# Tennis ball tracking using Kalman Filter

**Daniel Surpanu**
4120700
surpanudaniel@gmail.com
**Digital Signal and Image Processing Project**

## 1 Problem definition

In this project, the objective is to use the Kalman Filter to track a tennis ball in a video sequence. In order to do this, it was first required to individuate the position of the ball in each frame using computer vision techniques. These positions, were then used with the Kalman Filter to estimate the position of the ball. The idea is that the position is not always available, so the filter has to infer it based on past measurements.

The project consist of three main steps:

- usage of a computer vision tools to load the video and subdivide it into frames;
- find the position of the ball in each frame;
- apply Kalman filter to track the ball.

## 2 Implementation

In this section, each step is described with the relative problems and difficulties that were encountered during the development.

### 2.1 Reading the video file

This is a fairly straight-forward process. In order to read the video file and split it into individual frames, the *OpenCV* Python library was used. The function *read_video* present in the project code, uses the *VideoCapture* class to read the video from the local storage and split it into individual frames. The function returns an array where each element is a frame from the video. The function also gives the possibility of play the video frame-by-frame, or as a whole sequence.

### 2.2 Detecting the ball

There are multiple methodologies that can be used to track objects in a video. In this project, two methods were used. The first one is based on tracking the ball using a color mask, and the second using the *Hough Circle Transform* method.

#### 2.2.1 Color based object tracking

This method relies on the fact that the object to be tracked is quite distinguishable from the background. In other words, it has a color that is distinguishable from the other colors present in the scene. Assuming that we already have the frames from the previous step, the first operation is to blur each frame with a Gaussian blur filter in order to remove all the high-frequency details. This is done because only the shape of the ball is needed, without the relative details. Each frame is then converted from an RGB color format to an HSV color format. The HSV works better when we try to isolate colors. Since the ball does not have a uniform color on its surface due to illumination conditions, moving the camera, blurring effect due to velocity, etc., a lower and upper bound of the predominant

color of the ball are defined. Using the defined upper and lower bound of the ball color, the *inRange* function is used to obtain a binary mask in which all the colors in the range are white and the rest of the image is black. This allowed to isolate the ball from the background.

On the mask obtained in the previous step, the OpenCV *findContours* function is used in order to find the contours of all the objects present in the mask. Then the minimum enclosing circles and the centers are computed. Because multiple circles could be detected, a filtering operation is performed based on the expected radius of the ball in order to obtain a single blob. Next, to mark the position where the ball could be in the image, a circle and its center are drawn. The center coordinates obtained with this procedure, were successively used as measures for the Kalman Filter algorithm.

#### 2.2.2 Hough Circle Transform method

In this case, the Python function *HoughCircles* is used. The function detects all the circles present in the image according to the passed parameters.

- The first parameter is the image in which the circles have to be detected. The second one is the method used by the algorithm. In this particular case, *HOUGH_GRADIENT* was used.
- The third parameter represents the minimum distance between the centers of the detected circles.
- The fourth parameter represents the higher threshold passed to the Canny edge detector.
- The fifth parameter is the threshold for the accumulator, when computing the circle centers. This is an important parameter because it dictated how many false positives the algorithm is allowed to consider. Smaller values of this parameter allowed more circles to be detected in the surrounding area of the true circle present in the image.
- The sixth parameter is the minimum radius of the individuated circles.
- The last parameter is the maximum radius of the individuated circles.

This method returns an array of arrays, where each element of the outer array represents a circle in the image. The inner arrays has three elements, and represent the circle center position and the radius.

In order to obtain a satisfying result, these parameters had to be carefully tuned. This is the algorithm used in the final version of the project since it was the one that gave the best results. The method that implements the ball detection is called find_ball_houg, and returns two arrays; one representing all the frames with the detected circles and the centers displayed on the image, and the other one containing all the information for the detected circles (center and radius).

### 2.3 Implementing Kalman filter

In this section is described the process used for implementing the Kalman Filter algorithm. The process starts by defining the method *initialize_kalman* which has the purpose of initializing all the variables and matrices that the algorithm uses. It begins by defining the state vector $s_pred$, which is an N-dimensional vector describing the system. In this particular case, $N = 6$. This is because the system has three variables, and each variable has to be defined individually for the $x$ and $y$ coordinate system (since we are using images, hence a 2D system). The equations that describe the system are also six, one for each variable, and each variable has to be defined for the $x$ and $y$ coordinate system. In particular, the following equations are considered:

$$x = x_0 + v_0 + \frac{a_x * t^2}{2} \qquad y = y_0 + v_y + \frac{a_y * t^2}{2}$$

$$v_x = v_0^x + a_x * t \qquad v_y = v_0^y + a_y * t$$

$$a_x = \frac{v_x - v_0^x}{t} \qquad a_y = \frac{v_y - v_0^y}{t}$$

The above equations are represented in the initialization method through the *Phi* matrix. It is a $N \times N$ matrix and it is called the state transition matrix. This matrix is presented in *Table 1*

| Variable | x | y | $v_x$ | $v_y$ | $a_x$ | $a_y$ |
|---|---|---|---|---|---|---|
| x | 1 | 0 | dt | 0 | $\frac{dt^2}{2}$ | 0 |
| y | 0 | 1 | 0 | dt | 0 | $\frac{dt^2}{2}$ |
| $v_x$ | 0 | 0 | 1 | 0 | dt | 0 |
| $v_y$ | 0 | 0 | 0 | 1 | 0 | dt |
| $a_x$ | 0 | 0 | 0 | 0 | 1 | 0 |
| $a_y$ | 0 | 0 | 0 | 0 | 0 | 1 |

*Table 1: Transition matrix*

In the matrix, the time difference between two states is represented by $dt$, the object's x position is represented by $x$, the object's y position is represented by $y$, the velocity in the x coordinate is represented by $v_x$, the velocity in the y coordinate is represented by $v_y$, the acceleration in the x direction is represented by $a_x$, and the acceleration in the y direction is represented by $a_y$. $a_y$ is always known in this case, and it is given by the gravitational acceleration ($-9.8m * s^{-2}$).

The *covariance matrix* $P_pred$, which is an $N \times N$ matrix, is then defined. The $Q$ matrix represents the *covariance noise matrix*, and it is an $N \times N$ matrix. $H$ is instead the measurement matrix, and it is a $M \times N$ matrix, where $M$ is the measurement vector length. In this particular case, we have $M = 2$ because the measurements represent the $x$ and $y$ position of the center of the ball. Finally the $R$ matrix, which is a $M \times M$ matrix representing the measurements noise, is defined. This method returns all the aforementioned variables initialized to default values.

The method predict_kalman implements the prediction step in the algorithm. It starts by computing the $s_pred$ variable, which represents the previous state of the system. This is computed using the $Phi$ and $s$ variables as defined in Algorithm 1. After it computes the previous state of the covariance matrix by using the $Phi$ matrix, the current covariance matrix $P$ and the covariance noise matrix $Q$. The method returns these two computed variables.

Now the update step is implemented in the *update_kalman* method. First, the current measurement vector is acquired (variable $m$). The Kalman gain is computed as described in the algorithm. Following that the innovation is computed using the current measurement, the measurements matrix $H$ and the system's previous state $s\_prev$. Then current state and the current covariance matrix are computed and returned by the method.

The last method is the run_kalman method and it has the objective of correctly call the above defined methods. It starts by invoking the *find_ball_bough* method, which returns all possible positions of the ball in the frames. Since multiple or none of the positions can be returned by this method, some further considerations had to be made before using these values. The first consideration is that if the method did not return any ball position in the first frame, then the ball position is set to $(0, 0)$. This is done with the method set_initial_position. Then the Kalman initialization step is made by calling the initialize_kalman method. As already mentioned, the ball detection could return false positives, and this would decrease the algorithm performance. In order to prevent this and improve the performance, the idea is to filter all the returned ball positions, and take into consideration only the position that is closer to the previously measured position. To do this, a vector with two elements, which represent the $x$ and $y$ position of the previous measurement, is defined and updated accordingly.

At this point, the iteration step starts by looping through all the video frames and applying the update and predict steps. Now the possibility that no ball is detected detected by the detection algorithm in the current frame is again taken into consideration. The two possibilities that were considered are: $i$) to ignore the frames in which any ball is detected, $ii$) to set the current measure to the value of the previously measured position. In the final implementation, the first option was used. If there is at least a position returned by the detection step, the update step is applied by invoking the *kalman_update* method. This method returns the current state, which represents the position of the ball and the covariance matrix. Before invoking the kalman_update method, the compute_nearest_measure method is invoked which computes the nearest position, in case multiple position are returned by the detection step, with respect to the previously detected position. Then the predict step is applied using the values returned form the update step; the predicted state $s\_prev$ and the predicted covariance matrix are hence computed. The position of the ball is then saved for each frame for later use. After computing all the positions, the circles representing the ball location, with the contour and the center, are drawn on each frame. The run_kalman method returns an array with the frames comprising the

drawn balls' positions, and the array containing the values of the predicted positions. This method concludes the implementation of the Kalman Filter algorithm.

## 3 Considerations

The most difficult part of the project resulted to be the detection part. In particular, using the first mentioned method (color detection), it was quite difficult to find good thresholds to isolate the ball color. This is because the background in almost all the analyzed videos was very "noisy". Another problem was that the ball resulted very small, deformed, and hard to see in the frames due to the high velocity and low frame rate. Another difficulty was finding an effective method to remove the false positives in the detection step.

## 4 Results

In this section, the obtained results are presented and some other considerations are discussed. In order to have a better performance, the choice was to apply the algorithm on a slow motion video and from a perspective where the ball was clearly visible. In Figure 1 the results obtained with the first video are presented.

*Figure 1*

In the above image, the coefficient for the covariance noise matrix is set to $100$, the coefficient for the measurement noise matrix is set to $10$, and the $dt$ is set to $10$. In fact, it can be seen from the image that the model is quite precise, without much variation in the predictions. This is because it trusts more the measurements, and because the measurements are quite often. Another important thing to notice is the error in predicting the trajectory when the ball suddenly changes direction. It can be seen in the image on the right that the prediction for a certain number of frames was very off (at the feet of the player), but then it corrects the trajectory when new measurements are received. This is a well-known behavior in the Kalman Filter.

In the next images, the setting was different. The covariance and the noise matrices are kept the same but the $dt$ is set to $30$. The images emphasize how with a smaller $dt$ (less frequent measurements), the trajectory predictions varies allot. This, as expected, is because the algorithm has to infer the position of the ball with less knowledge (measurements).

*Figure 2*

Below is an experiment carried out with another video. The settings remain the same as above. What it can be seen from the images is that there is a lot more noise in the predictions. After some analysis, it was discovered that the main reason is that in some frames, Federer shoes are detected as the only circles in the image, so the algorithm tries to compensate for this anomaly in the measurements.

The conclusions that is drawn from the results is that having accurate measures is very important. Also, the frequency of the measures is crucial for obtaining a good result. More frequent and accurate measures allow the algorithm to correct itself when the predictions are wrong, or when the trajectory suddenly changes. As already seen, having accurate measures in this case was not an easy task, due to the fact that the object to be detected is small, it is often deformed due to the impact with the surfaces, and it is not clearly visible due to the high velocity. Considering all these facts, I think that the final result is pretty god.

The project is also available on GitHub at https://github.com/danigit/kalman_filter_exam.git. In the *data* folder are present the videos that were used in the project, and in the *result* folder are present the obtained results.

*Figure 3*

**Algorithm 1:** Kalman Filter Generic

**Initialization:** $\hat{s}_1 = m_1,\ \hat{s}_2^- = \hat{s}_1,\ P_2^- = P_1 = R$
**for** *k = 2 to N* **do**
  **1** *Acquire the measurements* $m_k$
  **2** *Set the Kalman gain* $K_k = P_k^- H^\top (H P_k^- H^\top + R)^{-1}$
  **3** *State update* $\hat{s}_k = \hat{s}_k^- + K_k(m_k - H\hat{s}_k^-)$
  **4** *Variance update* $P_k = (I - K_k H)P_k^-$
  **5** *State estimate projection* $\hat{s}_{k+1}^- = \Phi \hat{s}_k$
  **6** *Variance estimate projection* $P_{k+1}^- = \Phi P_k \Phi^\top + Q$
  **7** k $\leftarrow k + 1$
**end**

## References


[1] Alessandro Verri. Aulaweb - *DSIP notes*.

[2] Rvkrish. Github - *Kalman filter ball tracking*.

[3] OpenCV. *OpenCV documentation*.

[4] Adrian Rosebrock. *OpenCV Track Object Movement*.