A 3D Tennis Complete Analysis

Omid Daliran[1], Amirreza Azari[2], Reza Heidari[3], Parham Rezaei[4], Payam Taebi[5]

[1] 400104931, [2] 99101087,[3] 400109616,[4] 400108547,[5] 400104867
[1] hopebraves@gmail.com, [2] amirrezaazari1381@gmail.com,[3] r4heidari@gmail.com
[4] parhamix@gmail.com,[5] xpayamtaebix@gmail.com

Keywords: 3D Computer Vision - Tennis - Motion Detection - Keypoint Extraction - Homography - Multiview Reconstruction - Deep Learning - Image Processing - RANSAC

Abstract

This project presents a comprehensive 3D analysis of tennis. Due to the recent advancements in computer vision methodologies, the contemporary landscape of sports analytics witnesses an increasing reliance on automated frameworks empowered by computer vision for robust data extraction. In the specific context of tennis, our investigation centers on the detection of players, ball trajectory tracking in singular or multiple perspectives, homography and the establishment of a general 3D comprehension of the court along with a framework to classify player poses. These components collectively serve as foundational pillars for the development of a reliable system designed for fault detection and precise reporting of game statistics. Our methodology integrates well-established computer vision techniques with novel concepts in a 3D analysis framework for the game of tennis. This project aims to combine conventional methods in computer vision with state-of-the-art deep learning networks to create a reliable framework for 3D tennis analysis.

1. Ball Tracking

1.1 Introduction

Detecting a tennis ball during matches presents considerable challenges. The complexity primarily arises from two aspects: the high velocity of the ball, which often renders it as a blur rather than a distinct spherical shape in video frames, and the potential for occlusion by various elements within the scene, such as players or the net. This problems guides us toward using different methods such as classic and deep learning models in order to properly detect the ball. Successfully addressing this problem offers numerous benefits, such as enabling detailed analytics including ball speed, and supporting ball-based analysis, for instance, counting net hits. Additionally, it plays a crucial role in our Enhanced Tennis Broadcast (ETB) method.

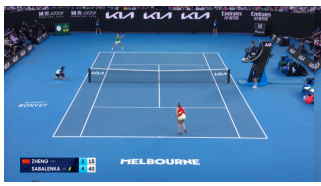


Figure 1. last frame

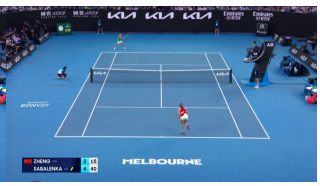


Figure 2. ball detected

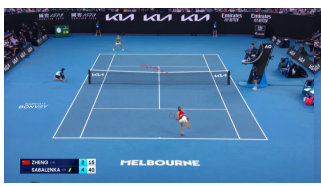


Figure 3. trajectory detected

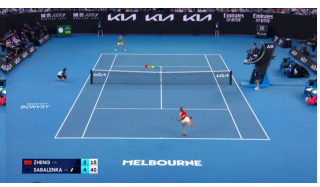


Figure 4. index colored

1.2 Related Work

Historically, there have been both classical and deep learning-based approaches to this task. The method developed by (Fazio et al., 2018) for ball trajectory estimation in tennis employs stereo smartphone videos. They utilized a combination of image segmentation, morphological processing, and multiple-view geometry. By determining point correspondences and calculating camera positions relative to a fixed origin, they were able to accurately estimate the ball's position using state estimation filtering. This detailed approach, capitalizing on accessible smartphone technology, highlights their innovative technique for trajectory analysis in a sports setting.

In addition, (Qazi et al., 2015) developed an automated ball tracking system for tennis videos, utilizing machine learning and multiple image processing techniques. Their approach involved video stabilization, random forest segmentation, and the application of saliency features to accurately detect and track the tennis ball in match videos captured by quadcopter-mounted cameras. Despite achieving a notable accuracy of 94%, their system only could handle partial occlusions.

In contrast, the deep learning approach, notably the methodology outlined in the well-known paper by (Huang et al., 2019), leverages a convolutional and upsampling U-Net structure to estimate the ball's position. Our approach aligns with this modern trend. There are projects done using this model, but due to the lack of official code they mostly used only the last frame in order to decide.

1.3 Method

TrackNet (Huang et al., 2019) , employing a convolutional neural network (CNN), accurately predicts a tennis ball's position in videos. It processes three sequential frames, each 640x360 in size, to produce a heatmap indicating the ball's likely location. This process leverages an encoder-decoder network to generate

the heatmap, followed by a Hough transform to identify the ball as a circle on the heatmap. The CNN was trained on a dataset compiled from YouTube tennis matches, using a mix of manual and automated labeling techniques. By using the best model proposed in the paper we should be able to achieve up to 99.7% accuracy in ball position prediction. The architecture is illustrated in figure 5.

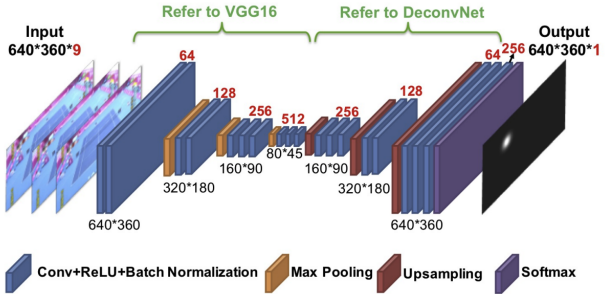


Figure 5. Ball Detection Model Architecture

1.4 Challenges

To address the occlusion problem, we have used beyond what has been used in projects which rely on only the most recent frame. The model inputs the last three frames to estimate the ball's position in the current frame, significantly enhancing the accuracy of ball tracking, as evidenced by human comparative analysis. However, this method still encounters limitations in scenarios of continuous occlusion. To overcome this, we have developed two novel approaches.

1. Interpolation: we utilize the last visible points to linearly estimate the ball's position in the current frame, effectively reducing the incidence of non-values in the model's output. While this method shows improvement, it is not remarkably superior in cases of heavy occlusion.

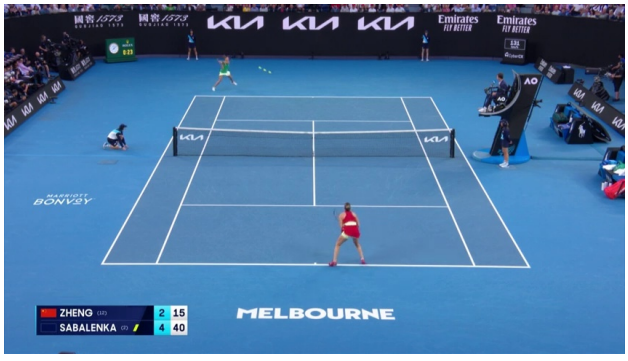


Figure 6. linearity in consecutive frames

2. Multi View Correspondence: our second method employs multi-view 3D knowledge of epipolar lines to estimate the ball's position in the current view. Utilizing two additional cameras, commonly present in major tennis matches, we can determine a more realistic position of the ball from our primary camera's perspective. This is achieved by estimating the ball's position in the auxiliary views using the model, and then applying the intersection of epipolar lines in the primary view for a more accurate estimation. For a sample of the result see Figures 7-10.

2. Efficient Tennis Broadcast (ETB)

2.1 Introduction

The Efficient Tennis Broadcast (ETB) project introduces an innovative approach to broadcasting tennis matches, particularly beneficial under limited internet bandwidth conditions. Traditional broadcasting methods often compromise video quality due to bandwidth constraints. ETB addresses this by implementing a novel broadcasting technique that focuses on the key elements of a tennis match - the players and the ball.

2.2 Method

ETB uses a fixed background image as the base layer of the broadcast. During a match, instead of streaming the entire video frame, ETB dynamically updates and transmits only specific regions: boxes surrounding the players and the ball. This method significantly reduces the amount of data required for transmission. Our human-centric experiments revealed that viewers often do not notice the static nature of the environment, as their focus remains primarily on the players and the ball. This insight is pivotal to the effectiveness of ETB.

Figure 11. Resolution Loss (1/8 of pixels transmitted)

The implementation of ETB involves real-time detection and tracking of players and the ball, which are then encapsulated in bounding boxes. These boxes are periodically updated on the fixed background, ensuring that viewers receive the most critical visual information. This technique not only conserves bandwidth but also maintains a level of visual quality and continuity in the broadcast.

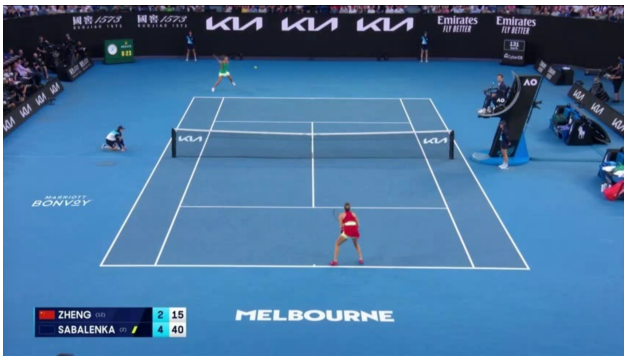


Figure 12. ETB output (1/8 of pixels transmitted)

Figure 7. unknown ball

Figure 8. ball from left view

Figure 9. ball from right view

Figure 10. ball estimation

2.3 Further Insights

For a more comprehensive understanding and visual representation of the ETB method, interested readers are encouraged to visit our GitHub repository, where a demonstrative video is available. This video showcases the practical application and effectiveness of ETB in real-world scenarios.

This section outlines the key aspects of the Efficient Tennis Broadcast method, detailing both its innovative approach and practical implementation. The invitation to visit the GitHub repository offers additional resources for readers seeking a deeper understanding or visual demonstration of the system.

3. Player Detection and Tracking

3.1 Method

Our approach to player tracking incorporates a sophisticated fusion of techniques, beginning with the utilization of MOG background removal methods to initiate the detection of player motion. Initially, we employ MOG2 background subtractor to delineate foreground masks by contrasting the current frame with an established background model, thereby isolating dynamic elements within the scene. Subsequently, we identify significant connected components within the background frame, thereby initiating the process of player detection.

Following the initial detection phase, we proceed to generate initial bounding boxes encompassing players utilizing the information gleaned from the first few frames. This serves as a foundational step in our tracking methodology, enabling the subsequent monitoring of player movement throughout the duration of the game. To achieve this, we leverage the robust capabilities of the CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability) tracking algorithm.

The CSRT tracking mechanism operates by employing discriminative correlation filters, which are adept at estimating the location of objects and continuously tracking their trajectory over time. This approach capitalizes on the discriminative power of correlation filters, allowing for precise and robust object tracking amidst dynamic and complex scenes characteristic of tennis matches.

Furthermore, to enhance the accuracy and reliability of player tracking, we incorporate morphology and connected component analysis techniques. These methods enable us to refine the detection process, identifying regions of interest corresponding to players with greater precision and reducing the likelihood of false positives or erroneous identifications.

By seamlessly integrating these methodologies, our approach to player tracking achieves a comprehensive and reliable means of monitoring player movement throughout the entirety of the game. This multi-stage process ensures that our system is capable of accurately identifying and tracking players amidst the dynamic and challenging environment of a tennis match, thereby laying the groundwork for a comprehensive 3D analysis of the game.

3.2 Implementation

The implementation of the proposed player tracking system is realized through the utilization of OpenCV, an open-source com-

puter vision library renowned for its versatility and efficiency in handling various image processing tasks. Leveraging the rich suite of functions provided by OpenCV, we have seamlessly integrated the MOG2 background subtraction algorithm and the CSRT tracking mechanism into our framework.

The initial stage of the implementation involves the application of the MOG2 background subtraction algorithm to compute foreground masks, which highlight regions of interest containing dynamic elements within the scene. This is achieved through the use of OpenCV's MOG2 function, which efficiently computes the foreground masks by subtracting the current frame from an established background model.

Subsequently, we employ morphological operations and connected component analysis to identify and isolate regions containing players within the foreground masks. OpenCV's morphology functions are instrumental in this process, allowing for the refinement and extraction of player regions based on predefined criteria.

Following the initial detection phase, we utilize the CSRT tracking algorithm to continuously monitor and track the movement of players throughout the duration of the game. OpenCV provides native support for the CSRT tracker, enabling seamless integration of the tracking mechanism into our framework.

4. Detecting Tennis Court

4.1 Detecting Tennis Court Pixels

The first task step that we take to extract all the pixels of the tennis court lines is to apply a mask which will remove all non-white pixels. The reason we are looking for white pixels is because the tennis court lines and the net line are white. Since the pixels aren't extirely white, a range of RGB values must be provided to mask colors that are not close to a white color. The mask will keep all pixels which have RGB values between [180, 180, 100] and [255, 255, 255]. The pixels that do not fit within this range will be set to black: [0, 0, 0]. Then the image in converted into a binary image to make things simplier to deal with. Since we are dealing with broadcast videos, this still leaves a lot of noise in the image as white pixels can be seen in areas other than the tennis court. The goal now is to remove the white pixels that come from objects other than the court lines.
A safe assumption that can be made is that the tennis court is very often in the center of the image. Due to this, we can safely set all pixels around the border of the image to black. We want to make this border as large as we can without risking accidently cutting off some of the actual tennis court. Cutting off 10be a safe assumption that helped with reducing the noise in the image.
A final algorithm is applied to the binary image to try to remove the white pixels that do not appear within a line of other white pixels. Let τh and τv represent the width of the horizontal and vertical tennis court lines in pixels. For each white pixel, check if the pixels h to the left and right of the given pixel is white. Similarly, check if the pixels τv above and below the given pixel is white. If the pixels h to the left and right or the pixels τv above and below then we should keep the current white pixel. However, if the pixels to the left, right, above, and below are all white, then we should not keep the current white pixel as it is occurring within a block of other white pixels and we are trying to find the tennis court lines.

4.2 Finding Tennis Court Lines from White Pixels

Now that the white pixels that mostly come from the tennis court lines and the net line have been identified, the next step is to fit lines to each of them. In order to fit a line that each of the white pixels belongs to, the RANSAC algorithm will be used. RANSAC allows you to specify a residual in which the data points are considered inliers. In order to detect the court lines, the residual can be set to approximately the width of the tennis court line in pixels. With this configuration, a single run of RANSAC with the data points being the locations of the white pixels in the image will result in a set of white pixels that occur within a line and has the maximum number of inliers within the residual offset. A line can be fit to the inliers of this particular run of RANSAC to obtain line which corresponds to a tennis court line or the net line in the frame.

4.3 Finding Tennis Court Keypoints from Tennis Court Lines

With the 9 tennis court lines and 1 net line identified, the keypoints can be identified by systematically finding the intersection of lines correspond to a particular keypoint.
In order to identify what each line corresponds to on the tennis court, a simple approach was taken. First, identify which of the lines are horizontal by looking at the angle of that line. If the angle of the line is below some threshold h, then it will be classified as a horizontal line. Once the horizontal lines have been identified, find the midpoint of each of the lines and sort them by their y-coordinates. The order of the sorted horizontal lines corresponds to the order of the lines in a tennis court: far baseline, far service line, net line, close service line, close baseline. The same approach can be used for the vertical lines, which are the lines that were not labeled as horizontal. Once the lines are sorted by the x-coordinate of their midpoints, then you can identify them as the left doubles line, left singles line, center service line, right singles line, and right doubles line. One check that can be done to verify that the court detection was done correctly is to check that there were 5 horizontal lines detected and 5 vertical lines detected.

4.4 Calibration Using Keypoints

Now that the keypoints have been located, calibrating the camera is quite straightforward. All that is required is to solve the system of linear equations to find the matrix M which maps 3D coordinates to the 2D pixel locations
To map out the corresponding 3D world coordinate points, the bottom left keypoint of the tennis court can be used as the origin, [0, 0, 0]. A standard professional tennis court is 23.77 meters long and 10.97 meters wide. The world coordinates of the top left, top right, and bottom right keypoints are therefore [23.77, 0, 0], [23.77, 10.97, 0], and [0, 10.97, 0], respectively. Lastly, the net line splits the court in half and has poles that have a height of 1.07 meters are placed 0.91 meters away from their doubles lines. This results in the top of the left pole and the top of the right pole having world coordinates of [5.485, 0.91, 1.07] and [5.485, 10.51, 1.07].

5. Pose Classification

5.1 introduction

In this section, we aim to discuss the impact of certain vision models on improving the performance of neural networks. We

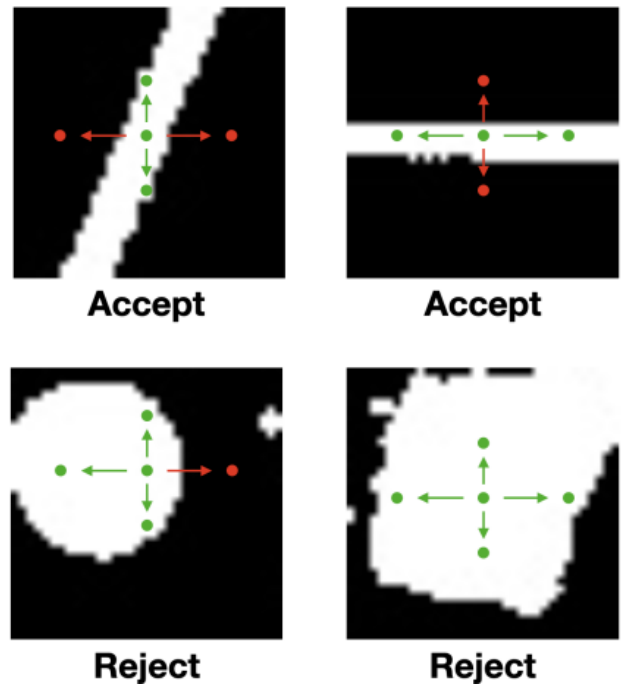


Figure 13. Examples of filtering out white pixels using τv and τh

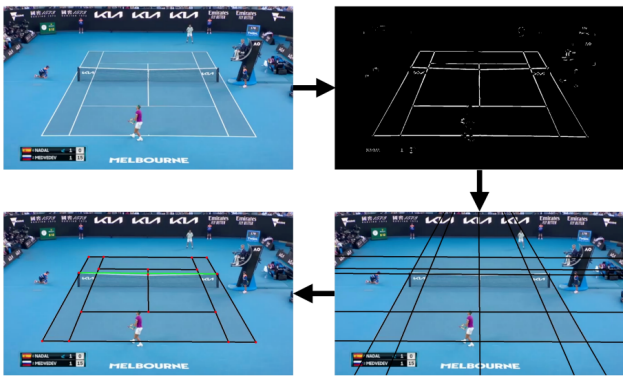


Figure 14. Top left shows input frame from broadcast video. Top right shows white masked image with the τ filtering algorithm to detect white pixels within court lines. Bottom right shows lines fitted to RANSAC inliers. Bottom left shows detected keypoints by intersecting court lines and finding the net line.

utilized the *THREE DIMENSIONAL TENNIS SHOT A HUMAN ACTION DATA SET* [1], which includes a wide variety of tennis game movements such as different types of serve strokes, etc., as explained in Appendix A. Initially, we applied a conventional neural network model for classification tasks, the details of which are provided in Appendix B. To examine the influence of classical methods, we applied several image processing techniques on the inputs of this neural network to determine whether better outputs can be achieved or not. Subsequently, we elaborate on these classical methods.

[1] `http://thetis.image.ece.ntua.gr/`

5.2 Methods

Canny Edge Detection:

The input videos of this dataset mostly have a static background, with a person standing in the middle of the video performing a specific tennis movement. Therefore, the idea that comes to mind is to eliminate any additional data that should not be fed into the neural network (Canny, 1986). For example, the color of the person's clothing or the background behind the player should not affect the network. Only the type of player movement and the movement of the racket they hold are important. Thus, we attempt to extract useful information from the video and provide it as input to the neural network in the next step. For this purpose, we first use Canny edge detection to find all the edges in the images of all frames.

Canny edge detection is a popular edge detection algorithm developed by John F. Canny in 1986. It's widely used in computer vision for detecting a wide range of edges in images or video frames. Let me break down the steps involved in Canny edge detection as implemented in the code:

- **Grayscale Conversion:** The first step in Canny edge detection is to convert the input image or frame to grayscale. This simplifies the edge detection process by reducing the image's dimensionality and removing color information that is not necessary for edge detection.
- **Gaussian Blur:** Before performing edge detection, the grayscale image is often smoothed or blurred using a Gaussian filter. This helps in reducing noise and unwanted details in the image, which can cause false edge detection.
- **Edge Detection with Canny:** Canny edge detection is performed on the blurred grayscale image. It works by detecting areas of significant intensity change or gradient in the image, which typically correspond to edges. The algorithm uses the following steps:
  1. **Gradient Calculation:** The algorithm calculates the gradient magnitude and direction for each pixel using image derivatives (such as Sobel operators).
  2. **Non-maximum Suppression:** This step thins down the detected edges to a single pixel-wide line by suppressing non-maximum pixels along the edges.
  3. **Double Thresholding:** Canny edge detection uses two thresholds: a lower threshold (weak edge) and a higher threshold (strong edge). Pixels with gradient magnitudes above the higher threshold are considered strong edges, while those between the two thresholds are considered weak edges.
  4. **Edge Tracking by Hysteresis:** Weak edges are considered as candidate edges and are traced along the strong edges. If a weak edge is connected to a strong edge, it is considered part of the edge. Otherwise, it is discarded.
- **Adjusting Thresholds:** In the code, the Canny edge detection function (`cv2.Canny()`) is called with threshold values of 50 and 150. These values determine the minimum and maximum gradient magnitudes for detecting edges. Adjusting these thresholds can affect the sensitivity and specificity of edge detection.

- **Output:** The output of Canny edge detection is a binary image where white pixels represent detected edges and black pixels represent non-edge regions.

At this stage, we are dealing with videos where edges are clearly defined. In this phase, our objective is to eliminate static edges present in all frames, as they typically represent background elements and offer no valuable information. Hence, we filter out background edges.

Subsequently, the resulting output comprises edges of the player and the racket they hold, essentially encompassing edges of any moving objects in the image. However, it also includes noise originating from the background of the image. In this section, our goal is to eradicate this noise and enhance the video's smoothness. To achieve this, we proceed as follows:

1. **Edge Detection**:
    - Utilizing the Canny edge detector, the algorithm identifies edges within each frame of the video by detecting significant intensity changes, typically representing object boundaries or transitions.
2. **Noise Reduction**:
    - Detected edges often contain noise, attributed to factors such as lighting variations or sensor imperfections. To address this issue, a noise reduction step is performed.
    - Small contours, likely to represent noise rather than meaningful features, are filtered out based on a minimum contour area threshold. Subsequently, these contours are filled to produce a cleaner edge mask.
3. **Edge Enhancement**:
    - Following noise reduction, the edges undergo enhancement to improve their visibility and robustness. This enhancement is achieved through dilation, a morphological operation that expands the boundaries of detected edges.
4. **Temporal Smoothing**:
    - In addition to processing individual frames independently, the method integrates temporal information from previous frames to achieve smoother edge transitions over time.
    - For each frame, a weighted average of edges from the current and recent frames is computed. The weighting diminishes with the temporal distance from the current frame, facilitating a gradual transition between frames.
    - This temporal smoothing effectively reduces flickering or sudden changes in the detected edges, resulting in a more aesthetically pleasing output.
5. **Implementation**:
    - The methodology is implemented in Python using the OpenCV library, a widely-used tool for computer vision tasks.
    - The functionality for edge detection, noise reduction, and temporal smoothing is encapsulated within a class named `EdgeDetector`.
    - A separate function named `process_video` orchestrates the processing of each frame in a video, utilizing an instance of the `EdgeDetector` class.

In summary, this approach combines conventional image processing techniques with temporal analysis to enhance the quality and consistency of edge detection in videos. Consequently, it finds applicability across various domains such as surveillance, motion analysis, and video editing.

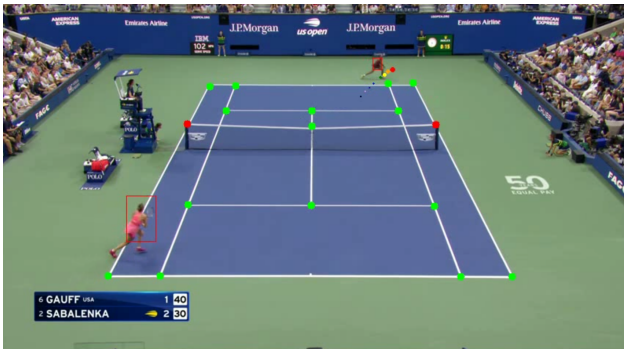


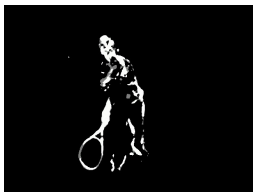
Figure 15. court with player and ball

6. Motion Detection

6.1 Overview

Motion detection is a fundamental concept in computer vision, crucial for various applications like surveillance, object tracking, and activity recognition. Essentially, it involves identifying changes in object position or appearance within a video sequence.

input frame



result

Table 1. An example of a specific frame that, according to this process, is transformed into input for a deep model, causing only important information to be fed into the neural network.

6.2 Background Subtraction

6.2.1 Overview: Background subtraction is a key technique in motion detection methodologies. It revolves around comparing each video frame against a background model to detect significant changes indicative of moving objects.

6.2.2 Implementation: Background subtraction is implemented using libraries such as OpenCV. Developers create a background subtractor object with functions like cv2.createBackgroundSubtractorMOG2(), which encapsulates the background modeling process.

Application of this technique involves invoking methods like fgbg.apply() for each frame in the video stream, resulting in a binary motion mask where foreground pixels represent areas with motion, and background pixels remain static. This binary mask effectively delineates regions of motion within the video frame.

6.3 Noise Reduction

6.3.1 Overview: Noise reduction is crucial for improving image fidelity and interpretability by mitigating unwanted artifacts. In motion detection, noise can manifest as salt-and-pepper noise, presenting as isolated white or black pixels scattered throughout the image.

6.3.2 Implementation using Median Filtering: Overview: Median filtering is a prominent strategy for noise reduction, replacing pixel values with the median value of neighboring pixels, thus reducing noise while preserving image features.

Implementation: Developers integrate functionalities from libraries like OpenCV, employing operations such as cv2.medianBlur() to apply median filtering on binary motion masks.

A critical aspect of this approach is selecting an appropriate kernel size, which determines the extent of the filtering operation. Larger kernel sizes result in more aggressive noise reduction but may compromise object edge preservation.

By iteratively applying median filtering, isolated noise artifacts within the binary motion mask are reduced, resulting in a cleaner representation of motion within the video frame.

In conclusion, integrating background subtraction for motion detection and median filtering for noise reduction forms a robust framework for extracting meaningful motion information from video streams. Through careful parameter tuning and systematic application of these techniques, practitioners can achieve enhanced accuracy and reliability in motion detection applications, driving progress across various domains.

7. Deep Learning Framework

7.1 Method

Our proposed approach consists of two main components that where introduced in (Chollet, 2017): a fine-tuned CNN for spatial feature extraction and an LSTM for temporal modeling. We begin by pre-processing tennis videos and extracting 16 frames per video at even time intervals. These frames are then fed into a pre-trained Xception CNN model, which is fine-tuned on our dataset to extract spatial features. The extracted features from the CNN's second-to-last layer are passed to the LSTM model, which processes them as a sequence of feature vectors with a sequence length of 16, corresponding to the frames per video. The LSTM model consists of a dropout layer for regularization, followed by an LSTM layer with a specified number of hidden units. Dropout layers are inserted before and after the LSTM layer to prevent overfitting. Finally, a TimeDistributed dense layer with softmax activation is used to predict the action class for each frame, and the outputs are averaged across the sequence using a Lambda layer to obtain the final prediction. the LSTM model flow can be seen as below:

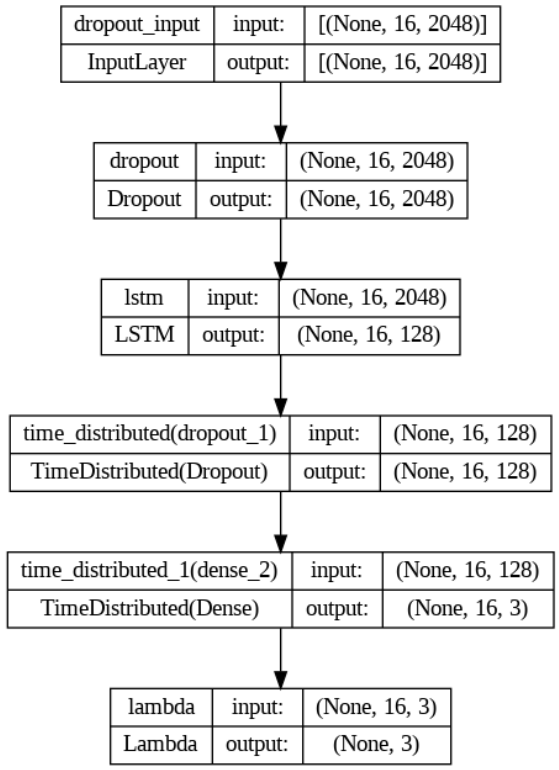


Figure 16. This image shows the flow of data in the deep model

7.2 Experimental Evaluation

We evaluate our proposed method on a dataset consisting of tennis serve videos captured from various angles and player positions. The dataset is divided into training, validation, and testing sets, ensuring a balanced distribution of action classes. We train the CNN model using the fine-tuning approach described earlier and then extract features from the second-to-last layer for all videos in the dataset. These features are used to train the LSTM model with hyperparameters such as hidden units, dropout rate, and regularization strength optimized through cross-validation on the validation set. The trained model is evaluated on the testing set using metrics such as categorical cross-entropy loss and categorical accuracy.

8. Advertising

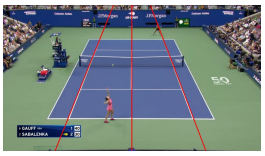
In the advertising segment, our concept involved streaming our chosen advertisement live within a designated area of an image. For instance, within the backdrop of a tennis court where ads typically appear, we aimed to substitute our desired advertisement. However, this plan faced several hurdles due to people and various objects obstructing the view, rendering conventional methods ineffective. To address this, we turned to deep learning techniques. Our solution centered around obtaining the homography matrix, which transforms the perspective view of the ground within the image into a rectangle. By utilizing the inverse of this matrix, we could integrate our rectangular images seamlessly into the image space, creating a natural appearance for viewers watching the live game.

The primary challenge in this endeavor lies in acquiring a matrix capable of seamlessly integrating the input advertisement image onto the tennis court surface. Subsequently, we will elaborate on the methodologies employed.

The homography matrix facilitates a perspective transformation between two planes in space. Within computer vision,

input advertise

input frame

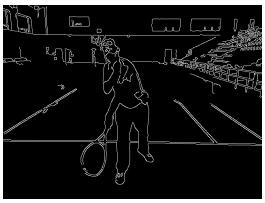


vanishing lines

result frame

Table 2. Example of how the input changes after applying the computed homography matrix obtained from the vanishing line and what the output looks like.

input frame



canny edge

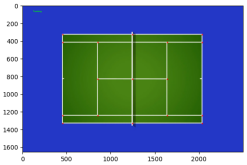
remove static edge

noise reduction

Table 3. An example of a specific frame that, according to this process, is transformed into input for a deep model, causing only important information to be fed into the neural network.

this transformation is commonly utilized to align images of the same scene captured from different viewpoints. The transformation encompasses rotation, translation, scaling, and skewing.

court points



2D mapped points

Table 4. In this table, you can see the points that are related to each other, from which the homography matrix is constructed.

A theoretical overview of how `cv2.findHomography` computes this transformation is as follows:

- **Point Correspondences:** The function requires corresponding points from two images, typically manually selected or detected using feature matching algorithms.
- **Homogeneous Coordinates:** These points are represented in homogeneous coordinates, extending the Euclidean coordinate system by adding an extra coordinate, enabling translations through matrix multiplications.
- **Homography Matrix:** The objective is to find the homography matrix $\mathbf{H}$ that maps points from one image to their corresponding points in the other image. Mathematically, for corresponding points (x, y) and (x', y'), the relationship can be represented as:

9. Homography Equation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \mathbf{H} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

where $\mathbf{H}$ is a 3×3 matrix representing the transformation.

- **Solving for Homography:** The function employs mathematical techniques to solve for the elements of the homography matrix $\mathbf{H}$, often utilizing Direct Linear Transform (DLT) algorithms.
- **Robust Estimation:** Real-world scenarios may present outliers or mismatches in corresponding points due to noise or occlusions, which can be handled using methods like RANSAC.
- **Final Homography:** After robust estimation, the function returns the homography matrix $\mathbf{H}$, representing the transformation between the two images.

In summary, `cv2.findHomography` computes the homography matrix aligning corresponding points between images, enabling accurate transformation and registration, even in noisy environments.

An alternative method for finding $\mathbf{H}$ involves determining the vanishing line and points, allowing for the construction of the $\mathbf{H}$ matrix. However, exploiting the truly vertical nature of the middle ground's vertical line in our specific case ensures consistent results compared to the previous method.

To proceed, applying the inverse of the $\mathbf{H}$ matrix to any input image, followed by scaling and translation, suffices to place it amusingly within the original image.

References

Canny, J., 1986. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, 8(6), 679–698.

Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions.

Huang, Y., Liao, I., Chen, C., Ik, T., Peng, W., 2019. TrackNet: A Deep Learning Network for Tracking High-speed and Tiny Objects in Sports Applications. *CoRR*, abs/1907.03698. http://arxiv.org/abs/1907.03698.

Qazi, T., Mukherjee, P., Srivastava, S., Lall, B., Chauhan, N. R., 2015. Automated ball tracking in tennis videos. *2015 Third International Conference on Image Information Processing (ICIIP)*, 236–240.

Method	**Performance for Different Service Poses (%)**			**Performance for All Poses (%)**		
	Training	**Validation**	**Test**	**Training**	**Validation**	**Test**
Without Classic	38.9	33.3	43.1	80.6	64.6	76.5
Fine Tuning on Classic	**100**	**56.2**	**54.9**	**100**	**97.9**	**94.1**

Table 5. Merged accuracy comparison for different service poses and all poses.