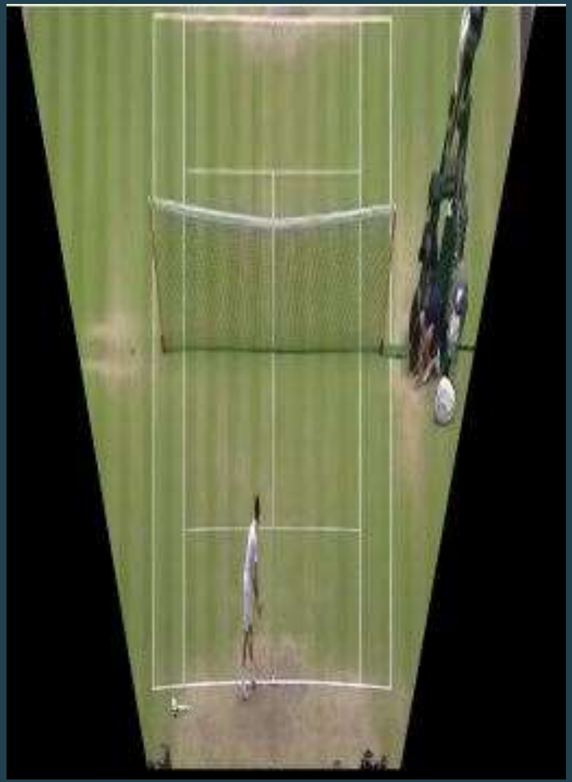
Result analysis

Performance and Tests

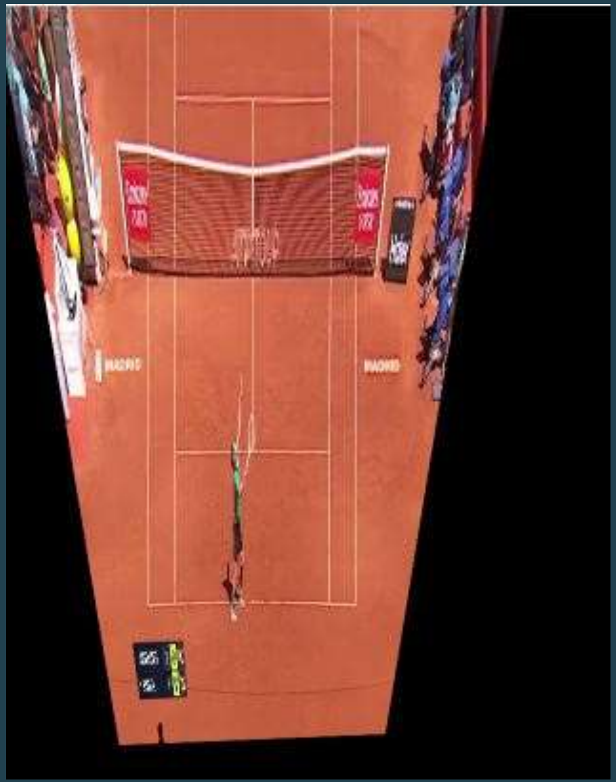
Result analysis

Tests

Australian Open



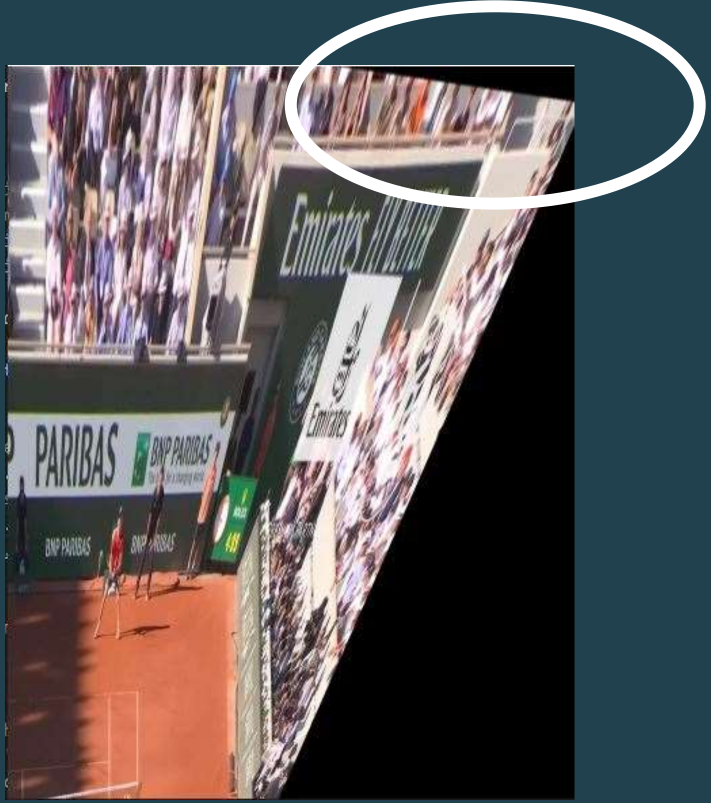
Wimbledon

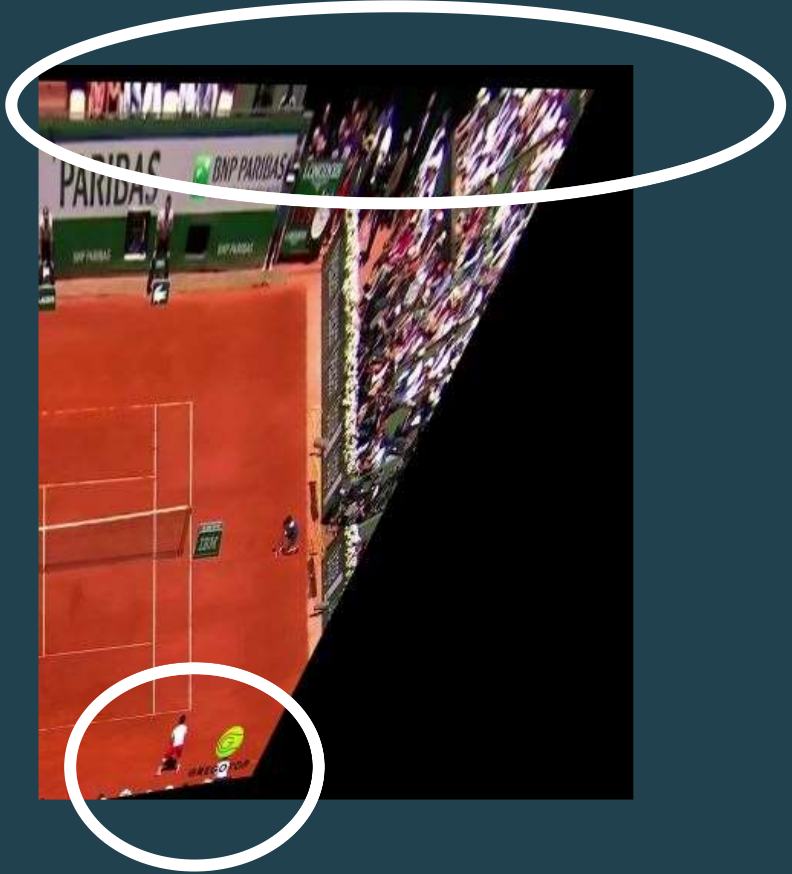


Roland Garros

Result analysis

Performance

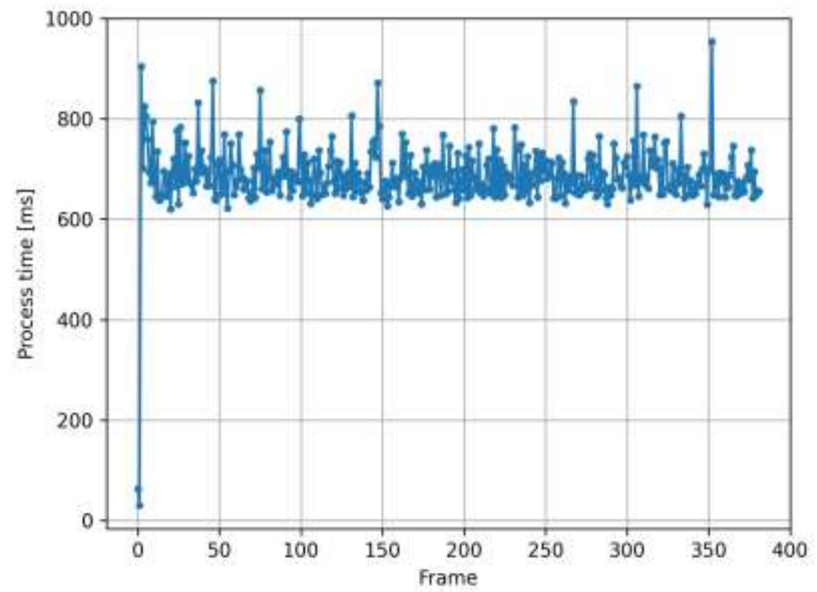




Warping issue due to external factors affecting the homography identification

Statistics

Computational Performance



Total processing time: 296.09 s
Average processing time: 775.10 ms/frame
Max: 1042.95 ms - Min: 31.03 ms

Hardware Platform:

- CPU: AMD Ryzen 7 3700X 8-core
- GPU: AMD RX 470 4GB
- RAM: 16GB DDR4 3000MHz CL18
- ROM: 512GB M.2 NVMe SSD

Software Environment:

- *Python version: 3.12.4 64bit*
- *OS: Windows 11 64bit*

Conclusions

Summary

Conclusions

Factors adding difficulty & main challenges

MAIN CHALLENGES

- Dynamical selection of the frame areas where to estimate the human pose
- Automatic detection of movement associated to players' feet
- Ball Trajectory interpolation and Filtering
- Racket Hit detection through ball trajectory
- Automatic computation of the homography from field to image

- Low quality camera
- Motion blurring

- Low resolution
- Low framerates

- Referees in the background

- Environmental changes: shade

Factors adding difficulty to the trajectory tracking of the tennis players