FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Walter Herold Veedla

Automatic analysis of squash straight drives accuracy from a single camera view

Department of Software and Computer Science Education

Supervisor of the bachelor thesis:	doc. RNDr. Elena Šikudová, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2023

I would like to thank my supervisor doc. RNDr. Elena Šikudová, Ph.D. for her patience and supervision during the project.

Title: Automatic analysis of squash straight drives accuracy from a single camera view

Author: Walter Herold Veedla

Department: Department of Software and Computer Science Education

Supervisor: doc. RNDr. Elena Šikudová, Ph.D., Department of Software and Computer Science Education

Abstract: Squash is a racket and ball sport with an estimated 20 million players worldwide. Compared to sports like tennis and golf, squash tracking and analysis systems are relatively underdeveloped and performance analysis is often done by manual instruction or by pencil-and-paper. While in the recent years more advanced squash specific technology has become available, it requires high-cost specialized hardware and does not capture the location of the bounce of the ball on the floor. This project attempts to tackle this gap of existing squash analysis tools by using computer vision techniques to automate the collection of shot data of a common squash training drill ”straight drives”, where the ball is being repeatedly hit parallel to a side-wall of the court.

An analytics program is developed that can process a video file of a player performing the ”straight drives” drill and produce accuracy metrics from the video. The result of this work is a computer program that allows an easy way for the user to get feedback from their training and track their progress.

Keywords: image processing, video, sport

Contents

Introduction

Squash is a competitive racket and ball sport played by millions of players around the world. Unlike in tennis where the ball is hit over a net, in squash the ball is hit against a wall from which it rebounds. Being able to hit the ball and make it travel parallel to the wall is considered one of the most important skills in playing squash. This skill is generally developed by practicing alone and training with partners.

Tracking accuracy metrics in squash is important not only for professional players but also for amateur players. However, tracking these metrics can be tedious and time-consuming, often requiring a human annotator to manually record data. Without an external annotator, players may have trouble accurately assessing their own performance as their focus is divided between performing well in the moment and tracking their overall progress. An automated solution for collecting straight drive performance data in squash could help players track their progress and get an objective view of their results. This thesis presents a solution to this problem by outlining the design and implementation of a tool for visualizing and calculating statistical metrics on straight drive accuracy in squash. The tool uses a video recording of a player performing the straight drive exercise to automatically extract data on the number of shots that land in the target area. This allows players to focus on maximizing their performance without worrying about tracking their own data.

Chapter 1 provides an overview of the game of squash and explains why straight drives are an important aspect of the sport. Chapter 2 gives an overview of the existing research on squash analytics and discusses the current state of the art in this field. Chapter 3 describes the dataset that was used to develop and test the application. Chapter 4 covers the design and implementation of the tool in detail, and Chapter 5 evaluates the effectiveness of the algorithms used.

1. Introduction of squash

Squash is a racket and ball sport played by an estimated 20 million players in over 185 countries (US Squash). Unlike tennis, to which it is most often compared to, the game is played by two players in a shared four-walled court with a small and semi-soft rubber ball. The aim of the game is to hit the ball in such a way that the opponent is unable to produce a valid return before the ball bounces off the ground twice. In squash, players are allowed to exploit the geometry of the court and as such, the ball may be hit any side wall and change direction multiple several times during its flight, on condition that it bounces off the front wall once. These factors play a role in characterizing squash as a complex physical, technical, and tactical sport (Jones et al. [2018]). Squash has also been once described as the world's healthiest sport by Forbes magazine (Santelmann [2003]).

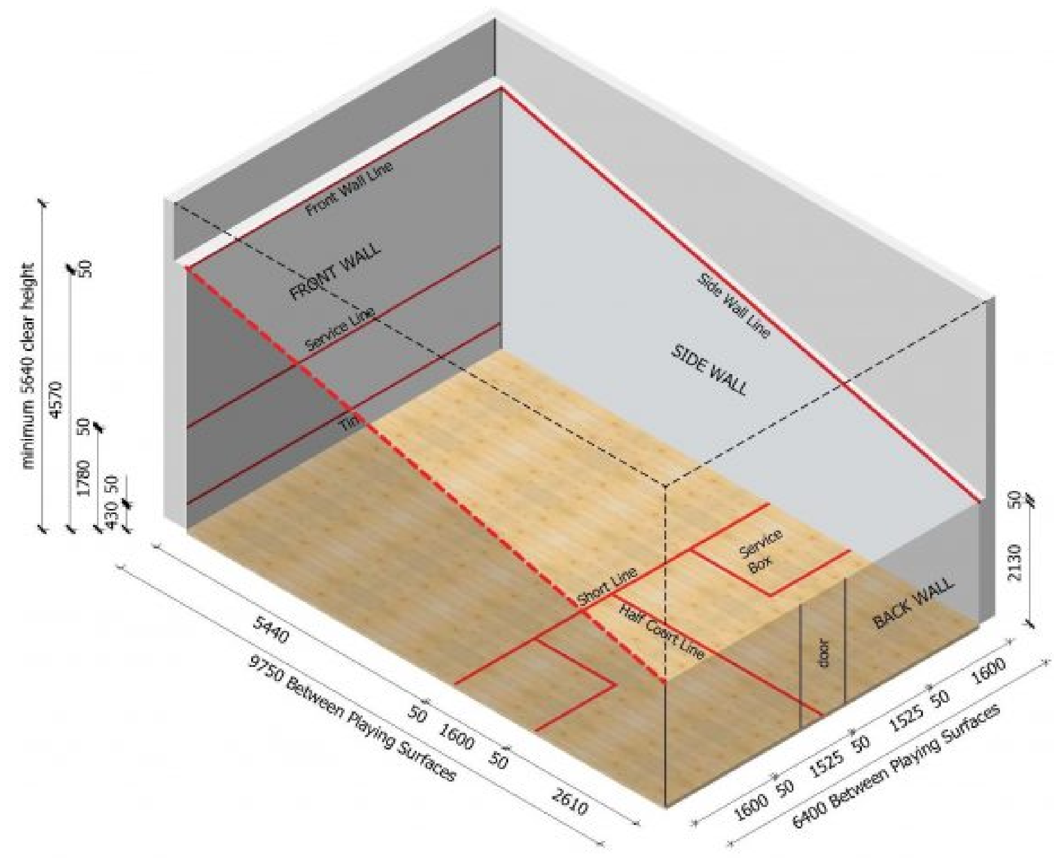


Figure 1.1: Illustration of World Squash Federation squash court specification (World Squash Federation [2016]).

As of January 2023, there are more than 950 players registered with the Professional Squash Association (Professional Squash Association [2022]) (abbreviated PSA) and a series of tournaments are held regularly around the world with prize purses ranging from a few thousand up to a million dollars. Though bidding, squash is not yet a part of the Olympic Games program.

Squash has many different types of shots, and while this thesis focuses specifically on straight drives, it can be beneficial to provide an overview of the general terminology of shots for the reader. Some examples of the various types of shots in squash include:

Shot	Description
Drive	"A shot played after the ball has bounced off the floor and is hit at the front wall with speed"
Cross	"A drive played at an angle such that the ball is hit across the body"
Boast	"A shot played such that the ball hits a side or back wall before the front wall"
Drop	"A shot that is hit such that the ball hits the front wall (gently) and lands close to the front wall"
Lob	"A shot that is hit off the front wall with a high arc, such that it lands near the back of the court"

Table 1.1: Definition of shot types (Williams et al. [2017])

1.1 Relevance of straight drives

As a competitive racket and ball sport, squash requires a range of skills in order to be successful. One of the most important of these skills is the ability to hit the ball and make it travel parallel to the wall - a straight drive. This type of shot is not only the most common in professional squash matches (Vučković et al. [2013]), but is also considered by some experts to be the most important skill in the game. For example, Ross Norman, a former world number two, has been quoted as saying that developing the ability to hit the ball straight down the wall is the key to success in squash (McKenzie [1993]). This follows, because as the ball is traveling close to the wall, it leaves the opponent less room to maneuver and can force them into making a weak return.

The importance of straight drive accuracy is supported by recent academic research. In a study by Williams et al. [2017], the researchers administered the Hunt Squash Accuracy Test, which includes four skill tests that measure a player's straight drive accuracy. The study found significant correlations between a player's tournament rank and their score on the drive tests, further emphasizing the importance of straight drive accuracy in squash.

By focusing on straight drive accuracy, this thesis aims to provide information and statistics for players interested in improving their performance in squash. By automating the tracking and analysis of straight drives, the aim is to make it easier for players to track their progress and identify areas for improvement, ultimately helping them to succeed in the game of squash.

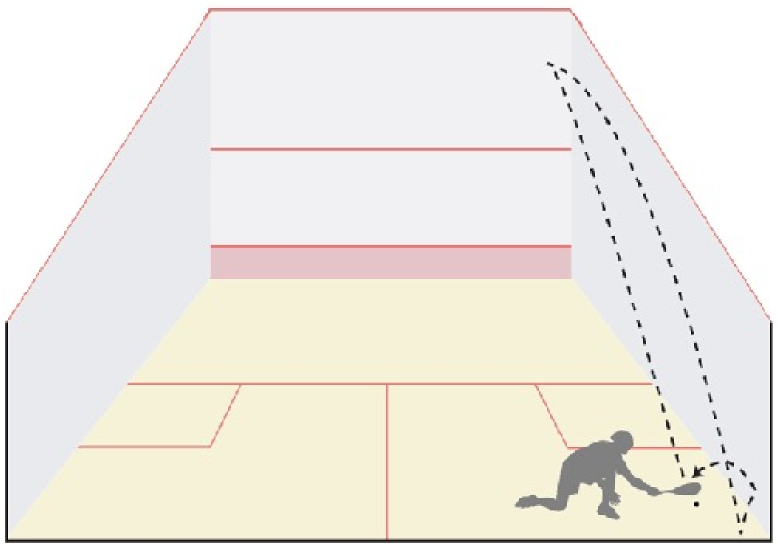


Figure 1.2: Illustration of the solo straight drives exercise (Inside Squash).

2. Related work

In this chapter, the author will discuss the analysis technologies that are available and currently used in both recreational as well as professional squash. In addition, this chapter explores research papers related to the present work and relevant projects that have been developed by independent developers outside of an academic context. The author explains ways of how this research has been applied and how these technologies are being used to improve the experience of players and spectators alike.

2.1 State of sport analytics in squash

The use of sports analytics in the game of squash has not had a long and well-documented history. In most cases, analysis of squash matches has been done by analyzing replays of squash matches, with an emphasis on either understanding a particular opponent's playing patterns or self-analysis of how the player themself reacts in different situations. It is only in recent years that more public-facing analysis of professional play has become available, and still, this analysis is typically performed through human analysis of the game and manual annotation.

According to the best knowledge of the author, external analysis of single-player exercises, such as solo straight drives, is not very common, since it is done by the player at the time of performing the exercise as training. For external analysis, a video is usually recorded and then analyzed afterwards.

However, as new technologies continue to emerge, they are being increasingly incorporated into the analysis of squash games, providing more sophisticated and comprehensive ways for spectators to examine and for players to improve in the sport. In the following sections a general overview of such technologies is given.

2.1.1 Court augmentation

Launched in 2016, there exists a software-and-hardware "smart-court" solution "InteractiveSQUASH" that can retroactively be fitted to almost any squash court (InteractiveSQUASH [2022]). The InteractiveSQUASH hardware solution consists of:

- a central console
- a camera using infra-red light for ball detection
- a specialized front tin with infra-red sensors for ball detection on the front wall
- cameras for recording and player detection
- a camera using infra-red light for ball detection.

This system provides a racket-and-ball user interface for selecting various applications which are then displayed on the front wall of the court by the projector. These applications include training tools that provide instant visual feedback, such as highlighting the location of ball hits on the front wall or drawing target areas among many others. There also exist adaptations of other sports games

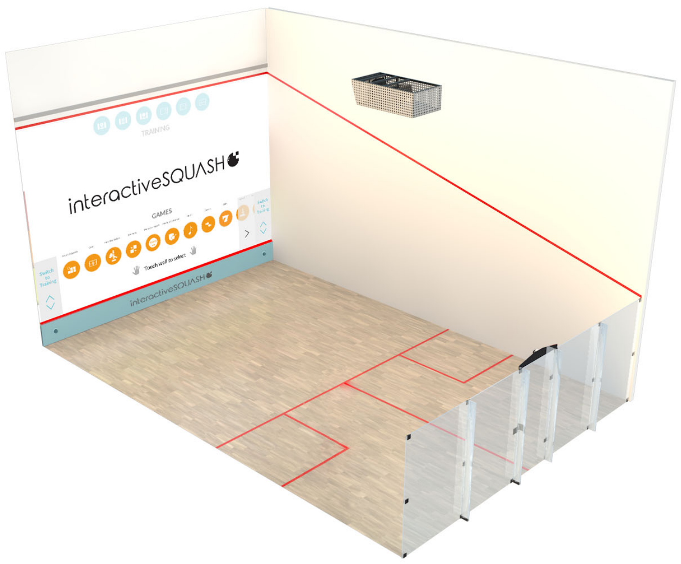


Figure 2.1: Illustration of the InteractiveSQUASH system (InteractiveSQUASH [2022]).

such as darts or even computer games, like Space Invaders (InteractiveSQUASH [2022]). Such games are especially popular for skill development among children. Overall, this system provides a versatile and engaging platform for users to improve their squash skills and have fun at the same time.

Recognizing the push for integrating technology into sports, the Professional Squash Association has been collaborating with InteractiveSQUASH since 2018 for developing the "MoTrack" system to showcase statistics such as player movement speed and positional heat-maps for added spectator entertainment during high-profile events (Professional Squash Association [2018]). While such a hardware-software system is flexible and extensible in its usage of software, the disadvantage of such a specialized system is its high cost.

2.1.2 Wearables

According to (Mencarini et al. [2019]) the use of wearable analytics tools in sports can be divided into two main categories: those that are related to physical performance and those that are related to the acquisition of motor skills. In squash, performance-related tools such as heart-rate monitors and smartwatches are commonly used to track physiological metrics like the intensity of the exercise, the number of steps, and the duration of exercise. In contrast, tools related to motor skills acquisition in squash offer detailed insights into the technical aspects of the game, such as the swing mechanics of a striker. Currently, there exists just a single device for squash that is capable of providing such data: the Racketware sensor. Racketware produces small battery-powered sensors, that are attachable to squash rackets and sample motion at 600 times per second. Using the raw sampled data and machine learning algorithms, it can give the user high-level statistics such as points won and lost or provide data on the intricacies of the user's squash swing with live feedback through its mobile app (Racketware [2022]).

2.1.3 Human analysis

More generally, squash analysis is performed through manual annotation, with the most prominent example being York-based Cross-Court Analytics, which provides in-depth analysis and statistics of squash games. This type of analysis is carried out by human annotators (Dale [2022]). Alternatively, there are also annotation tools specifically designed for squash, such as 'SquashTrack' (SquashTrack [2018]) for the purpose of streamlining the annotating process.

2.2 Related research

Rozumnyi et al. [2017] conducted research on the detection of fast-moving objects in which the study included detecting the movement of squash balls. However, when the methods developed by the authors were applied on their dataset of squash matches, the methods failed in detecting the ball. In their dataset, the ball was white in color, which is uncommon for most squash matches, and was very small when seen from the perspective of the camera. Hrabalik [2017] created a modified version of this algorithm to run in real-time on mobile devices, however, the proposed method in this thesis did not improve the detection of squash balls. It was concluded that the method was unsuccessful due to the size of the ball in the dataset's video. Nevertheless, their methods are still of interest if applied to less complex data.

Work of Judd and Wu [2014] aims to track both the players and the ball in a squash match. They discussed trade-offs between placing the camera behind the players or above the court, ultimately opting for analysis from a top-down view. To implement ball detection, the authors used background subtraction with respect to an input frame to identify an object in the foreground with minimum size and maximum roundness. They also acknowledge the challenge of accurately tracking the ball due to its high speed, which caused it to disappear from frames and use the Kalman filter for ball position estimation. However, they concluded that their method did not yield effective results for obtaining accurate ball location data in the video. The result of their work is being able to provide information on how the players are moving at each point in time.

This thesis mainly builds on the work done by (Sachdeva [2019]), where the author conducted a comprehensive review of related work done in the field of ball detection and tracking and devised an efficient method for tracking a squash ball for machines with low-computation power. The author discusses in detail the methodology and steps for detecting contours, eliminating non-ball contours, and evaluates the system on footage of professional squash matches.

2.3 Community projects

There have been multiple hobby projects that combine computer vision and deep learning approaches to extract court-coverage data from video recordings of squash matches. According to (McKenzie [1993]) the player who controls the center of the court the most is the player in the advantageous position and this metric is therefore of interest in analyzing squash matches.

Both authors tackled this problem and achieved comparable results using deep learning techniques. Dixon used a pre-trained pose estimation model DeepPose to identify joints of the players in the frame, followed by k-means clustering on the output of the model to identify players (Dixon [2018]), whereas Pretto employed the YOLOv3 (Redmon and Farhadi [2018]) algorithm for identifying players (Pretto [2020]). Dixon would then use the joints of the player to accurately map the position of the player based on their feet and Pretto would estimate the position of the player based on the estimated location of the player's torso in the detected contour by his employed detector. Both authors would then use homography to map the movements of the players into 2D space and generate individualized heat-maps of the players' movements. The results are comparable to what is produced by the MoTrack system employed by the PSA.

A system built by (Mullaney [2020]) attempts to take the work done by the previously mentioned authors even further by in an addition to player tracking, incorporating ball tracking and trying to infer points won and lost by the players.

3. Dataset

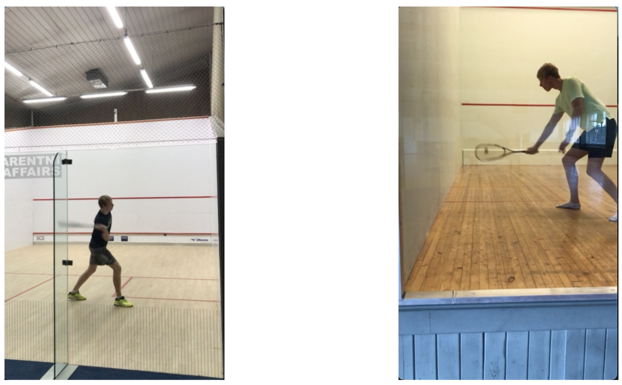
The dataset utilized for the testing of the application was captured by the author of this thesis. The videos were recorded using the personal smartphone of the author, as this was perceived to be the most realistic use case for the application. Both videos were captured using the default settings of the camera and by finding a suitable area in the environment for its placement. For more precise positioning the camera could be mounted on a tripod. Both videos were trimmed using the video editor available on the smartphone to only include footage of the straight drives exercise being performed, with the setup and tear-down of the camera being excluded from the final video.

The initial dataset was recorded using an iPhone 7 prior to the commencement of the application's development and was heavily used throughout the process. The final video was two minutes and three seconds in duration, featuring the author playing a total of sixty-one forehand drives, with sixty bounces visible on camera. The playing area in the video was illuminated by LED lighting.

In order to verify the application's suitability as a general-purpose tool for the analysis of squash drives, a second set of videos was captured as the development of the application neared its conclusion. These videos were captured in a different squash center that featured a large presence of natural sunlight as well as halogen-bulb lighting of the playing area. In the final trimmed recording, ninety-six backhand drive shots were hit, of which ninety-two bounced within the field of view of the camera, over a duration of three minutes and thirteen seconds.

The files were transferred wirelessly to a computer for analysis. For the purpose of speed during development, the video was downscaled to 480p to facilitate faster processing.

In conclusion, the dataset utilized for the testing of the application was collected in a realistic common-use scenario to allow for the accurate testing of the application's performance and to confirm its effectiveness as a tool for the analysis of squash drives.



(a) Frame from the first dataset.

(b) Frame from the second dataset.

Figure 3.1: Example frames from both datasets.

4. Implementation

This chapter presents the theory and implementation of the application. It is divided into two major parts. In the first part, an overview of the theory and steps involved in ball detection, position estimation, tracking, and bounce detection is given. Algorithms at each step are described and their weaknesses and trade-offs are discussed. Followed in the second part of this chapter, where the implementation of the application is described, including the choice of programming language, libraries, and the usability features that make the application user-friendly.

4.1 Constraints on input video

The application has defined a small set of requirements for the input video:

- the camera must be placed behind the player
- the point of contact of the ball with the floor must be visible
- the rear boundary of the court must be visible.

The author deems these requirements as reasonable, as most squash courts have back-walls made of glass for allowing a referee to see inside the court. In cases where the back wall is made of non-see-through material, there is usually an alternate area, such as above the court, with a view compliant with the requirements is visible and a camera can be placed for recording.

The application cannot validate that these requirements are met, and it is up to the user to conform to them for the optimal performance of the application.

4.2 Building blocks of the application pipeline

The core of the application is divided into four categories, each with distinct responsibilities. As a preliminary step, a detector is used for telling apart the foreground of the image from the background of the image. Secondly, an estimator is used for producing hypotheses of the ball's location based on its history. The former and latter data are then combined and given to the tracker from which a selection is made for which contour becomes a part of the tracked trajectory. Lastly, the reliable tracking of the trajectory plays an integral part in producing accurate results with the bounce detector, which at each time step outputs a decision on whether and where a bounce of the ball has occurred.

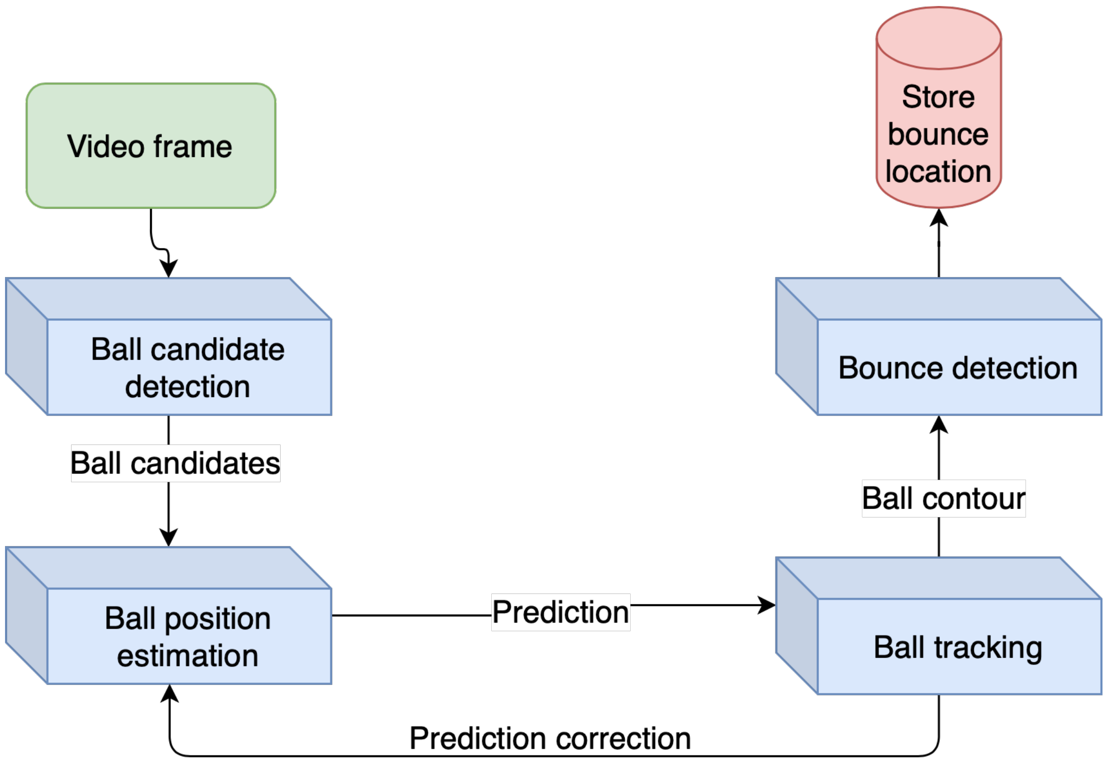


Figure 4.1: Illustration of the algorithmic pipeline of the application.

The following chapters describe the algorithms used and the rationale for accomplishing the steps described above.

4.2.1 Pre-processing and detection

In this thesis, a modified pre-processing method described in (Sachdeva [2019]) is employed in the role of the detector. The main goal of this step is to extract the contour of the ball from a frame. This method involves several steps: first, a window of three consecutive frames is collected and converted from a 3-channel RGB (Red-Green-Blue) representation to a single-channel grayscale image. The images are then smoothed using Gaussian blurring to remove noise from the image, after which the consecutive frames are pair-wise differenced from each other. Those processed frames are then combined using a boolean ”and” operator and then binarized using thresholding. Finally, morphological closing is applied.

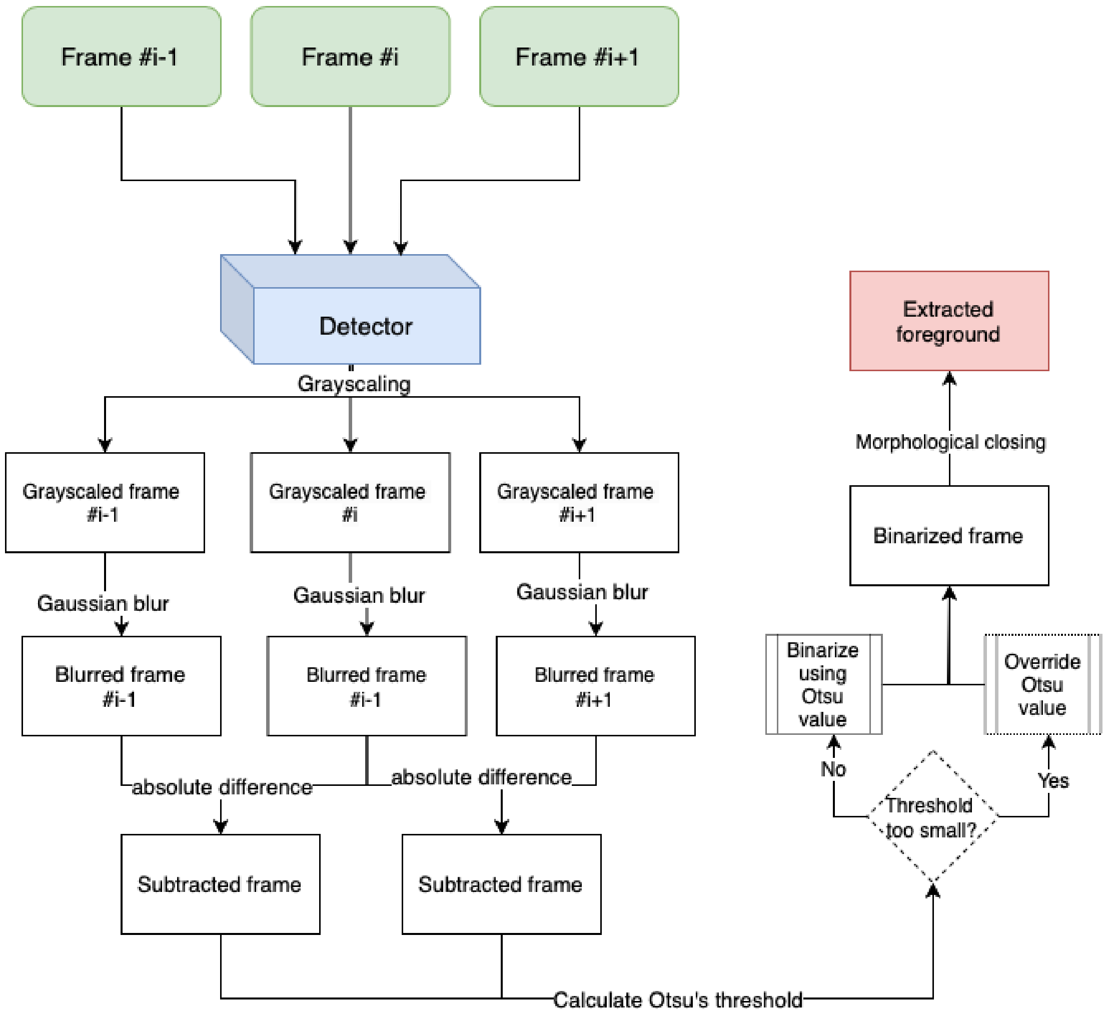


Figure 4.2: Illustration of the detection pipeline.

For Sachdeva, the three-frame differencing approach is motivated by robustness to various sources of noise during competition broadcast videos. These can include various reflections of the glass environment, advertisements, and spectator movements among others (Sachdeva [2019]). However, this approach is also welcome in non-professional videos, as structural elements of the glass back-wall of the court tend to pick up reflections from many external sources and produce visible noise in the video. Formally this approach can be described as follows,

$$D_1 = |I_t - I_{t-1}| \tag{4.1}$$
$$D_2 = |I_t - I_{t+1}| \tag{4.2}$$
$$D = D_1 \wedge D_2 \tag{4.3}$$

where I_t denotes the input image at frame t, D denotes a differential image and the operator $\wedge$ is the common boolean AND operator.

Compared to the original use-case of this pipeline, which was to analyze professional squash matches, the nature of the input data that we are analyzing is more static, and therefore there can occur large class imbalances between foreground and background pixels. Consequently, in this case, the suitability of Otsu's thresholding is not a given according to (Kittler and Illingworth [1985]).

The authors of that paper also propose a solution by modifying the thresholding algorithm.

A simpler approach to address the issue was found for use in this thesis instead. During development when tackling this issue the author observed that telling that thresholding had fundamentally failed in distinguishing the foreground and the background did not in this case require study of the frame's histogram nor seeing the result of the operation, but was distinguishable from a low thresholding value. This hypothesis was confirmed, when testing on the second dataset where the issue was even more pronounced.

The chosen approach was found experimentally and is simple in nature: If the thresholding value as calculated by Otsu's algorithm is small, then this value is discarded and a higher one is used, otherwise use the calculated value.

Algorithm 1: Image binarization algorithm

```
Input: $frame$
Data: $minimum\_otsu\_value \geq 0$
Data: $overriden\_threshold\_value \geq 0$
Result: $binarized\_frame$
$threshold\_value \leftarrow calculate\_otsu\_threshold()$;
if $threshold\_value \leq minimum\_otsu\_value$ then
    $binarized\_frame \leftarrow$
      $binarize\_image(frame, overriden\_threshold\_value)$;
else
    $binarized\_frame \leftarrow binarize\_image(frame, threshold\_value)$;
```

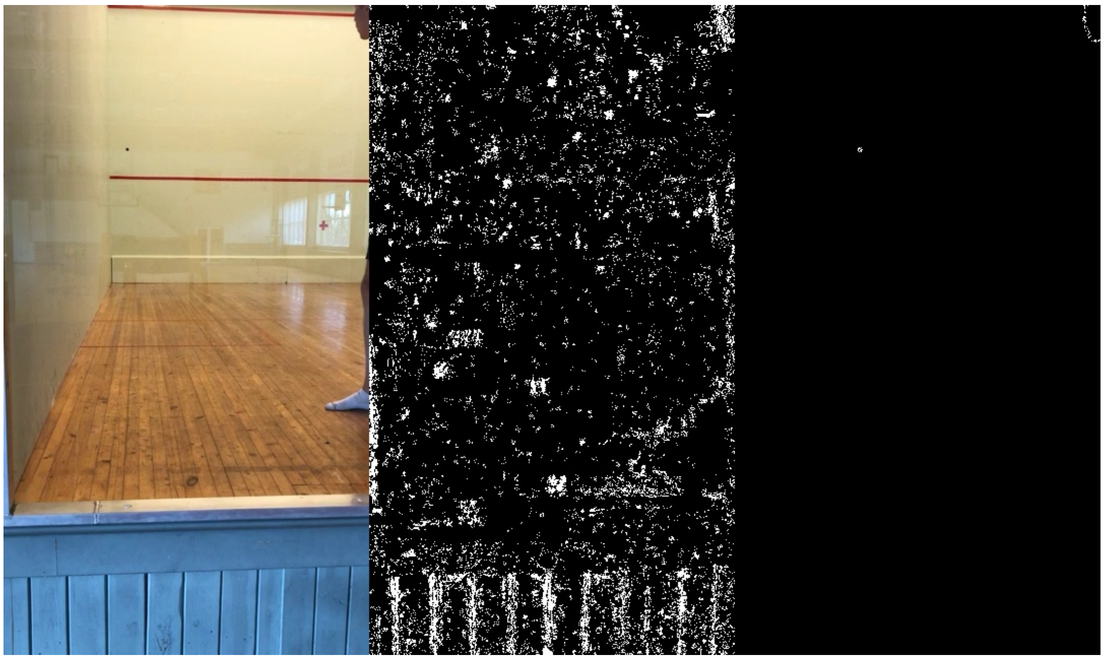


Figure 4.3: Regular frame (left), Otsu thresholded frame (middle), overriden thresholding(right).

Another distinction from Sachdeva's implementation put forward by the nature of the input data is that compared to the dynamic play of professional squash matches, the player when practicing straight drives, tends to be fairly still. As this is not recoverable using the employed frame differencing method, then the contour of the player's body is often fragmented into small pieces. As a remedy, a large number of morphological dilations are used to enlarge the white blobs and close gaps between them, followed by erosion for removing unwanted small objects (Szeliski [2022]).

The ideal result of the described process is the extraction of two contours - the player and the ball. Nevertheless, failure from the ideal is probable in two distinct ways such as not detecting the ball or detecting multiple ball-like contours. Cases, where the ball is not visible, are commonly caused by occlusion by the racket or when there is little movement from the view of the camera. This perception can occur for example when the ball is moving parallel to the view-plane of the camera or the ball is in contact with the front wall and is changing direction. Situations, where there are multiple probable candidates, are more common and occur due to reflections from the environment and poor segmentation of the player contour.

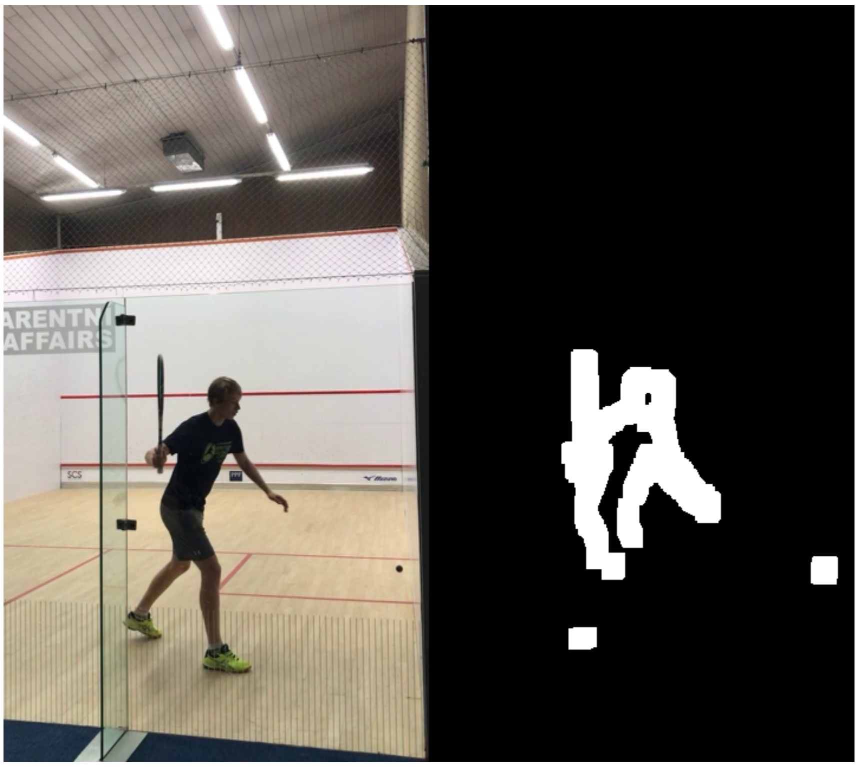


Figure 4.4: Normal frame (left) and extracted foreground from the background with two ball candidates(right).

4.2.2 Ball position estimation

Ball position estimation in this thesis relates to the process of forecasting the position of the ball in the future based on past observations. The presence of such a step in the pipeline is justified as long as the detector is not perfect and consequently is employed precisely as a fallback-mechanism in cases of detector failure, which were briefly analysed in the previous chapter.

The choice of algorithm for position estimation was Holt's double exponential smoothing. A key trait of this forecasting technique is that it accounts for trend in the data (Holt [2004]). It was first applied in the field of ball tracking by Sachdeva and was observed to provide smoother and more accurate trajectories compared to those of the Kalman filter (Sachdeva [2019]). The inclusion of the trend component in the estimation is desirable, as when playing straight drives, hits of the ball sends it traveling towards the front wall followed by the bounce and return of the wall back to the striker, exhibiting periods of trending (see figure 4.5).

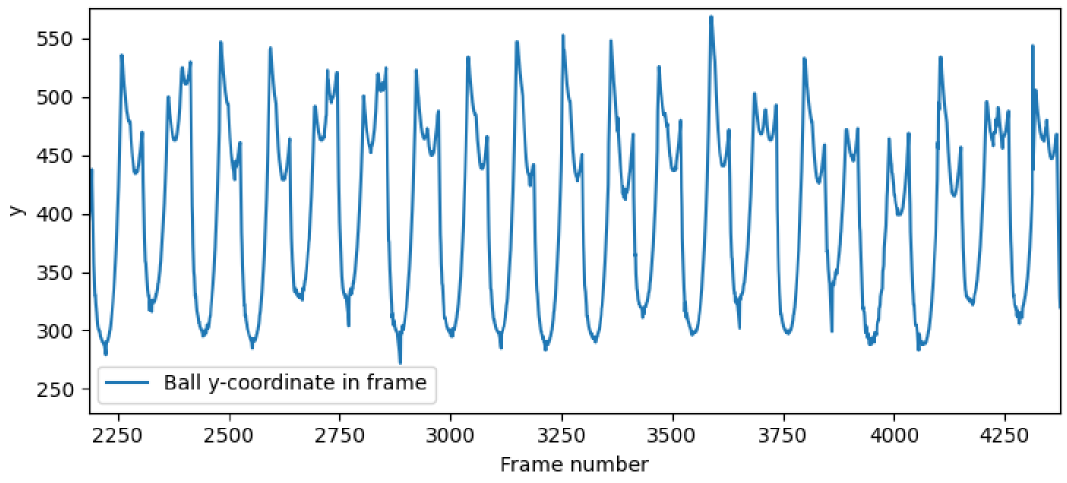


Figure 4.5: Snapshot of ball trajectory from the forehand dataset.

The formulas for performing double exponential smoothing are described below, with S_t signifying the smoothed value and b_t the trend value at time step t. The symbols α and β stand for pre-chosen constants for the data smoothing factor and for the trend smoothing factor in the range [0, 1]. The symbol y_t is a true data-point at time t and y_i is a prediction made for time step i (Sachdeva [2019]).

$$S_t = \alpha * y_t + (1 - \alpha)(S_{t-1} + b_{t-1}) \tag{4.4}$$
$$b_t = \beta(S_t - S_{t-1}) + b_{t-1}(1 - \beta) \tag{4.5}$$
$$y_i = S_t + i * b_t \tag{4.6}$$

4.2.3 Ball contour tracking

Tracking the ball contour is divided into two steps: candidate elimination and candidate selection (see figure 4.6). In short, candidate elimination is performed by joining nearby contours and then filtering them based on their area. The rationale behind this lies in the assumption that nearby contours belong to the same object and once that object is re-created by the join, area-based filtering is more robust against false positive observations. On the other hand, candidate selection relies on the fact that the trajectory of a squash ball is continuous and assumes that observations are pair-wise close to each other. Based on that assumption, a limited history of observations is searched for a most-continuous path, with other heuristics also applied, to label a contour as a ball.

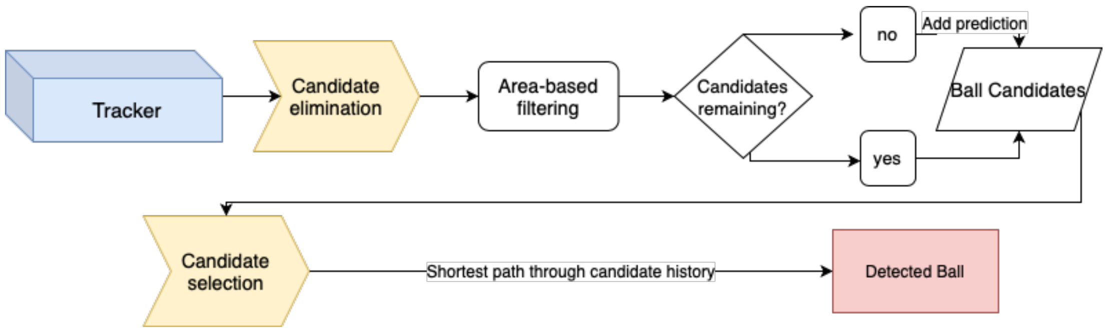


Figure 4.6: Illustration of steps employed in the tracker.

The first step in tracking the ball is minimizing the search-space for the later parts of the tracking algorithm. Effectively the search-space consists of sequences of bounding-rectangles of contours as seen by the detector, with the most common scenario being at each step many false positives and a single true positive detection. The rectangle-join algorithm employed at this step addresses this common case by actively trying to reduce the number of false positives by joining contours that lie within a certain distance threshold from each other (see figure 4.7).

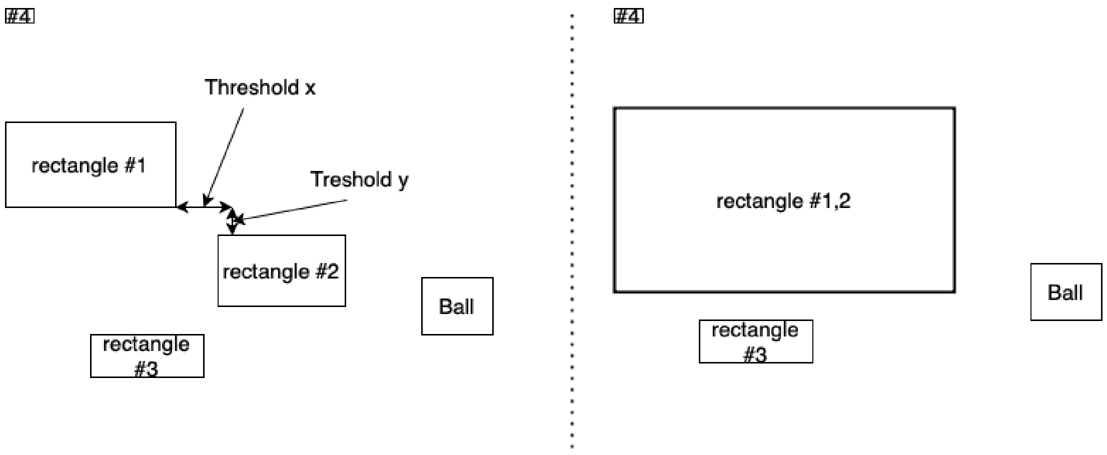


Figure 4.7: Rectangle-join illustration.

While this approach proved to be effective in minimizing the number of possible ball candidates, it also poses the risk of joining the ball with some other sufficiently close contour and removing a true positive from the observations. This drawback is later implicitly addressed in the tracking algorithm. Secondly, the candidate elimination step uses the fact that the size of the ball contour stays mostly consistent in its area. This allows for another cheap operation of discarding candidates with area-based filtering employed on bounding boxes (see figure 4.8). If it so happens that no candidates remain as a result of the filtering then the estimator's predicted contour is used as step-in ball contour (Sachdeva [2019]).

Once the search-space has been reduced, then tracking is based firstly on finding the shortest path across sequences of observations and secondly based on the shape of the observations. Also, it is recognized that this process at times can be working with incomplete information, and if the shortest path exhibits a jump greater than the previous shortest path plus a threshold, then the continuity

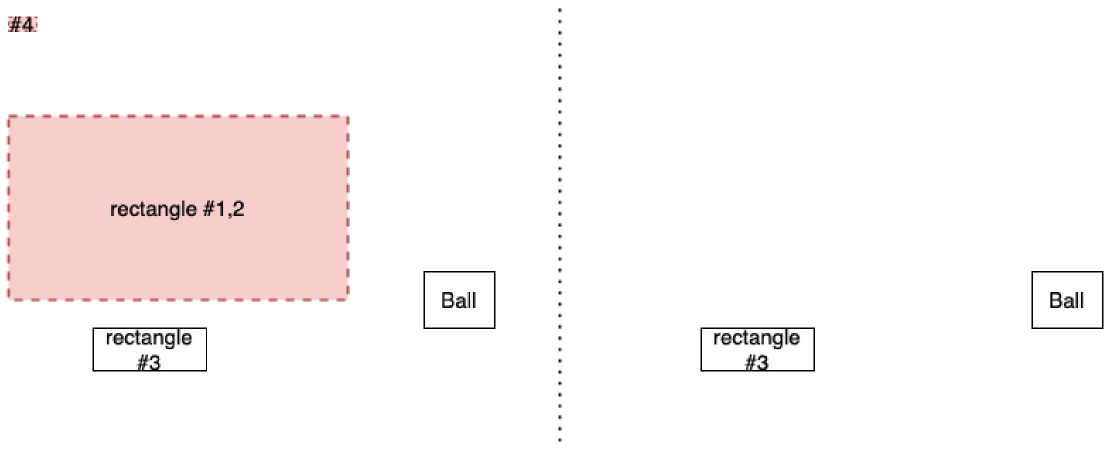


Figure 4.8: Rectangle area based filtering illustration.

assumption is deemed as violated and it is assumed that the true positive is missing. In that case, the prediction is returned instead.

In this process, the length of the observation history and the shape play an important role in the accuracy of the tracking algorithm. The length of the observation history controls the number of layers of observations. A shorter history is subject to more noise as noise is likely to persist for only a few frames, whereas a longer history contains more information and is, therefore, more robust with respect to false positives at the expense of increased computational cost. The shape is considered useful since the resulting contour of the ball after morphological manipulation often resembles a square and provides a useful feature for distinguishing between contours. Thus when traversing through the observations the distance between the observation is scored by the sum of the euclidean distance and the ”square-ness” penalty of the data-point. The ”square-ness” value is used for penalizing non-square contours by inflating the distance score and is obtained by evaluating $|width - height|$. The algorithm is described in algorithm two.

4.2.4 Ball bounce detection

We define the bounce of the ball as the action of the ball rebounding off the floor after its contact with the front wall (see figure 4.5 again for visualization of ball trajectory). For detecting bounces, the patterns exhibited by the ball bouncing are used and for localizing bounces the bounce detector applies homography to map between the 2D image-coordinates and the 2D top-down view coordinates.

Homography, also known as a perspective transform, is a linear transformation that preserves straight lines. The perspective transform mapping is obtained between homogeneous coordinates in the images by calculating $\tilde{x}' = \mathbf{H}\tilde{x}$, where $\mathbf{H}$ is the 3×3 homography matrix. A direct mapping between pixels of the source and destination planes requires normalizing $\tilde{x}'$ according to:

Algorithm 2: Ball tracking via most-continuous path

```
Data: History of contours at each time-step: $history$
Data: Previous shortest distance through $history$:
      $previous\_shortest\_dist$
Data: Distance cut-off $cut\_off$
Data: Prediction contour: $prediction$
Data: All possible shortest links between nodes: $path$
Result: Ball contour
/* iterate over all layers of observations */
for $timestep$ in $history$ do
    /* assume that the true trajectory goes through the given point */
    for observation $i$ in $timestep$ do
        $best\_distance \leftarrow \infty$;
        $nearest\_node \leftarrow None$;
        $squareness \leftarrow |width_i - height_i|$;
        /* find which point in the previous layer is nearest */
        for observation $j$ $at\ previous\ timestep$ do
            $distance \leftarrow \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + squareness$;
            if $distance \leq best\_distance$ then
                $best\_distance = distance$;
                $nearest\_node = j$;
            end
        end
        add ($nearest\_node$, $i$, $best\_distance$) to $path$;
    end
end
$best\_dist, ball\_contour \leftarrow$ find shortest sum through $path$ of same length
  as $history$
/* If the obtained best distance through the history is greater than
   the previous distance + cutoff distance then we assume that a
   failure occurred */
if $best\_dist - cut\_off \geq previous\_shortest\_dist$ then
    /* Inflate the previous shortest distance to avoid getting stuck in
       a loop */
    $previous\_shortest\_dist \leftarrow 1.2 * previous\_shortest\_dist$;
    return $prediction$;
end
$previous\_shortest\_dist \leftarrow best\_dist$;
return $ball\_contour$;
```

$$x' = \frac{\mathbf{H}_{0,0}x + \mathbf{H}_{0,1}y + \mathbf{H}_{0,2}}{\mathbf{H}_{2,0}x + \mathbf{H}_{2,1}y + \mathbf{H}_{2,2}} \tag{4.7}$$

$$y' = \frac{\mathbf{H}_{1,0}x + \mathbf{H}_{1,1}y + \mathbf{H}_{1,2}}{\mathbf{H}_{2,0}x + \mathbf{H}_{2,1}y + \mathbf{H}_{2,2}} \tag{4.8}$$

(Szeliski [2022]).

The homography matrix is calculated using correspondence points between the two images. Fortunately, squash courts contain many intersecting lines, making selecting such points simple in this case. The correspondence points, that we are interested in, are those that encompass the target area when practicing straight drives. More specifically, we would like to capture the upper corners of the service box and the lower boundary of the court (see figure 4.9).

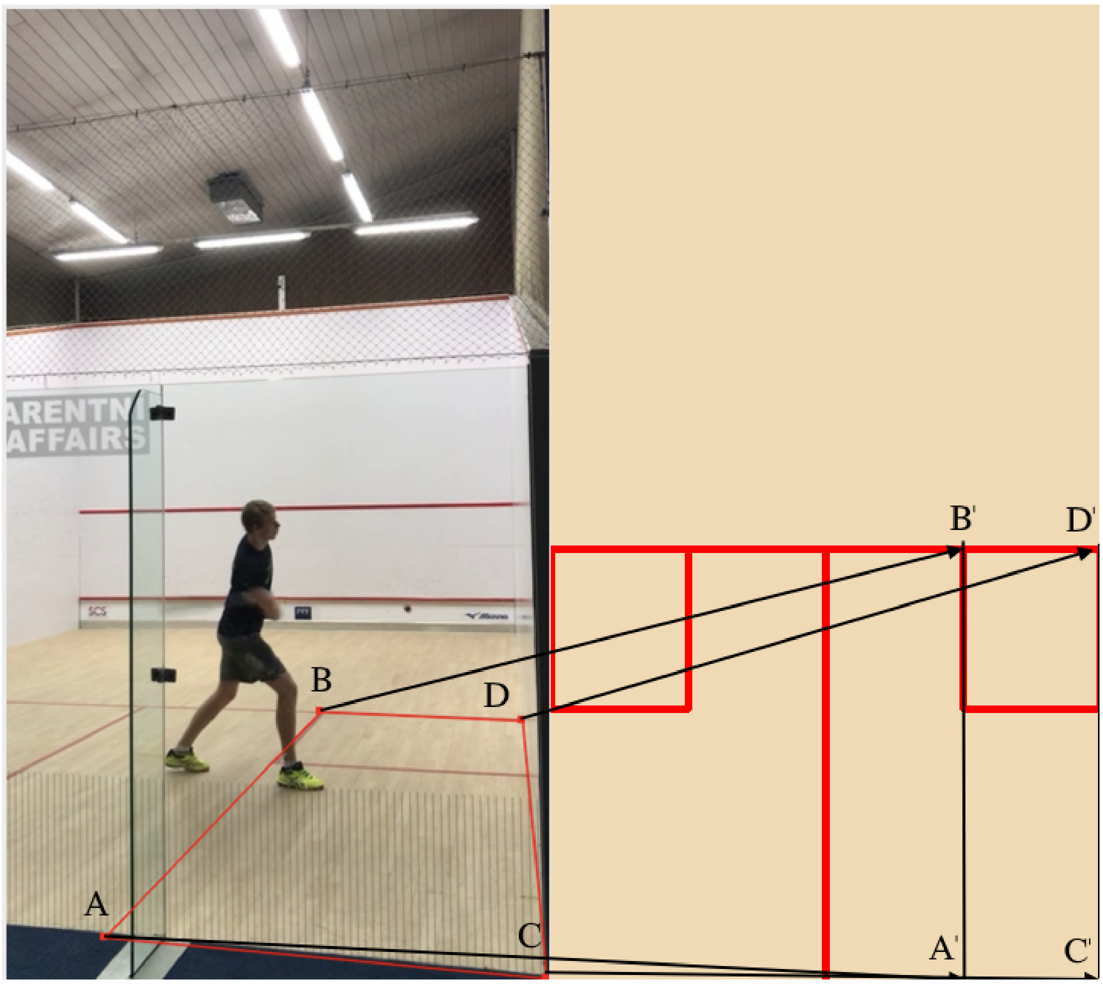


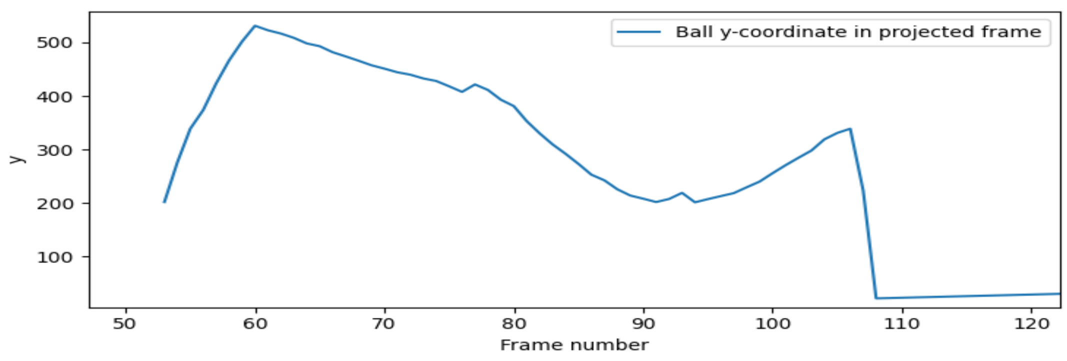
Figure 4.9: Illustration of forehand dataset reference points for homography mapping.

By obtaining the homography matrix and finding the ball contour in the input frame, we gain the ability to calculate the position of the ball in the top-down projection of the court, and thus, localize the bounce positions.

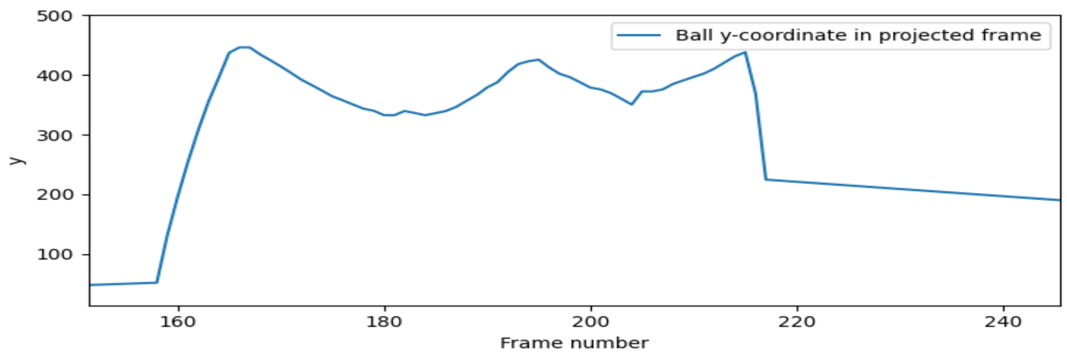
For detecting the occurrences of ball bounces, it is useful to distinguish all three possible trajectory patterns that can occur with a ball bounce and valid return.

1. Ball bounces on the floor and bounces off the back wall or bounces on the floor and is hit back at the front wall (see figure 4.10a).
2. Ball bounces on the floor, experiences a peak, and falls onto the back wall (see figure 4.10b).
3. Ball bounces off the glass wall and then contacts the floor (see figure 4.10c).

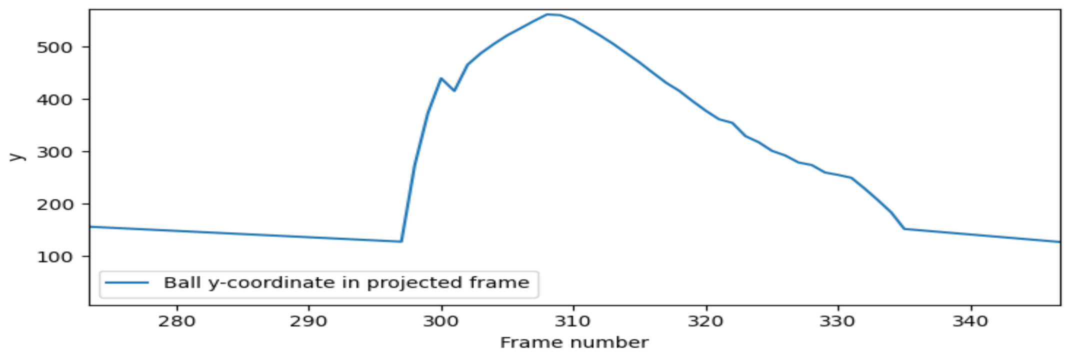
Each of these three cases exhibit a consistent pattern with respect to the peak y-coordinates from the view of the projected top-down frame.



(a) Trajectory when the ball bounces on the floor and bounces off the back wall or bounces on the floor and is hit back at the front wall



(b) Trajectory when the ball bounces on the floor, experiences a peak, and falls onto the back wall.



(c) Trajectory when the ball bounces off the glass wall and then contacts the floor.

Figure 4.10: Example frames from both datasets.

In figure 4.10a we can distinguish two prominent peaks in the y-coordinate. The first peak corresponds to the bounce of the ball off the floor with enough velocity that the ball arcs only following the bounce off the glass wall, drawing the second peak. A similar pattern is exhibited when the ball bounces off the floor and is hit back at the front wall without contacting the back wall. In figure 4.10b the first peak represents the bounce of the ball off the floor, however in this case the ball has smaller velocity, arcs mid-air, and drops onto the back wall, drawing the second peak. The third peak corresponds to the ball dropping towards the floor after contacting the glass fall, before being hit towards the front wall. Figure 4.10c reveals that the bouncing of the ball off the back wall does not manifest from the projection view and in this case only the prominent peak, corresponding to the bounce of the ball off the floor, must be detected.

Through case-by-case analysis of figure 4.10, the issue of bounce detection has been reduced to finding the first peak in the sequence, barring noise, and discarding the following subsequent peaks. The details are discussed in the implementation part of bounce detection (section 4.3.6).

4.3 Implementation of the Application

In this section, various implementation details of the complete application pipeline are discussed more closely (see figure 4.11).

4.3.1 Platform and Programming Languages

This project heavily utilizes the open-source computer vision library OpenCV. OpenCV is written in C++ and offers optimized implementations of various computer vision algorithms, making it a common choice for computer vision projects. Furthermore, OpenCV allows for interoperability with the Python programming language through the use of OpenCV-Python bindings (OpenCV Documentation [2022]). The advantage of using Python over C++ is its more concise, English-like syntax, which simplifies the process of reading and writing programs.

In addition to its user-friendly syntax, Python was also selected due to the abundance of available packages. The graphical user interface was developed using the cross-platform library Tkinter, which is often bundled with Python installations (Python Software Foundation [2022]). Furthermore, popular packages such as MatPlotLib were utilized during development for visualizing and identifying patterns in the data.

Taking all of these factors into account, as well as the author's familiarity with the language, Python was ultimately chosen as the development language of choice for this project.

4.3.2 Setup and calibration

Contrary to initial intuition, in this application, we are not concerned with real-world coordinates of the bounces, but rather we opt for mapping points from the image-plane to a top-down image of the court. The latter view is chosen as it provides the clearest scene for visual analysis of the results.

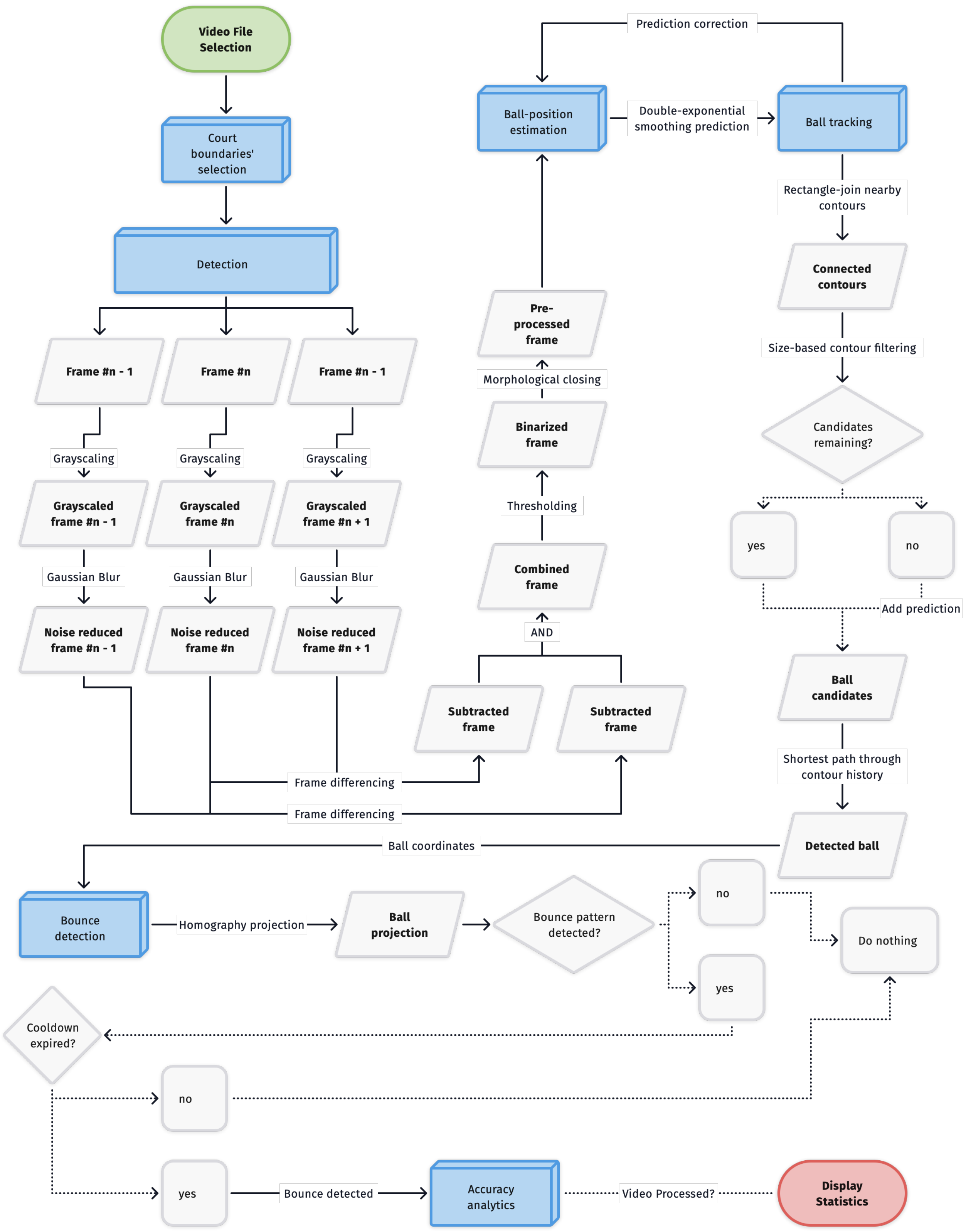


Figure 4.11: Illustration of the full application pipeline.

The research done by Brumann and Kukuk [2022] explores automatic camera calibration based on the known geometry of the squash court. The latter is specified by the World Squash Federation and is an accepted standard around the world. For this, they successfully use genetic algorithms but warn that the effectiveness is related to the quality of the image and the success of edge-detection in preprocessing. They further clarify that whether this automated approach is reasonable to use depends on the use case of the application and that the current technology is more suited for coarse-grained calibration. Since accurate calibration is crucial for this application, the author opted for a manual calibration method.

The manual calibration method is also based on the known geometry of a singles squash court, however, localizing the points is relegated to the human user. The calibration is carried out by clicking on a set of six points:

- 4 corners of the service box
- rear corner of the court
- any point along the rear of the court.

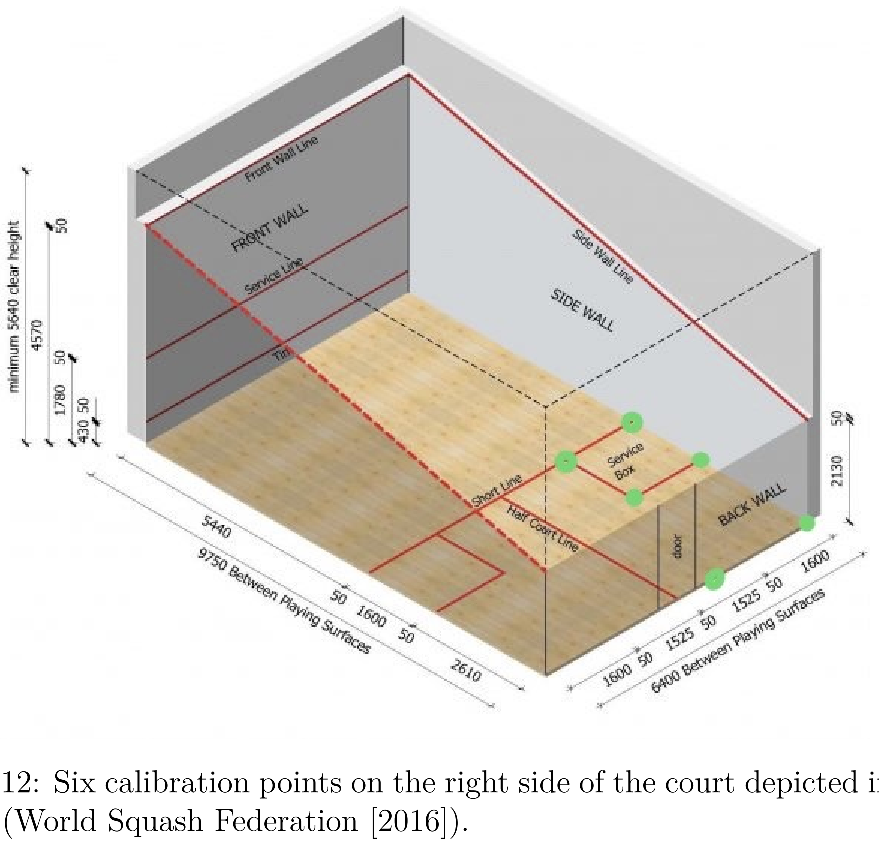


Figure 4.12: Six calibration points on the right side of the court depicted in green marking (World Squash Federation [2016]).

Whereas a homography transformation requires a minimum of four points, the selection of the aforementioned six points is used instead to make the capture of the four points depicted in figure 4.9 more accurate. This approach is used, since the precise clicking of such points is very difficult by hand compared to leveraging the geometry of the court. By capturing the service box and the rear of the court, the selection shown in figure 4.9 is re-created by constructing lines from the selected points and calculating their intersections.

To facilitate accurate clicking, the user interface features a side-by-side view with a magnified view of the cursor position.

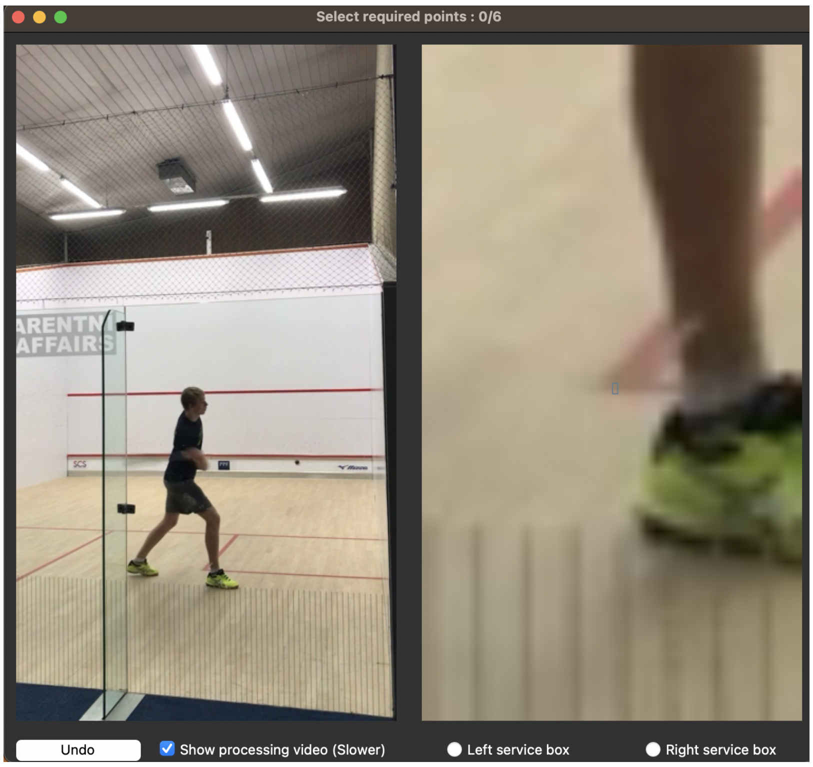


Figure 4.13: Normal view on the left and magnified view on the right.

The order of clicking the points is unimportant as long as the service box geometry is marked first, as the points are sorted and re-assigned based on the assumptions of the input video. The application also tries to infer which side of the court was marked based on the image coordinates of the marked points. If the guess is incorrect then the value is overrideable in the user interface with a button.

4.3.3 Detection

Functionality for extracting the foreground from the image is implemented using the deque data structure from Python's standard library as a frame-buffer. It is a convenient option for such a use-case as it automatically discards the last frame before adding a new one. The extraction steps related to image manipulation are assembled using optimized implementations from the OpenCV library. Kernels for blurring and morphological operations were chosen based on experimental results on the first dataset.

4.3.4 Ball position estimation

In this thesis Holt's double exponential smoothing was implemented as a Python class based on the equations given in section 4.2.2. The class was tweaked for use in this specific application and provided the functionality of estimating the future positions of the bounding-rectangle of the ball. To achieve this, the estimation is performed independently in two variables at the same time, namely for x- and y-coordinates in the image coordinate system. Values for the width and height of the predicted rectangle are not estimated, as it increases computational overhead with negligible effect for a mostly constant-area ball contour. The results from the previous true observation are used for the width and height of the prediction.

Initialization of the estimator is performed during object creation, with the initial true observation artificially placed at the origin of the image coordinate system.

4.3.5 Ball tracking

The tracker is implemented as a standalone class that operates on binary images and returns the rectangle of the hypothesized ball contour. It consists of two main subroutines: one for candidate elimination and one for candidate selection. The methodology for each is discussed section 4.2.3. There are, however, important parameters that were not discussed in detail in the aforementioned chapter.

The candidate elimination routine consists of bounding rectangle joining and area-based filtering, each of which have parameters that play an integral role in the algorithm and lie on the assumptions made on camera placement being behind the player. Firstly, for joining nearby bounding boxes, thresholds must be set on joining distances in both x- and y-coordinates. Since the flight path of the ball is mainly along the y-axis of the image, then avoiding unwanted joins with the ball contour was implemented by providing a more lenient joining threshold on the y-axis. The weakness of such an approach is its coupling with the resolution of the video and an approach employing some clustering algorithm on contours could be used as a more robust alternative. Secondly, area-based filtering is based on a pre-existing constant, which is an estimate of the ball contour's area in the input video.

Values for aforementioned parameters were set during development on the forehand dataset and confirmed to be effective without modification on the backhand dataset.

4.3.6 Ball bounce detection

Ball bounce detection is implemented based on the methodology described in section 4.2.4. The homography matrix is calculated based on the source and target coordinates obtained during application setup and is calculated based on an optimized implementation provided by OpenCV.

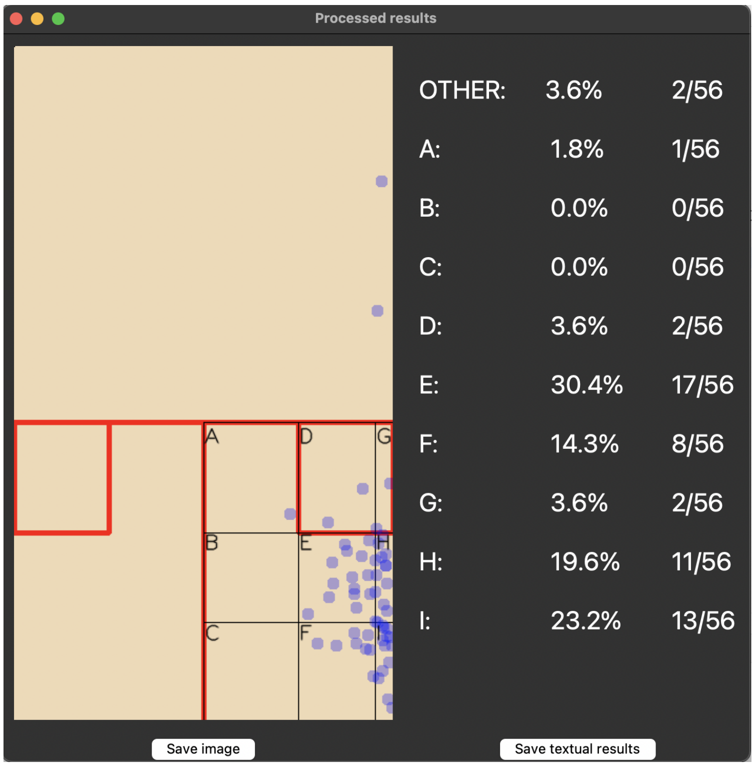
Bounce detection is implemented using a buffer of past observations and a cool-down period. A minimum of three observations are required to detect a peak in the sequence, however such a short history is subject to noise. To counteract this, a buffer of length five was chosen, with the sequence subject to the following condition 4.9 for a bounce detection.

$$x_{i-2} \leq x_{i-1} < x_i > x_{i+1} \geq x_{i+2} \tag{4.9}$$

After condition 4.9 has been met, a bounce is said to have been detected, and the location of the ball is obtained by finding the center of the top-down projected ball contour. After a detection, a hard cool-down period is applied, during which, no detections take place for the purpose of discarding subsequent non-relevant peaks.

4.3.7 Application outputs

The outputs of the application are first and foremost the results of the analysis. These results are presented in a drawn image, where each ball bounce is marked by a semi-transparent circle for providing a heat-map-like visualization, with overlapping bounces looking darker and others looking lighter. Analytics are displayed in a textual format based on the target regions in the image and presents statistics based on bounces within the areas (see figure 4.14).



OTHER:	3.6%	2/56
A:	1.8%	1/56
B:	0.0%	0/56
C:	0.0%	0/56
D:	3.6%	2/56
E:	30.4%	17/56
F:	14.3%	8/56
G:	3.6%	2/56
H:	19.6%	11/56
I:	23.2%	13/56

Figure 4.14: Analysis results of the forehand dataset

During setup the user also is presented with an option to enable or disable the view of of the analysis for visual verification of the program. This view (see figure 4.15) presents the output of the tracker and estimator, by outlining the estimated and tracked ball contour in the video and draws the detected bounces on the court-projection in real time. During processing a progress bar is also displayed.

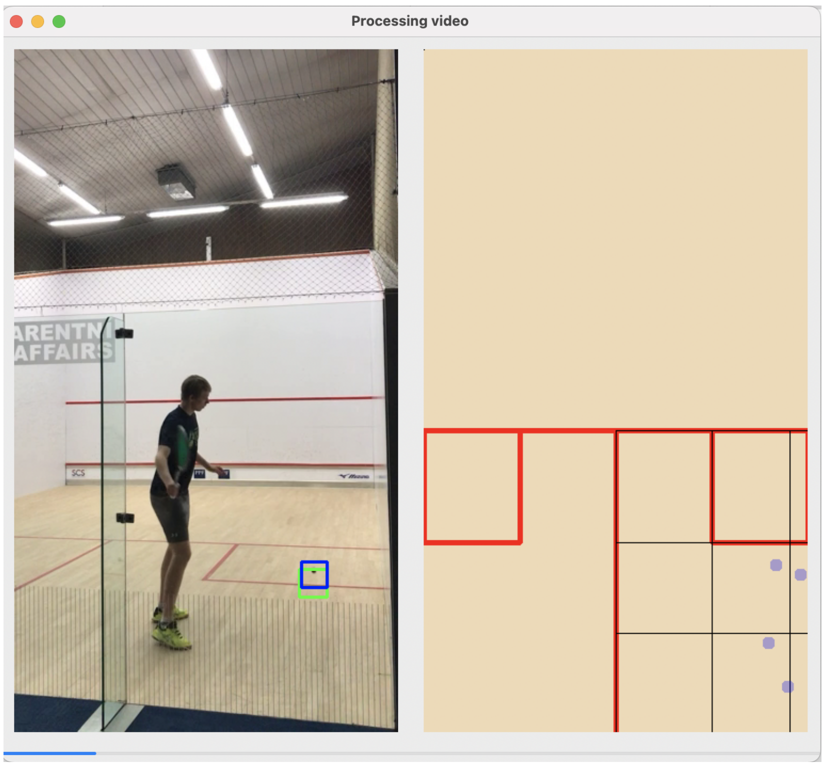


Figure 4.15: Processing with view enabled on the forehand dataset.

5. Evaluation

In this chapter, we look at how the components of the system perform at their tasks in isolation as well as how they perform in unison. A detailed description of the data on which the application was tested on is described in chapter 3. For each part of the evaluated system, the methodology is described individually.

5.1 Obtaining Ground Truth Data

In order to systematically evaluate the performance of the application, it was necessary to annotate the more than 15,000 frames in the captured dataset. Given the large number of annotations required, the author decided to develop a custom annotation script that was suitable for quickly annotating the frames given the nature of the required annotations (illustrated in figure 5.1).

For annotating ball contours in the image the tool utilized the existing building blocks of the application, such as the detector and tracker. The annotation process involved writing the following data for each bounding rectangle of the ball contour into a CSV file:

- number of the frame
- x-coordinate of the top left corner of the bounding rectangle of the contour
- y-coordinate of the top left corner of the bounding rectangle of the contour
- the width of the contour
- the height of the contour.

This data was then used to evaluate the performance of the detector, estimator and tracker.

In performing that annotation process, the author visually confirmed that the tracker had correctly identified the ball contour, and only made corrections when the tracker had made an error and locked onto an incorrect contour, or when the detector failed and the estimator's guess was too broad. This approach allowed for the efficient and sufficiently accurate capture of the necessary ground truth data for evaluating parts of the application.

For evaluating bounce detections, the author used a simplified version of the script and simply annotated the number of the frame in which the bounce was perceived. The annotations do not include only ball bounces from straight drives, but also those instances when the ball was bounced on the floor and thus the number of annotations is greater than the number of drives played.

Since the ground truth data for the location of the ball bounce was too difficult to capture, it was only evaluated visually to be sufficiently accurate.

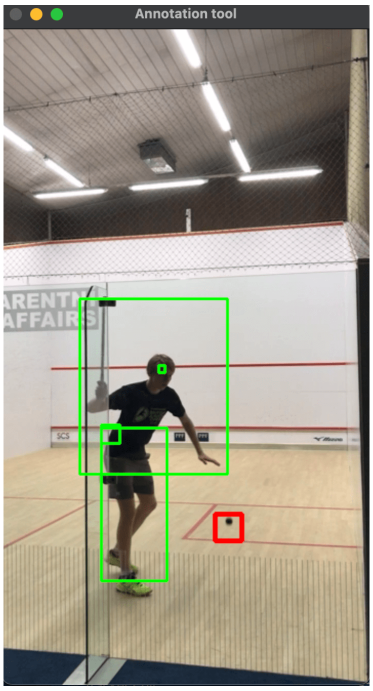


Figure 5.1: Screen-capture of the trajectory annotation script.

5.2 Detector

The effectiveness of the detector is evaluated by assessing its ability to correctly identify the presence of a ball contour in a given frame, as well as its ability to avoid identifying other contours.

The output of the detector was captured by using it to classify regions of the image as the foreground, creating bounding rectangles around detected contours and saving the results into a CSV file in the format as described in section 5.1. The output of the detector was then compared frame-by-frame against the ground-truth annotations using a script written by the author.

Naturally, we define true positives as instances where the detector correctly identifies the location of a ball contour in a frame. For this evaluation we defined the minimum threshold for a true positive overlap at 33%. False positives refer to instances where the detector incorrectly identifies a non-ball contour as a ball contour and false negatives refer to instances where the detector fails to identify the presence of a ball contour in a frame even though one is present.

The results of this evaluation are presented in table 5.1.

Metric	**Forehand dataset**	**Backhand dataset**
True Positives	5418	7036
False Positives	13638	23471
False Negatives	1695	4073
Precision	0.28	0.23
Recall	0.76	0.63
F1-score	0.41	0.34

Table 5.1: Evaluation of the detector on the forehand and backhand datasets.

This analysis reveals that the recall score of the detector on both datasets is mediocre, with the detector failing to identify the presence of a ball contour in roughly 25% to 35% of cases. A poor precision score is expected, as the strategy of the utilized detector is to capture all moving elements in the frame that are subject to filtering later in the pipeline. However, it is worth noting that on that dataset, the false negatives most often occur due to the occlusion of the ball by the racket or when the ball is stationary from the perspective of the camera. The latter case happens most when the ball contacts the front wall and bounces back.

The evaluation metrics could be improved by increasing the history of the background or by using a different strategy and dividing detection into multiple steps and using a combination of detectors during the detection step at the expense of computational cost.

5.3 Estimator

The effectiveness of the estimator was evaluated by assessing its ability to predict the location of the ball in subsequent frames. This evaluation was performed in two ways: by measuring whether the prediction overlapped with the annotated contour and by calculating the average distance between the center of mass of the predictions and the annotations.

The estimator was assessed using the annotated data of ball positions in frames. The contours of the ball were input to the estimator, and the estimator was then used to predict the future location of the ball. After making a prediction, the internal state of the estimator was updated using the annotated ground truth contour. The evaluator was run in the same way that it would be used in the application, simulating predictions of the ball's location, assuming that the detector has not detected the contour for one, two, and three consecutive frames.

The output of the detector was compared to the ground truth annotations using a Python script written by the author. In this comparison, true positives were defined as contours that overlapped between the prediction and the annotation, and false positives were defined as contours that did not overlap. False negatives were not considered in this evaluation, as they do not exist in the annotations. In addition, the mean distance between the predicted position of the ball and the annotation was measured in pixels to gauge the accuracy of estimation. A lower mean distance indicates better performance than a higher distance, with a score of zero indicating a perfect score in following the annotated path of the ball. The letter "T" in the table 5.2 below refers to the timestep used to predict the number of frames in the future.

The table shows that the double-exponential estimator is relatively accurate when tasked with predicting just a single frame ahead, however, the table presents a clear trend of dropping precision as predictions are made for future time steps. Higher accuracy could be achieved by a systematic search of the trend- and smoothing parameter values that would maximize precision, however in the context of this thesis that would risk over-fitting the estimator for the example dataset given the amount of testing data.

Dataset	T	TP	FP	Precision	Mean Distance
Forehand	1	6350	423	0.94	10.52
Forehand	2	6035	739	0.89	14.48
Forehand	3	5705	1070	0.84	18.89
Backhand	1	9552	1134	0.89	17.83
Backhand	2	8926	1761	0.84	23.24
Backhand	3	8297	2391	0.78	29.57

Table 5.2: Evaluation of the double-exponential estimator on the forehand and backhand datasets.

5.4 Tracker

In this work, the tracker is inherently linked to the detector and estimator, and as such, there was no optimal way to test the tracker in isolation. Therefore, this evaluation aims to assess the ability of the tracker to utilize imperfect detections and ball position estimations in order to generate a coherent trajectory of the ball's movement. This directly evaluates the tracking performance in a common use-case scenario. The trajectories were generated by providing the tracker with the contours produced by the detector and double-exponential estimator. The output was stored in a CSV file in the common format outlined in the introductory section of this chapter. These simulated data were then compared to annotations, with a true positive instance requiring 50% overlap with an annotated contour and false positives and false negatives defined in the same way as in the detector evaluation. The results are presented in table 5.3.

Dataset	TP	FP	FN	Prec.	Recall	F1	Mean Distance
Forehand	4835	1760	232	0.73	0.95	0.83	14.26
Backhand	8349	2530	208	0.77	0.98	0.86	16.03

Table 5.3: Evaluation of the tracker on the forehand and backhand datasets.

Given that the detector produces a lot of hypotheses of ball positions, then the tracker is able to consolidate them with the predictions of the estimator quite well, achieving F1-scores over 0.90 on both datasets. Given that false positives are mostly introduced when the failure of the detector and failure of the estimator coincide, improvements in either would in turn improve the tracker.

5.5 Bounce Detection

The bounce detection used in this thesis has two aspects that should be quantified: the accuracy of detecting bounces and the accuracy of localizing the bounce. As mentioned in the introduction of this chapter, the author did not manage to obtain objective ground-truth data on bounce locations and only visually inspected the localization to be accurate. Therefore, only the ability to detect bounces of the ball was evaluated.

For obtaining the simulated data, the author initialized the detector with the appropriate source and destination coordinates for each video in the dataset and proceeded with the simulation by applying the detector on the annotated ball trajectories. The output of the simulation was a CSV file that contained numbers of each frame when a bounce was detected, which was then compared against the annotations. Since the accuracy of the annotations can slightly vary due to the attention of the annotator, then the results were evaluated with a window of error of 10 frames, meaning that the bounce detected by the detector must be within 10 frames of the annotated bounce to be counted as a true positive. Bounces that occurred outside the field of view of the camera are represented as true negatives.

Dataset	TP	FP	FN	TN	Accuracy	Precision	Recall	F1
Forehand	56	4	4	1	0.85	0.93	0.90	0.92
Backhand	78	4	14	4	0.81	0.95	0.85	0.90

Table 5.4: Evaluation of the bounce detector on the forehand and backhand datasets.

In conclusion, even though the bounce detector achieved less than ideal F1-scores, the results provided are usable due to relatively good precision values. While the detector was designed to detect the bounce pattern of straight drives and would expectedly miss consecutive ball bounces between the racket and ball, the number of missed detections on the backhand dataset suggests that the detector might need further calibration.

5.6 Application Performance

For testing the complete application pipeline, the program was run on both videos in the dataset with the processing view enabled, during which, the bounce detections were visually analyzed. All bounces that were detected and localized correctly were marked as true positives, whereas bounces not visible were instances of true negatives. Cases deemed false positives were instances, when the ball did not bounce, yet a bounce was detected and false negatives, when the ball bounced, but was not detected as such. In cases where the ball bounced, but localization was wrong, both false positives and false negatives were incremented.

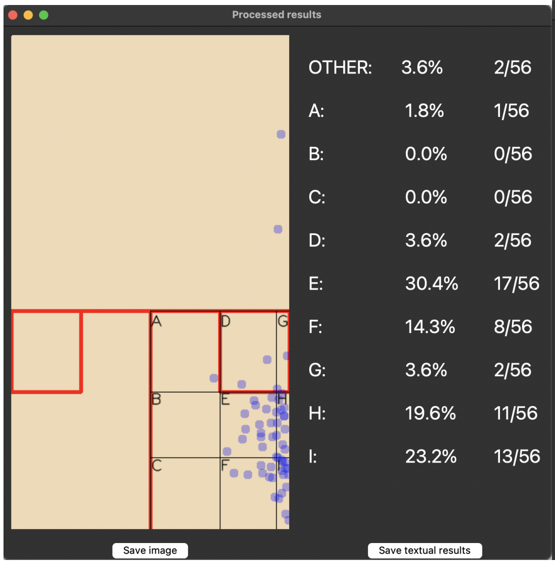
The results of this analysis is presented in table 5.5.

Dataset	TP	FP	FN	TN	Accuracy	Precision	Recall	F1
Forehand	51	4	9	1	0.80	0.93	0.85	0.89
Backhand	79	6	13	4	0.81	0.93	0.86	0.89

Table 5.5: Evaluation of the bounce detector on the forehand and backhand datasets.

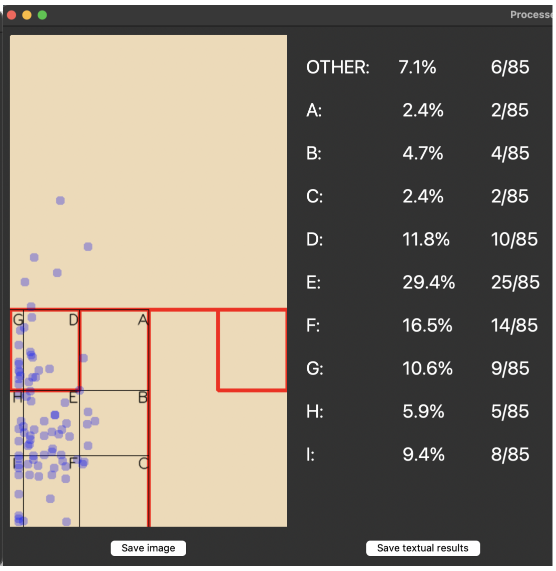
Processing the downscaled 480p forehand drives video on a 2,3 GHz Quad-Core Intel Core i5 (8259U) took 53 seconds. The longer backhand drives 480p video of three minutes and thirteen seconds took a total of 1 minute and 10 seconds. Enabling the visualization of the analysis took 8 minutes and 23 seconds and 12 minutes and 3 seconds respectively.

The output views of the applications are presented in figure 5.2.



OTHER:	3.6%	2/56
A:	1.8%	1/56
B:	0.0%	0/56
C:	0.0%	0/56
D:	3.6%	2/56
E:	30.4%	17/56
F:	14.3%	8/56
G:	3.6%	2/56
H:	19.6%	11/56
I:	23.2%	13/56

(a) Application output on the forehand dataset



OTHER:	7.1%	6/85
A:	2.4%	2/85
B:	4.7%	4/85
C:	2.4%	2/85
D:	11.8%	10/85
E:	29.4%	25/85
F:	16.5%	14/85
G:	10.6%	9/85
H:	5.9%	5/85
I:	9.4%	8/85

(b) Application output on the backhand dataset

Figure 5.2: Analysis results

Ultimately, the author evaluated that the application pipeline performs sufficiently well in giving a general overview of the number of shots hit and the accuracy of the striker in a reasonable analysis time.

Conclusion

In this thesis, we set out to create a computer application capable of analyzing amateur videos of a player performing straight drives and automatically extracting and visualizing data of their performance. We began this work with a brief introduction to squash, an explanation of the terminology involved, and the justification for the importance of straight drives as a subject of analysis. We then presented the existing solutions for squash analytics as well as related academic literature to understand the larger context in which this work is positioned. When designing the application, with accessibility in mind, we provided a set of restrictions on the input videos, such as positioning of the camera behind the player, which users should adhere to for optimal performance. An existing theoretical approach to tracking a squash ball, developed for broadcast-grade quality match videos, is implemented and modified at various steps. For the detection of a squash ball, a simple solution for cases of poor image segmentation due to the static nature of the data is proposed. On evaluation, the detector obtained a recall score of 74% on the development dataset and 63% on the testing dataset. For tracking a squash ball, a trajectory-filling algorithm was devised that achieved a high-recall and a precision of roughly 75%. For detecting and localizing bounces of the ball, a perspective transform is used to map the ball position between the view of the camera and the top-down projection of the court. The patterns of the ball movement were then analyzed and exploited for detecting bounces. The latter method achieved precision scores in excess of 90% and recall scores of 90% and 85% on annotated data from the development and testing videos.

The result of this work is a unique cross-platform application in the field of squash analysis, that within the constraints of camera placement, achieved precision scores in excess of 90% and F1-scores of 89% on bounce detections on both the video used in development, as well as for testing the application. Thus, this application can be evaluated as reasonably effective in providing an objective lens for evaluating straight drives accuracy automatically.

Bibliography

C. Brumann and M. Kukuk. Evolution Based Single Camera Resectioning Based on Distance Maps of a Known Geometry for Squash Sports. *IEEE Access*, 10: 1–1, 01 2022. doi: 10.1109/ACCESS.2022.3178832.

M. Dale. Cross Court Analytics: Players, coaches and federations flock to learn the secrets of their winners and errors, October 2022. URL `https://squashmad.com/breaking-news/cross-court-analytics-players-coaches-and-federations-flock-to-learn-the-sec` (Accessed: 2022-11-30).

P. Dixon. Player Tracking in Squash with Computer Vision and Deep Learning, June 2018. URL `https://parkerdixon.github.io/Squash-Vision/`. (Accessed: 2022-04-04).

C. C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2003.09.015. URL `https://www.sciencedirect.com/science/article/pii/S0169207003001134`.

A. Hrabalik. Implementing and Applying Fast Moving Object Detection on Mobile Devices. Master's thesis, Czech Technical University in Prague, 2017. URL `http://hdl.handle.net/10467/69501`.

Inside Squash. URL `http://www.insidesquash.com/drill_images/123.gif`. (Accessed: 2010-09-20).

InteractiveSQUASH. How the System works: Hard- and Software, 2022. URL `https://interactivesquash.com/operator/`. (Accessed: 2022-12-11).

T. W. Jones, B. K. Williams, C. Kilgallen, C. Horobeanu, B. C. Shillabeer, A. Murray, and M. Cardinale. A review of the performance requirements of squash. *International Journal of Sports Science & Coaching*, 13(6):1223–1232, 2018. doi: 10.1177/1747954118792492. URL `https://doi.org/10.1177/1747954118792492`.

D. Judd and R. Wu. Squash Sport Analytics & Image Processing. 2014. URL `https://www.cis.upenn.edu/wp-content/uploads/2019/08/HonorspaperforEAS499_Wu_Judd.pdf`.

J. Kittler and J. Illingworth. On threshold selection using clustering criteria. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(5):652–655, 1985. doi: 10.1109/TSMC.1985.6313443.

I. McKenzie. *The Squash Workshop:*. The Crowood Press, Ltd., 1993. ISBN 9781852237288.

E. Mencarini, A. Rapp, L. Tirabeni, and M. Zancanaro. Designing Wearable Systems for Sports: A Review of Trends and Opportunities in Human–Computer Interaction. *IEEE Transactions on Human-Machine Systems*, 49(4):314–325, 2019. doi: 10.1109/THMS.2019.2919702.

T. Mullaney. SquashAI: Video Analytics for Squash, 2020. URL `http://tommymullaney.com/projects/squash-ai`. (Accessed: 2022-11-30).

OpenCV Documentation. How OpenCV-Python bindings are generated?, September 2022. URL `https://docs.opencv.org/4.x/da/d49/tutorial_py_bindings_basics.html`. (Accessed: 2022-09-26).

F. Pretto. Squash Analytics: a Computer Vision and Deep Learning approach, 2020. URL `https://tealfeed.com/squash-analytics-computer-vision-deep-learning-nthuv`. (Accessed: 2022-04-04).

Professional Squash Association. PSA To Launch Real-Time Statistics Tracking System With interactiveSQUASH, January 2018. URL `https://www.psaworldtour.com/news/psa-to-launch-real-time-statistics-tracking-system-with-interactivesquash`. (Accessed: 2022-11-30).

Professional Squash Association. Men's world rankings, August 2022. URL `https://www.psaworldtour.com/rankings/`. (Accessed: 2023-01-03).

Python Software Foundation. Graphical User Interfaces with Tk, September 2022. URL `https://docs.python.org/3/library/tk.html`. (Accessed: 2022-09-06).

Racketware. World's only motion tracking sensor for squash, November 2022. URL `https://www.racketware.co.uk`. (Accessed: 2022-09-19).

J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement, 2018. URL `https://arxiv.org/abs/1804.02767`.

D. Rozumnyi, J. Kotera, F. Sroubek, L. Novotny, and J. Matas. The World of Fast Moving Objects. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jul 2017. doi: 10.1109/cvpr.2017.514. URL `https://doi.org/10.1109%2Fcvpr.2017.514`.

S. Sachdeva. Detection and Tracking of a Fast-Moving Object in Squash using a Low-Cost Approach. Master's thesis, Delft University of Technology, Delft, NL, 2019.

N. Santelmann. Ten Healthiest Sports. *Forbes*, 2003. URL `https://www.forbes.com/2003/09/30/cx_ns_1001featslide.html`. (Accessed: 2022-08-31).

SquashTrack. SquashTrack Makes Your Coaching Easier, 2018. URL `https://www.squashtrack.com`. (Accessed: 2022-12-01).

R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer International Publishing, 2022. ISBN 9783030343712. doi: https://doi.org/10.1007/978-3-030-34372-9.

US Squash. Squash Facts. URL `https://web.archive.org/web/20200823074254/https://www.ussquash.com/squash-facts/`. (Accessed: 2022-11-30).

G. Vučković, N. James, M. Hughes, S. Murray, G. Sporiš, and J. Perš. The effect of court location and available time on the tactical shot selection of elite squash players. *Journal of Sports Science and Medicine*, 12:69, March 2013.

B. K. Williams, P. C. Bourdon, P. Graham-Smith, and P. J. Sinclair. Validation of the Hunt Squash Accuracy Test used to assess individual shot performance. *Movement & Sport Sciences - Science & Motricité*, 100, 2017.

World Squash Federation. Specification for Squash Courts, January 2016. URL `https://www.worldsquash.org/wp-content/uploads/2021/08/171128_Court-Specifications.pdf`. (Accessed: 2022-08-31).

List of Figures

List of Tables

List of Abbreviations

PSA	Professional Squash Association
TP	True positive
FP	False positive
FN	False negative
TN	True negative

A. Attachments

A.1 Source code

The application source code. The source code is also available on the URL: https://github.com/veedlaw/squash-drive-analyst.

A.2 Video data

Contains the videos described in the dataset chapter.

A.3 Annotation and evaluation source code

Contains annotation scripts (.py), annotation and simulation data (.csv) and Jupyter notebooks (.ipynb) used for evaluation as mentioned in section 5.1.

A.4 User documentation

The software allows the user to load a previously captured video run the analysis program on the video. Upon completion of the analysis, the user will receive:

- A heat-map image of the ball bounce locations.
- Ball bounce location in percentages and absolute numbers with respect to target boxes.

A.4.1 Pre-requisite

Download and install Python3

A.4.2 How to launch the application

1. Download this repository and navigate to it:

```
cd squash_drive_analyst
```

2. Create a virtual environment:

```
python3 -m venv venv
source venv/bin/activate
```

3. Install the required packages:

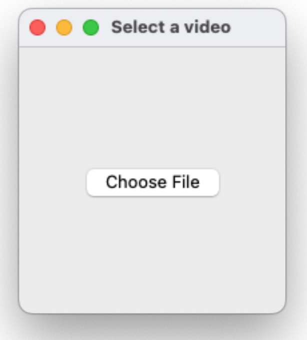
```
pip3 install -r requirements.txt
```

4. Launch using:

```
python3 main.py
```

A.4.3 Video selection

The program starts with a file selection screen which allows to navigate the file system and select a video to run the program on.

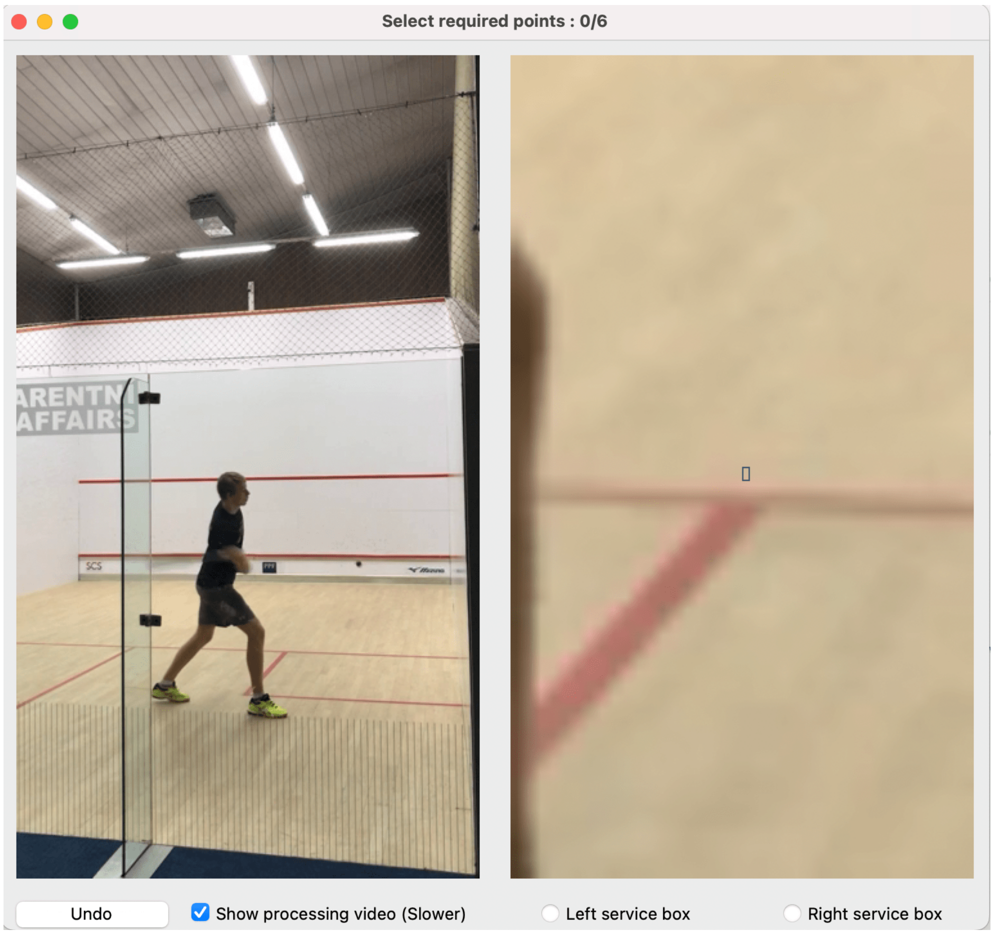


Tip: A lower resolution video can be processed faster.

A.4.4 Setting up

The set-up screen features the following elements:

- A preview frame on the left to mark the service box and the lower boundary of the court
- A zoomed-in view of the mouse cursor on the right
- An Undo button to undo last marker placement
- A checkbox that selects whether the video will be displayed during analysis or not
- Two radio-buttons marking on which side of the court the service box lies

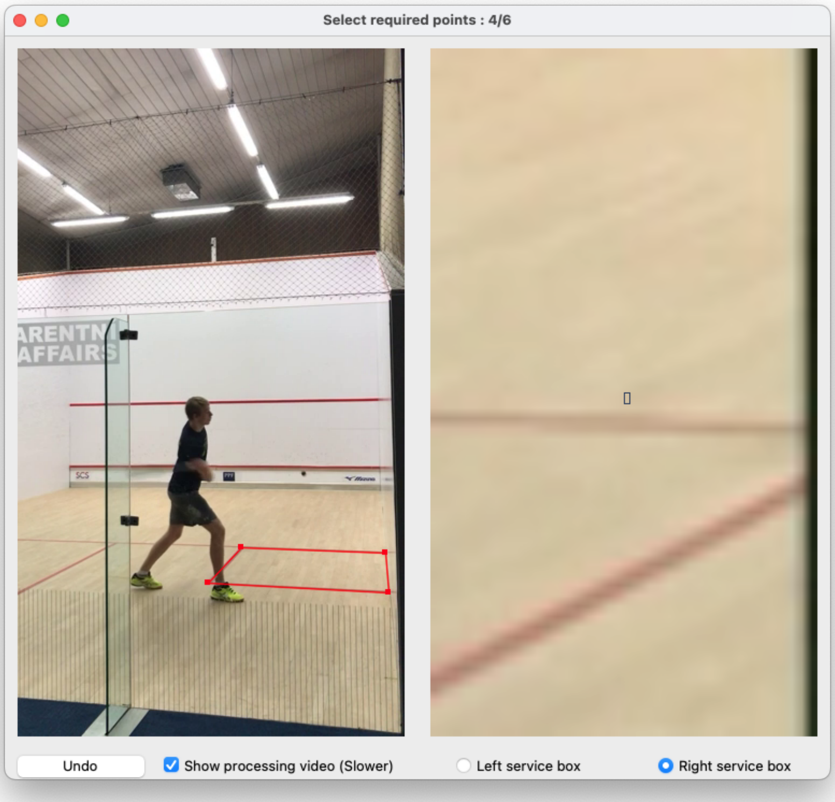


The set-up requires that first the four corners of the service box be marked and then the two remaining points lying on the lower boundary of the court from the view of the camera.

The application will try to infer on which side of the court the marked service box lies on and automatically selects the corresponding radio-button. If the inference is incorrect then it can be manually overridden by selecting the opposite radio-button.
Tip: Uncheck the ”Show processing video” checkbox for the analysis to run much faster.

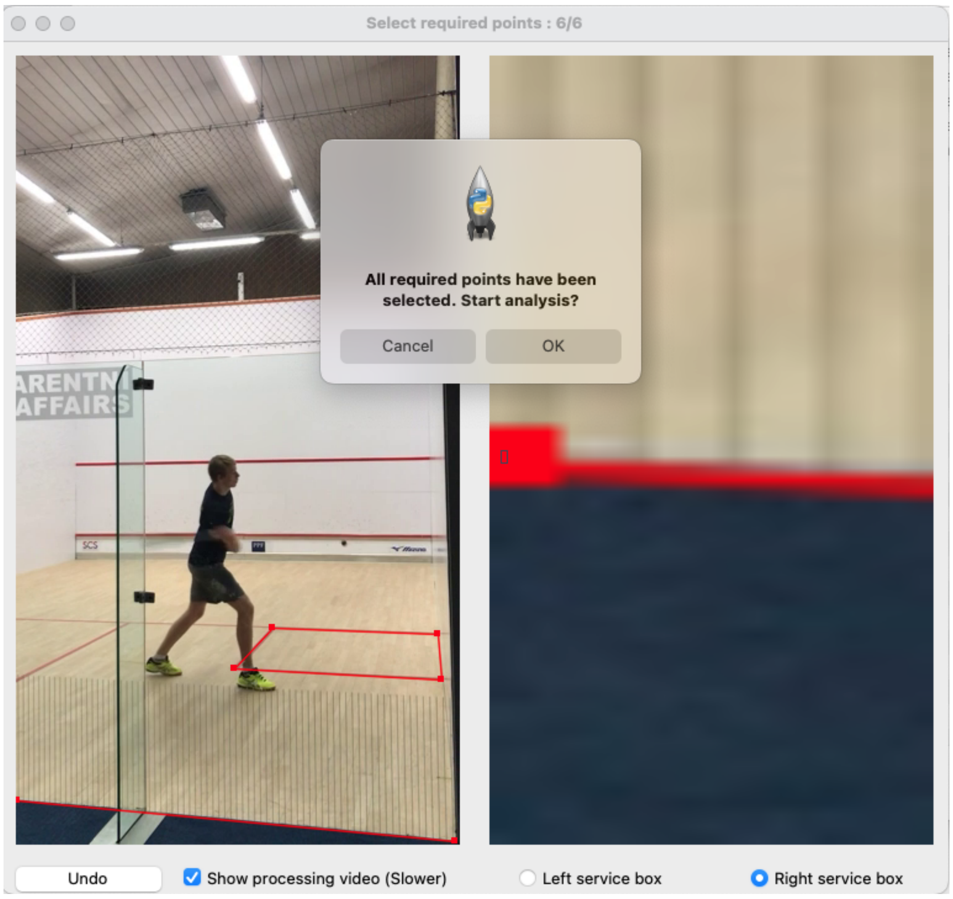
Marking the service box

It is advisable to aim for the outer edges of the corners of the service box. A visual overlay is automatically drawn that connects the clicked points. The application can be seen to have correctly inferred that the service box lies on the right side of the court by the active radio-button in the lower-right corner of the application.



Marking the rear of the court

Next any two points must be clicked on that lie on the boundary of the playing area and the floor. In the example image the points have been chosen as far apart as possible for visualization purposes.


Select required points : 6/6

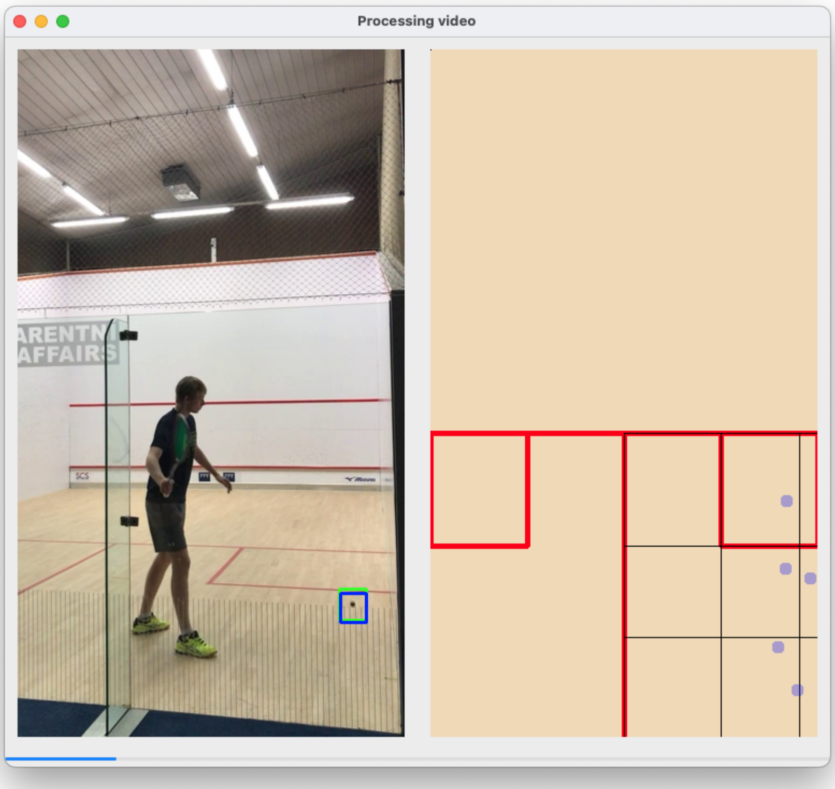
All required points have been selected. Start analysis?

[✓] Show processing video (Slower)

Left service box [✓ Right service box]

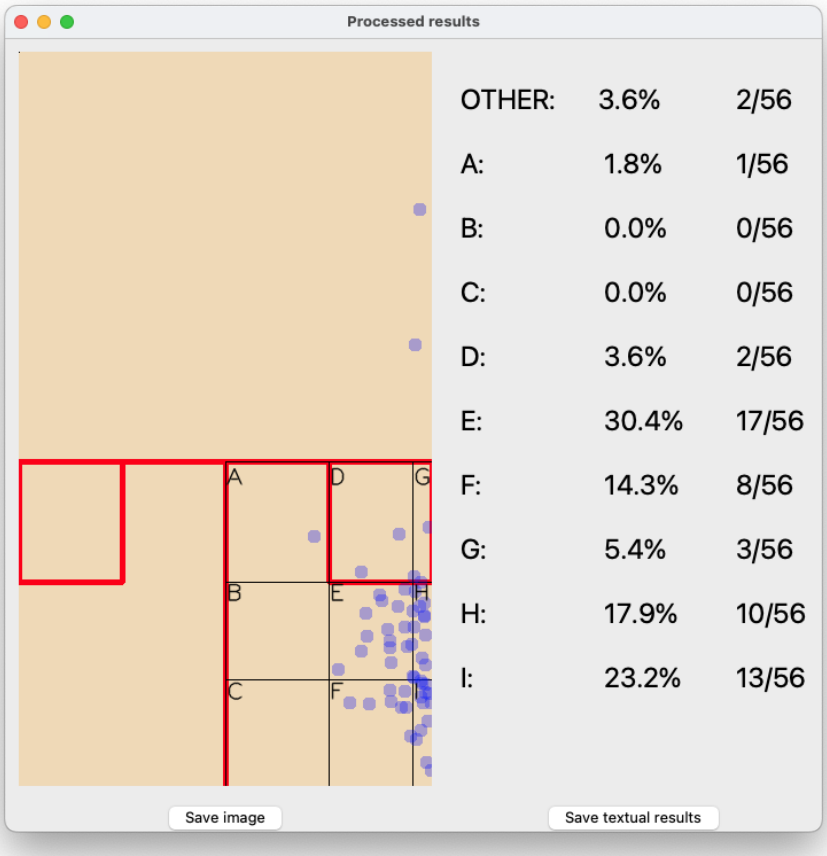
Once the points lower boundary has been clicked a pop-up window emerges. In case the last click was not placed correctly it is possible to click ”Cancel” followed by the ”Undo” button to readjust the selection. To proceed to the the analysis click ”Ok”.

A.4.5 Processing view



If the "Show processing video" option is selected then the video is shown in parallel with generating the bounce-marks. A progress bar is shown in either case to gauge the process of the analysis.

A.4.6 Output view



OTHER:	3.6%	2/56
A:	1.8%	1/56
B:	0.0%	0/56
C:	0.0%	0/56
D:	3.6%	2/56
E:	30.4%	17/56
F:	14.3%	8/56
G:	5.4%	3/56
H:	17.9%	10/56
I:	23.2%	13/56

The output view displays the results of the analysis. On the left panel, a heat-map-like image is show of the detected ball bounces, along with area markers. On the right panel is shown the counts of bounces per area.

Below each of the panels is a button allowing for saving the data into a user-selectable destination.

A.5 Developer Documentation

This annex describes each component of the squash drive analysis software and its implementation in detail.

A.5.1 Used tools, libraries and environment

This application was developed using Python 3.10 and PyCharm 2022.1.3 Professional IDE.

There are few dependencies of the application and they are provided at the root of the source code in the file requirements.txt for easy installation.

The dependencies are (excluding version numbers):

- OpenCV for optimized image processing algorithms
- Numpy for multi-dimensional array support

- MatPlotLib for data visualization during development
- Pillow for displaying images in the graphical user interface.

The project is structured in a file tree as follows:

```
squash-drive-analyst
├─ main.py
├─ pipeline.py
├─ detector.py
├─ double_exponential_estimator.py
├─ tracker.py
├─ bounce_detector.py
├─ stats.py
├─ gui
│  ├─ analysis_view.py
│  ├─ file_selection.py
│  ├─ guistate.py
│  ├─ output_view.py
│  ├─ panel_view.py
│  └─ set_up_view.py
├─ utils
│  ├─ court.py
│  ├─ rect.py
│  ├─ utilities.py
│  └─ video_reader.py
├─ requirements.txt
└─ README.md
```

A.5.2 Ball detection component

In the file ”detector.py”, the ball detection step is implemented. The main interaction with the Detector class is done by feeding it frames and receiving images where the foreground has been extracted from the background.

You can interact with the class through the following three methods: initialize_with(frame), ready() and process(frame). The first two refer to the initialization of the detector, as it requires three consecutive frames for the employed frame-differencing method. It uses Python double-ended queue to store a fixed number of images in memory that will be used for detection in the current timestep.

The process(frame) method is implemented by chaining various OpenCV function calls. It takes as input a regular video frame and outputs a binary image, with white areas representing the foreground and black regions the background. By modifying this method, it is possible to change the implementation of the detection and achieve different results.

A.5.3 Ball position estimation component

The functionality for estimating the position of the ball is implemented in the file "double_exponential_estimator.py".

The DoubleExponentialEstimator class implements Holt's double exponential smoothing in two coordinates and is able to hypothesize the future positions of a ball's bounding rectangle based on past observations. Instead of implementing the double exponential smoothing and applying it on each parameter, this abstraction is useful, as this this enables to encapsulate ball position estimation into a clear API with a single method call for obtaining a prediction.

The class has two main methods exposed for interaction: correct(position), which updates the observation buffer, and predict(t), which calculates the predicted location of the ball contour at a future time t.

A.5.4 Ball tracking component

The ball tracking is implemented in the class Tracker, which is located in the file "tracker.py". Besides the class constructor, which defines several important parameters, there is a single method for interacting with the class select_most_probable_candidate(frame, prediction). This method is intended to work with a binarized frame, such as one given by detector.process(frame), and a predicted ball position to output the position of the ball contour in the frame.

The class constructor defines multiple variables that can drastically affect the performance of the detector:

- __candidate_history defines the length of the observation history, which is searched for a ball trajectory
- __dist_cutoff defines a pixel threshold for maximum jump in observation distance to be disqualified as a ball contour
- avg_area defines the approximate area in square pixels of the ball's bounding rectangle

By calling select_most_probable_candidate(frame, prediction), a lot of work is done internally by methods hidden from the API. First, bounding rectangles of contours are extracted and contours that are close to each other are joined with __join_contours(frame). Then, the contours are filtered based on their relation to avg_area. If no contours remain after filtering, the prediction is used as the only observation. Finally, the __candidate_history is searched across layers for a path that scores the lowest according to the metrics defined in __find_shortest_path_candidate(prediction).

Changes to any step of the detector can lead to significantly different results.

A.5.5 Ball bounce detection component

The BounceDetector class is defined in the file "bounce_detector.py". Since BounceDetector uses homography for its detections, it requires both source and target coordinates during creation to compute a homography matrix. For ease of use, the source and destination coordinates do not need to be specified in any particular order, as they are sorted to correspondence before the homography matrix is calculated.

The class provides three methods, all of which should be used together when interacting with the bounce detector: update_contour_data(position), bounced(), and get_last_bounce_location().

The update_contour_data(position) method manages the data added to the position buffer of the BounceDetector. Internally, only coordinates projected with the homography matrix are used.

The bounced() method is used to check the contents of the ball position buffer for a bounce pattern and the get_last_bounce_location() method is used to retrieve the corresponding position of the ball from the buffer.

A.5.6 Statistics tracking component

The file "stats.py" contains the class AccuracyStatistics that manages the storage and representation of the ball bounces. It handles the division of of shots into target areas and calculating the statistics.

A.5.7 Utilities

The directory squash-drive-analyst/utils contains various python files containing utilities used in the application.

- The file "rect.py" contains a Python data class for representing a bounding box of a rectangle.
- The file "court.py" aggregates functionality for drawing a squash court and for drawing on it.
- The file "video_reader.py" contains functionality for reading frames from a video file.
- The file "utilities.py" contains various other utility functions.

A.5.8 Application pipeline

The complete analysis pipeline from the aforementioned building blocks is combined in the source file ”pipeline.py”. Aside from the object constructor, it provides a generator method process_next() that yields frames to be shown during the analysis, and a method get_progress() that returns a value in the range [0, 1] of completion of the analysis. The rationale for using a generator to access the analysis pipeline is rooted in the convenience, when the pipeline is run in a separate thread when combined with Tkinter.

A.5.9 Graphical user interface

The file ”main.py” is the entry-point of the application. The file contains a single class MainApplication that handles the interface state transitions and the logic-flow alongside it. The transitions in the application always follow the same ordering:

1. Begin in file selection view.
2. Upon file selection: transition to setup view.
3. Upon setup completion: transition to analysis view.
4. Upon analysis completion: transition to the output view.

All code related to the application window and user interface is located in the directory squash-drive-analyst/gui.