Tennis Court 2D Position Detection

Emerson Rodrigues Vero filho

June 7, 2025

Abstract

The main goal of the project was to build an application that can transform real images of a tennis match, through an homography matrix, into 2-dimensional maps of the player's positions, with a good accuracy. A U-Net model was trained to segment a tennis court from 2d images of tennis matches. The four outermost points of the binary mask yielded by the model were considered as a first guess for the homography points. A last post-processing step was taken to correct the coordinates of this points considering the closest corners detected by Harry Corners Detector. Yolo models were used to detect the player coordinates. A pipeline that converts player image points in 2d coordinates with respect to the court was built.

1 Introduction

In recent years, various application in sports analytics are fed with the detection of the key points of sports courts, ranging from automated scorekeeping and player performance analysis to real-time officiating and augmented reality-experiences. In the past, detecting the limit points relied on manual methods or rudimentary image processing techniques, that were very prune to error and inaccuracy. Recent advancements in computer vision and machine learning have opened up a world of possibilities for automating the detection of sports court boundaries. In a unsupervised fashion, one can think of applying edge detectors or Hough Transforms to detect the lines of the court, Harry Corner or a keypoint detection method like SIFT to detect the corners of the court combined with other unsupervised simple strategies. The pitfall of this strategies is that they struggle to handle differences in dynamic conditions of the images, such as lighting and shadows, or a very different color pattern. This methods alone are not suitable to be used in automated pipelines, as slightly different configurations of the images require different parameters tuning that ultimately relies in human manipulation.

Actually, current most functioning in-time tennis analysis applications utilize complex multi-cameras and other similar technologies. These systems, such as Hawk-Eye ad similar technologies, usually rely on an array of high-speed cameras strategically setups and sophisticated computer vision algorithms. There is also a human relying step that has to do with accurate camera calibration. Besides, these setups pose difficulties because of its high cost to install and to mantain.

In this work, a simple dataset of annotated tennis court images was used to build a pipeline that based on a supervised machine learning approach (a Deep-Learning U-net segmentation model) and enhanced with an unsupervised post-processing step is able to detect players positions on real-world-coordinates for images of tennis matches, and ultimately, for videos of tennis matches, if they are separated in frames.

2 Methodology

A dataset was made available on github [1]. It consists of 8841 .png or .jpg images, each associated with 14 annotated points with coordinates of the keypoints of the court, stored in a json file. The original resolution of the images is 1280×720. This dataset contains all court types (hard, clay, grass). It was originally relying

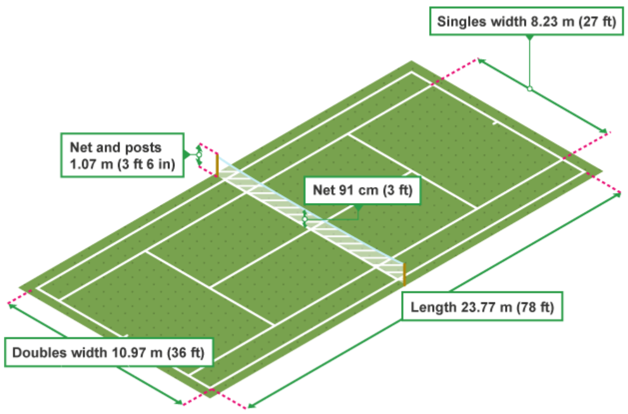
Singles width 8.23 m (27 ft)
Net and posts 1.07 m (3 ft 6 in)
Net 91 cm (3 ft)
Length 23.77 m (78 ft)
Doubles width 10.97 m (36 ft)

Figure 1: Tennis Court Real-World Measurements

designed to serve a regression model, that detects 2-d coordinates in RGB images of tennis courts [2], but after a first try to replicate it, and comparing the complexity of this problem to a simpler segmentation approach, the author decided it to reframe the problem as a segmentation one and to process the available dataset accordingly, by generating binary masks of the court that could feed a U-Net segmentation classic setting.

2.1 Data Processing

Considering the annotation set of 14 points of a sample as a start, a convex hull algorithm was applied to find the boundaries of the court. A convex hull is the smallest convex polygon that can enclose all the points [3] and in this case the output object coincides with the segments that limit the court.

Binary masks were produced for each image; points inside the hull were considered equal to 0, and outside equal to 1. A complementary folder with all the binary masks was produced.

Because of memory limitations, the images and masks resolutions were downsized to a 128x128.

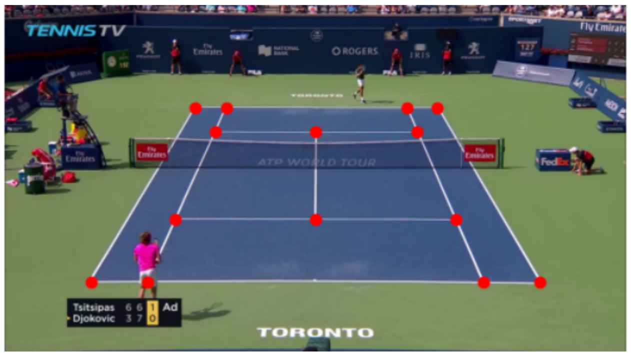

Figure 2: A sample of the referred dataset with the 14 annotations points plotted over it in red

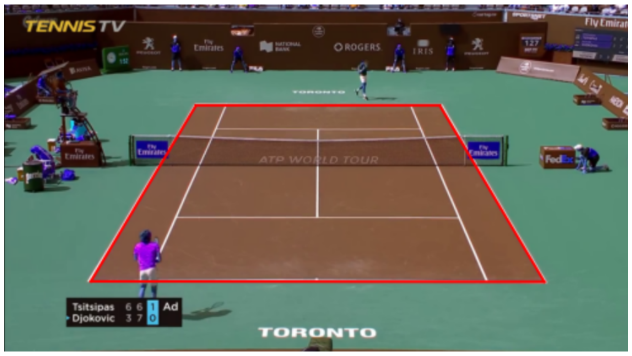

Figure 3: Convex Hull of the annotation points gives the boundaries of the tennis court

2.2 U-Net

For the segmentation task, a U-Net was used, a neural network well-suited for image processing. [4] The U-Net has a U-shaped structure with two main components: an encoder and a decoder.

The encoder functions like a typical convolutional network, compressing the image to extract important features. The decoder then expands the compressed image back to its original size while refining the details.

A key feature of U-Net is the skip connections that link layers in the encoder to corresponding layers in the decoder. These connections allow the network to combine detailed information from earlier layers with the broader context from deeper layers, resulting in

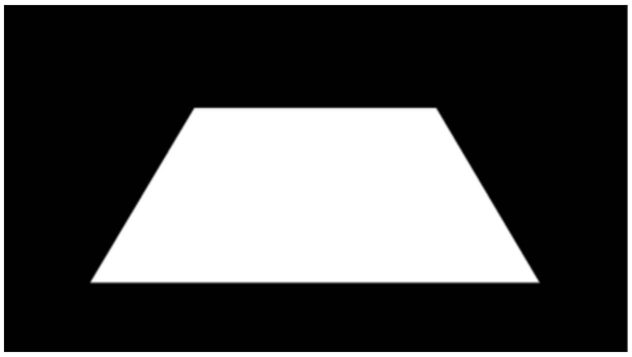

Figure 4: Generated binary mask of the tennis court

more accurate segmentations.

2.2.1 Architecture

After testing some architectures and making some hyperparameters tunning, the U-Net architecture that best suits our memory limitations and at the same time is able to learn well from the considered dataset is the following:

-Encoder:

- **Block 1:** Two 3×3 convolutional layers with 32 filters, ReLU activation, and 'same' padding. Followed by a 2×2 max-pooling layer.
- **Block 2:** Two 3×3 convolutional layers with 64 filters, ReLU activation, and 'same' padding. Followed by a 2×2 max-pooling layer.
- **Block 3:** Two 3×3 convolutional layers with 128 filters, ReLU activation, and 'same' padding. Followed by a 2×2 max-pooling layer.

Bottleneck:

- Two 3×3 convolutional layers with 256 filters, ReLU activation, and 'same' padding.

-Decoder:

- **Block 1:** A 2×2 transposed convolution with 128 filters and 'same' padding to upsample the feature map. The result is concatenated with the corresponding feature map from the encoder. Followed by two 3×3 convolutional layers with 128 filters, ReLU activation, and 'same' padding.
- **Block 2:** A 2×2 transposed convolution with 64 filters and 'same' padding to upsample the feature map. The result is concatenated with the corresponding feature map from the encoder. Followed by two 3×3 convolutional layers with 64 filters, ReLU activation, and 'same' padding.
- **Block 3:** A 2×2 transposed convolution with 32 filters and 'same' padding to upsample the feature map. The result is concatenated with the corresponding feature map from the encoder. Followed by two 3×3 convolutional layers with 32 filters, ReLU activation, and 'same' padding.

2.2.2 Cost Function

The model was trained using the binary cross-entropy loss function, which is widely used in binary classification problems, including segmentation tasks where each pixel is classified as either foreground or background. The binary cross-entropy loss for a single sample is calculated as:

$$\mathcal{L} = -\left(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})\right)$$

where y represents the true label (either 0 or 1), and $\hat{y}$ is the predicted probability for the positive class. The overall loss is computed by taking the average loss across all samples in the batch:

$$\text{Total Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \cdot \log(\hat{y}_i) + (1-y_i) \cdot \log(1-\hat{y}_i)\right)$$

where N is the total number of samples.

In image segmentation tasks, each pixel is treated as an independent binary classification problem. Therefore, binary cross-entropy is well-suited to this context, allowing the model to learn the likelihood of each pixel belonging to the foreground or background class. The binary cross-entropy loss works well with models that output probabilities (using a sigmoid activation function in the final layer), making it a natural choice for this segmentation task.

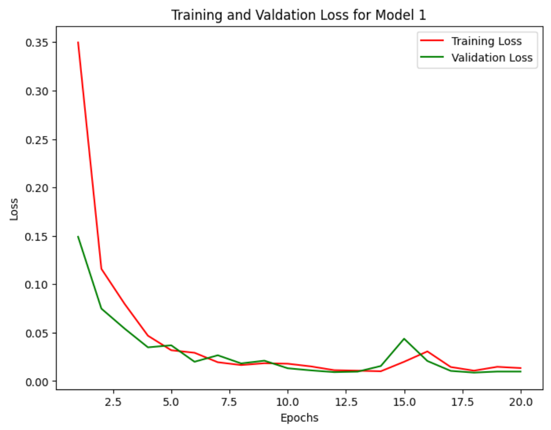

Figure 5: Model 1 Learning Curves - training and validation losses

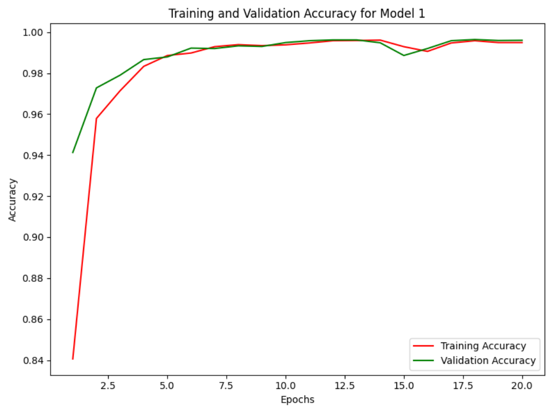

Figure 6: Model 1 Learning Curves - training and validation accuracy values

This loss function penalizes the model more heavily for incorrect confident predictions, encouraging the network to improve its pixel-wise predictions.

2.2.3 Training

Python's Tensorflow libray was used to build, compile and train the models. Besides the Binary Crossentropy strategy to compute the loss, Adam was used as optimizer, with standard learning rate of 0.01 and batch-size fixed at 8. The dataset is split into training and validation sets using an 80-20 split ratio. The random-state of all the computations was kept constant to allow easier comparisons between all possible configurations.

The accuracy and loss values were evaluated both in training and validation set during 20 epochs of training. Settings with more epochs were also tested, for the same learning rate. For higher number of epochs, after some epochs of showing a convergence trend it was observed a sudden decrease in accuracy and increase in loss values, what suggests that the chosen learning rate might be higher than the desired one at that point of the training, making the search for the optimal parameters move away from the minimum of the total loss function. Lower values of learning rates slowed significantly the training and thus were avoided.

3 Applying Yolo

YOLO (You Only Look Once) is a fast and accurate object detection algorithm in computer vision. It predicts bounding boxes and class labels for objects in a single pass through a neural network, making it suitable for real-time applications. It divides the image into a grid, predicting bounding boxes for each grid cell, and uses anchor boxes to handle objects of different sizes and shapes.

YOLO (version yolov7) was applied to the images, and it was able to pinpoint the position of the players on them. For some tennis images, as the one depicted in Figure 2, default yolo also detected the rackets as the class "tennis racket", and the ball of the game, as the class "sports ball". The focus of this work is only on the players and their bounding boxes. The output of YOLO model for each found class object are the coordinates of the extreme points of the bounding boxes (2d boxes that contain the object). The following line is an example of one part of yolo output dictionary, that contains the useful information:

```
32 0.91875 0.566667 0.0109375 0.0166667
```

This line corresponds to one object detected, the initial integer is the label of the object. The possible labels for this problem for YOLO v7 are 38, 0 and 32, that represent tennis rackets, people and sports balls, respectively.

The second and the third element of each row are the x horizontal and y vertical coordinates of the upper left corner of the box. The third and the forth coordinates are the x horizontal and y vertical coordinates of the lower right corner of the box. In the case of a person, the lower line of the box contains the point of contact with the floor (if it exists). This work models the position of the players on the court as the center of the lower limit of the bounding box, in image coordinates:

$$y_{floor} = y_2 \tag{1}$$

$$x_{floor} = x_1 + \frac{x_1 + x_2}{2} \tag{2}$$

The python library OpenCV can compute the homography matrix if at least 4 points correspondences in the two 2D spaces are given.

4 Post-Processing Step

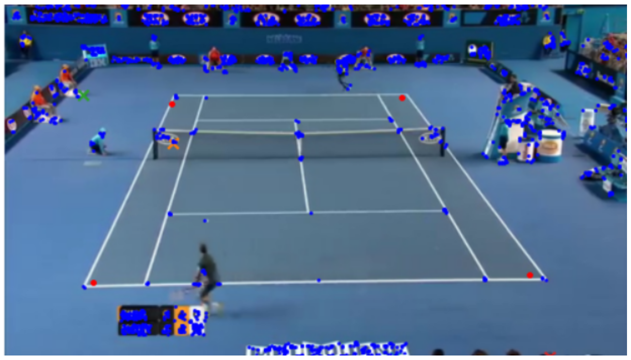

Figure 7: Harry Corners of the image are depicted as red points

Figure 7 above shows the corner points yielded by Harris Corner detector in blue for one image of the test set. It also shows the limit points guessed by the U-Net model in red. It is true that Harris is able to precisely detect the corners of the image, if occlusion does not get in its way. The issue with using solely Harris in a automated framework is that it also detects a lot of other corners that are irrelevant to this problem and it is impossible to cluster them out efficiently without using a supervised strategy. The post-processing idea is to increase the accuracy of the points yielded by the U-Net using Harris corners results. Harris algorithm outputs a binary vector that has a 1 for a position where a corner is found, and 0 otherwise. A point yielded by the U-Net can be used as a first guess as the coordinates of a 1, and if it finds a 0, it keeps searching for a 1 in its proximity considering a threshold of distance. The coordinates of the first 1 found are the ones this algorithm returns as the limit points. If no 1 was found within the threshold distance, the initial guess is returned as the limit point.

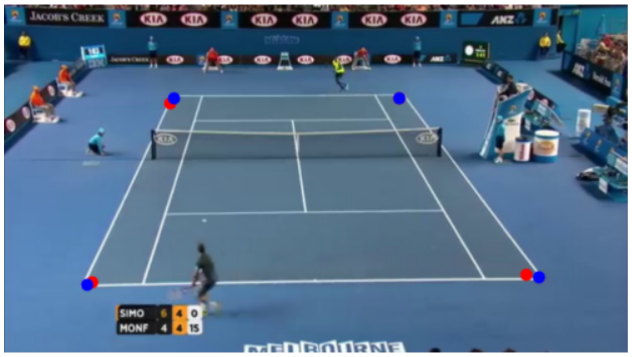

Figure 8: Initial guesses in red and final guesses in blue

5 Homography

Because only points on the floor are being considered, mapping the image coordinates into real world coordinates in this case is a 2D x 2D transformation. A homographic transform, often referred to as a homography or homographic transformation, is a mathematical transformation used in computer vision and image processing to map points from one two-dimensional plane to another. It describes the relationship between points in two different images or scenes that are related by a projective transformation. A quasi-parallelogram that forms the court in the image is mapped into a rectangle that has the dimensions of a regular padel court, as in Figure 1, using a homography matrix.

The homography transformation requires a first step of homogenization of coordinates, a z coordinate with value 1 is added to the 2D point.

To find the real-world coordinates $\mathbf{X} = [x, y]^T$, one applies the inverse homography transformation to $\mathbf{X}_h$:

$$\mathbf{X} = \mathbf{H}^{-1} \cdot \mathbf{X}_h$$

Here, $\mathbf{H}^{-1}$ is the inverse of the homography matrix.

The homography transformation is applied to all the points labeled as persons (the other classes were filtered out). The Real World Coordinates of the frame of Figure 9 is depicted in Figure 10.

6 Final Results & Conclusion

The U-net model has achieved 98.60% of accuracy on the test set and the further post-processing step visually increased the algorithm performance. Videos of tennis matches that were not present on the dataset were inferenced frame by frame, as a further tentative of confirmation of goodness of the model, and its combination with the post-processing step. For roughly 70% of the frames the results were as good as in Figure 10. A more complex setup that also segments the internal parts of the court, guaranteeing more robustness in the predictions, and more information about the right orientation of the point could steel enhance the results. Also, the masks can be built taking segmentation models of the players into account, that can give additional information to deal with occlusion situations, that might happen for instance when a player poses over a vertex. A more complete version of this application could be conceived by embedding a player classification algorithm (what could be achived using YOLO for instance).

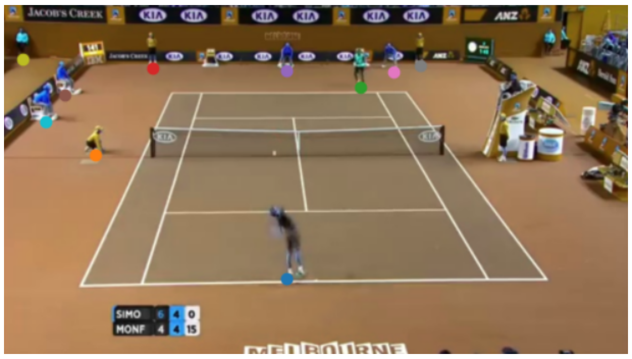

Figure 9: Yolo yielded floor locations of the people in the image

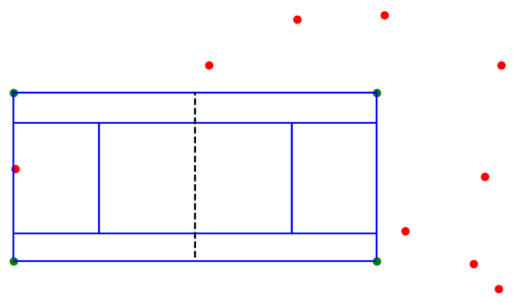

Figure 10: Good Results for 2d map of tennis court with people's positions on Figure 9

7 Conclusion

References

[1] Sergey K., *Tennis Court Dataset*, `https://github.com/yastrebksv/TennisCourtDetector`, 2023.

[2] Sergey K., *Tennis analysis using deep learning and machine learning.*, Medium, `https://medium.com/@kosolapov.aetp/tennis-analysis-using-deep-learning-and-machine-learni` 2023.

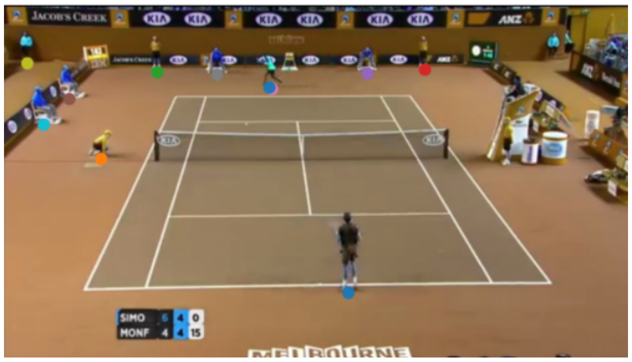

Figure 11: Yolo yielded floor locations of the people in the image

[3] R.V. Chadnov and A.V Svortsov, *Convex Hull Algorithms Review*, Medium, *Proceedings. The 8-th Russian-Korean International Symposium on Science and Technology*, 2004.Pages: 112-115

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015, pp. 234–241.