IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

3RD YEAR SOFTWARE ENGINEERING GROUP PROJECT

Revolutionizing Tennis with Machine Learning and Computer Vision

---

Authors:
Pinchu YE
Chuanqing LU
Huanyao RONG
Xi JIANG
Guy LEROY

Supervisor:
William KNOTTENBELT

July 2, 2019

Acknowledgements

We would like to express our sincerest thanks and gratitude to:

- Our supervisor William Knottenbelt, for his guidance, support, and proposing this project.
- Our client Matt Willcocks, founder of LV8Sport, and his team Cristian Lopez and Max Kingdon, for creating this project, their constructive feedback and giving us their vision of revolutionizing tennis.
- The Football and Fitness groups whose projects were closely related to ours and gladly worked with us on the multi-person pose estimation, exporting TFL projects on mobile, and setting up the server.
- Tom Titcombe for his enlightening report and code base on Pose Estimation.
- Lun Ai for his tennis project and vertical jump.
- Elliot Gunton for his work on the tennis and football side of the app.
- Silvia Vinyes Mora for her helpful thesis on machine learning in sports.
- Last year's group composed of Lun Ai, Elliot Gunton, Sunghun Jung, Andy Li, Eric Ruan Zhu and Jason Tsai for their precious report.
- Oliver Kaus for his project on athlete performance measurement.
- Anandha Gopalan for assisting with the project funding.

Contents

List of Figures

List of Tables

Executive Summary

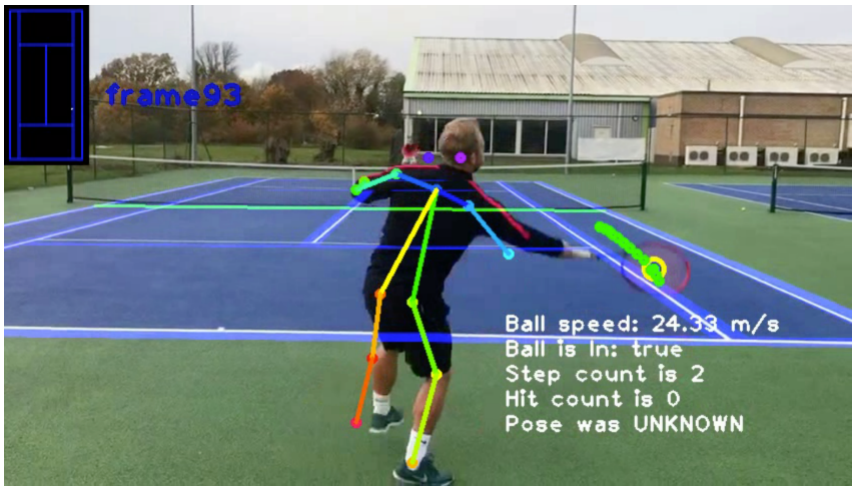


FIGURE 1.1: Pose detection shows skeleton of near player, ball detection is tracked with a series of green dots and court detection overlays a virtual court on the real one

Based on mobile platform and powered by machine learning and computer vision technologies, the users of our application, tennis players and coaching staff, are able to collect metrics in a tennis game. The users are allowed to obtain data such as ball speed, shot accuracy, step count and backhand/forehand count from filming with a smartphone camera or from video records of past matches. With the help of these data, the application assists players to improve their performance over time.

Advances in machine learning and computer vision are starting to revolutionize the sport industry. Similar solutions are appearing on the market but generally require investing in high resolution cameras or sensors and needs to be setup on each court. Our solution merely requires a phone and to download the app.

Our technology would better the lives of tennis enthusiasts, coaches and people working in the tennis industry in general. This proof-of-concept opens doors for more applications, such as enabling referees to not have to do the tedious tasks of counting points and checking whether each ball is in or out. Ultimately, it enables to have more competitive tennis tournaments, pushes the boundaries of humans' capacities in this sport. It shows that digitizing the sport industry as a whole is possible and can have a huge positive impact.

Introduction

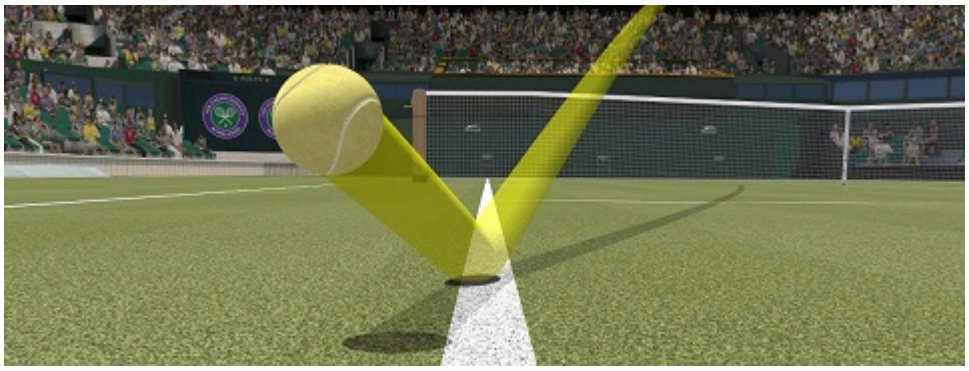


FIGURE 2.1: Hawkeye, technology costing $60,000 or more to set up on each court [1], reconstitutes shots in 3D.

2.1 Motivation

Athletes have been performing better with advances in technology, lighter, more flexible, and stronger shoes, tennis rackets got better with larger string holes and new materials: laminated wood, steel, weed aluminum and now composite. In a time dominated by disruptive technologies radically changing industries, we want to explore how sport can be digitally revolutionized. Athletes would reach more of their potential, and coaches would have more insightful metrics to base their advice on. In addition, we want to help democratize sports by creating low-cost tools that can be used by everyone, not only the top-paid performers, but also the casual practitioners with only an app on their smartphone. We chose to focus on tennis as this sport is widely popular and its well segmented court and game rhythm enables us to leverage the power of machine learning and computer vision in the time frame of the project.

Previous progress that were made to tennis were due to new and cheaper materials as well as better crafting techniques. Throughout the last couple decades, tremendous advancements have been made in computer science which lead to further progress with machine learning, computer vision and sensors. A major competitor in the area of tennis is Sony who developed Hawkeye [2], a technology helping referees make better informed decisions by detecting balls' impact locations. Those tools are often cumbersome and expensive, thus not available to the average player. We want to help push tennis players' performances by bringing to life

low-cost training tools, and create a proof of concept for future work in digital sport analysis.

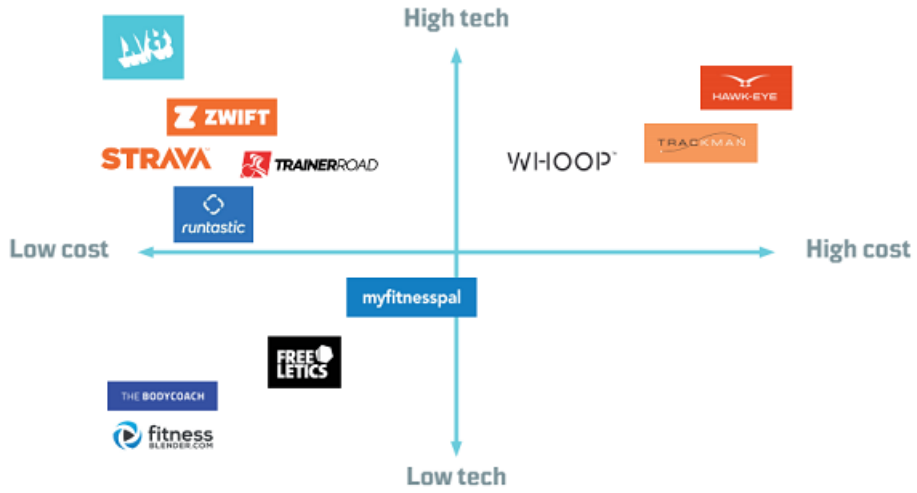


FIGURE 2.2: Competitive landscape, LV8Sport's goal is to deliver a high-tech, low-cost product.

2.2 Objectives

Our objectives for this project were multi-sided:

- Explore the possibilities offered by machine learning and computer vision to gather metrics from tennis footage of a smartphone.
- Deliver a proof-of-concept to the LV8Sport's team that could be showcased to investors and hopefully secure an investment to improve the product.

2.3 Contributions

Our contributions are the following:

- Applied state-of-the-art multi-person pose estimation.
- Applied ball tracking on near side of the court.
- Detected court from a hand held camera shot.
- Tracked several metrics related to the athlete's performance.
- Assessed the possibility of running embedded models on the phone.

Previous Work

3.1 Neural Network

Artificial neural network, inspired by the biological neural networks, has become the dominant framework for algorithms in the field of machine learning. A great part of this project relies on neural network to do inference on human pose estimation and object tracking.

Neural networks work by applying an affine transformation onto an input vector of N dimensions and generate an output by adding a bias to the result of transformation. According to this output, an activation function is applied to decide the significance of output, in other word, the next layer would know if this neuron is activated.

A numerous amount of neural networks have been developed such as Alex Net, VGG Net and MobileNet, each serving different purposes or being an improvement of another.

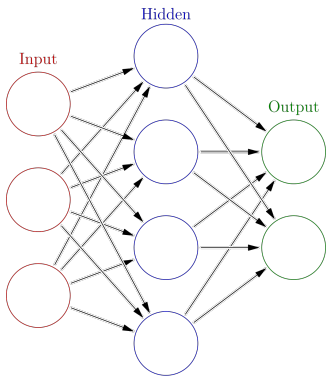


FIGURE 3.1: Simple neural network

3.2 Convolutional Neural Network

As the name implies, a convolutional neural network produces an output for the next layer by applying a convolution operation to the input. This methods preserves information of neighboring pixels so that better prediction could be made. Moreover, a CNN reduces its number of parameters significantly by sharing, a filter

in a single convolution operation is reused over the whole image.

CNNs usually include a pooling layer after a few convolutional layers to downsample the image filtering to keep the most import information, this improves performance of the network while having little change in accuracy.

Both networks involved in this project make use of a convolutional neural network as it has shown to be effective in image and video processing.

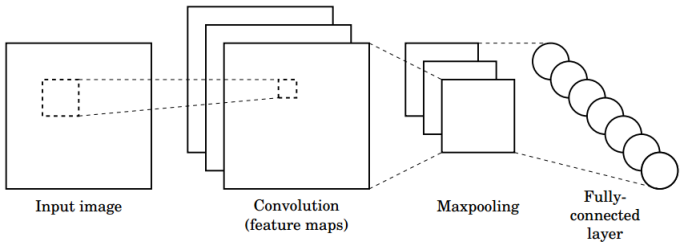


FIGURE 3.2: Simple convolutional neural network

3.3 Pose Estimation using Part Affinity Fields

Pose estimation focuses on locating and connecting anatomical keypoints of human bodies. It is still a very challenging problem in computer vision, and some difficulties are the following: [3]

1. Each image may contain an unknown number of people.
2. Interactions between people cause increased complexity due to contact and occlusions.
3. Time complexity of algorithms tend to grow linearly with the number of people.

Commonly, human pose estimation includes two stages: first detecting people within a frame then performing pose estimation for each person found. This method is quite error prone since there is great chance of losing track of people when there are interactions among them.

In 2017, a new way of multi-person pose estimation was introduced by Zhe Cao et al.[3] that runs in real time. The model simultaneously locates body parts and predicts association, that is, two neural networks run in parallel to produce the end result. Finally, a greedy parsing method is used to extract useful information from both branches to form human skeletons.

3.4 Object Detection (YOLO)

Object detection is mainly concerned with locating objects within a image, and it suits perfectly for our purpose of detecting flying tennis balls during a tennis game. In this section, we will discuss the techniques of You Only Look Once (YOLO) object

detection.

You only look once (YOLO) is an advanced object detection network that works quite differently from other networks. Unlike the other networks that carry out prediction on thousands of subregions of a image, YOLO applies the network to the whole image and the predictions are based on global context. This technique ensures speed as well as accuracy of the network. In our case, YOLO outperformed all the other methods we experimented and produced acceptable results.

The architecture of the YOLO network consists of series of residue blocks and convolutional layers. A residue block is basically several convolutional layers with a shortcut path, this shortcut effectively allows for deeper network to be constructed.

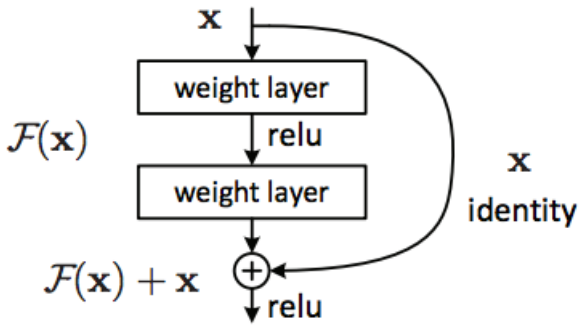


FIGURE 3.3: Simple convolutional neural network

Design & Implementation

4.1 Overview

This project has been previously undertaken by other teams and individuals, each time making their own contributions. It is an app developed under Android Studio. Android Studio is the official integrated development environment (IDE) for Google's Android operating system. The IDE supports Java which is the language of the legacy code. Java files are divided in activities which represent pages in the app and back-end code. We mainly worked on the back-end files as our goals were to implement or enhance artificial intelligence based algorithms. We used the library OpenCV for real-time computer vision.

Whenever a user records a tennis drill. The video is sent to a server which runs four different machine learning and computer vision based algorithms. The results are then used to produce metrics about the players' performances. Those metrics are sent back to the user's device, ready to be ingested.

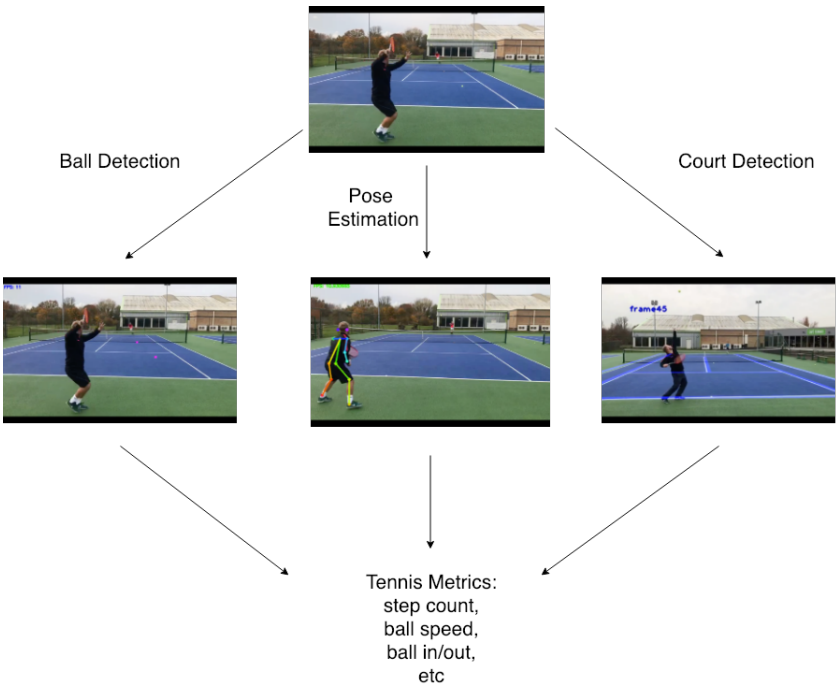


FIGURE 4.1: System Overview.

4.2 Pose Estimation

4.2.1 Initial Attempt

It is not an easy task to integrate pose estimation into our mobile application. The pose estimation is based on a machine learning model developed in 2016 [3]. In order to merge the model into the mobile application, we initially intended to shrink the model into a mobile version with the use of TensorFlow Lite[4]. TensorFlow is an open-source software library often used for machine learning applications such as neural networks. TensorFlow Lite runs machine learning models directly on mobile and embedded devices. However, it came to a failure as the compressed model did not provided expected output. Moreover, it requires high computation power to perform pose estimation on videos, so it seems to be impossible to have real time pose estimation on a mobile device.

In order to cope with the problem above, we decided to construct a cloud server from Azure for the application. Having high computational power, the server is able to handle all high cost computation steps. Therefore, we could move the pose estimation model to the server side and all the analyses can be done on the server. The mobile device then will only need to send the original video to the server and receive the annotated video from the server. Although the application is no longer offline, the processing speed of video is undeniably increased.

4.2.2 General Algorithm

The model introduces an efficient method of utilizing Part Affinity Fields (PAFs), a series of 2D vector fields which represent the coordinates and orientations of different human body parts inside the image. The PAFs are then able to produce association scores that demonstrated to be sufficiently well enough to achieve high quality detection results through steps of matching and greedy parsing.[3].

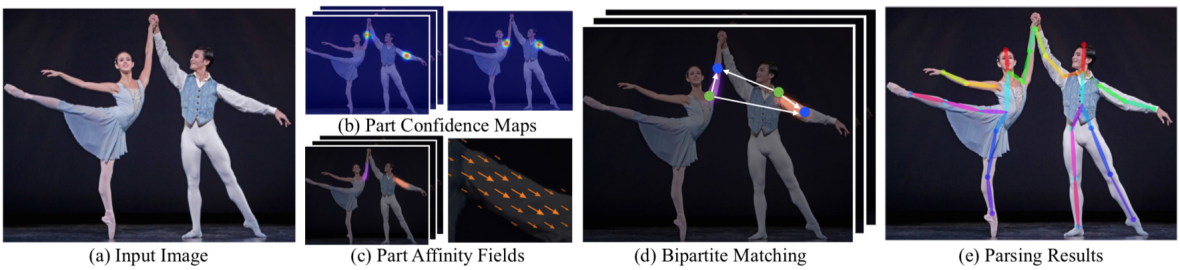


FIGURE 4.2: Image representation of pose estimation procedures [3].

4.2.3 Usage

The machine learning model is trained well enough to label human joints and limbs throughout the video with fairly high accuracy. The coordinates are then combined with other data such as ball coordinates to help determine metrics such as player shot recognition and step count.

FIGURE 4.3: The skeleton labels that would be drawn on the detected human. Each joint is mapped to a unique number. For example, head is mapped to 0 and left shoulder is mapped to 2.

4.2.4 Limitation

The pose estimation model has great dependency on the resolution of the camera. If the player or the camera moves too fast, the player shown in the video might not be clear enough for the model to detect humans and label the pose skeletons accurately and precisely. Moreover, currently the model is only able to detect the player that is closer to the camera. The player at the far side is rarely detected. Even sometimes the farther player is detected, he or she is labeled with all body parts coordinates crowding together, still giving out an unwanted result. Lastly, if the player only shows some parts of his or her limbs, the labelled skeleton will not be complete as well.

FIGURE 4.4: The image shows the unanticipated labels for the player on the far side of the court. In addition, the player close to the camera does not have a full skeleton due to not showing the full body.

In addition, sometimes we might only be able to see one side of the player if he or she is standing sideways, so that we are unable to detect some of their body parts as they are overlapped by each other.

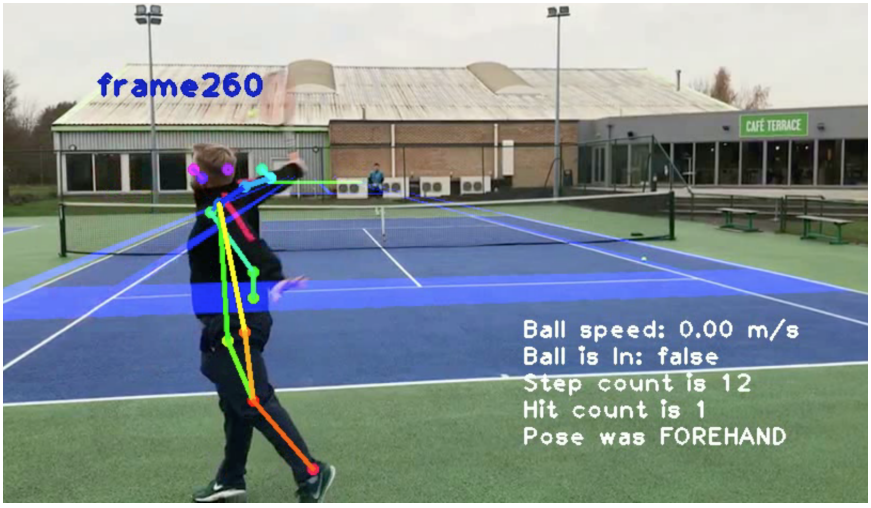


FIGURE 4.5: The image shows the unanticipated labels for the player when standing sideways. As one may see, the coordinates of legs overlap with each other.

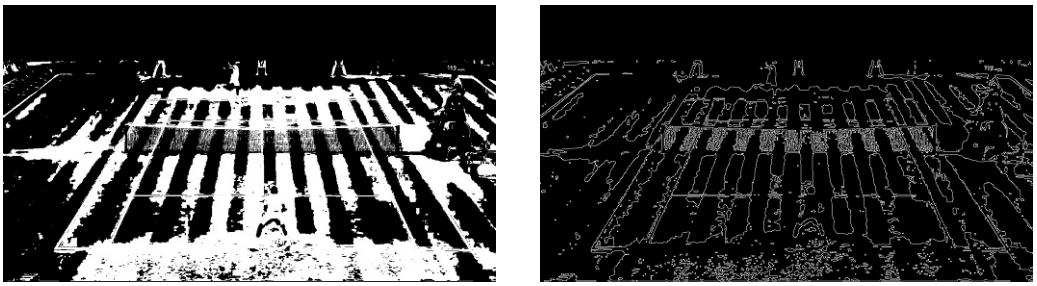
4.3 Court Detection

The aim of court detection is to give a point of reference for ball tracking and player's movements, e.g. be able to transform the frame coordinates of the ball to coordinates relative to the court in meters. The court should be correctly detected even when the camera moves, so that ball landing points for instance move on the screen alongside the court. Certain videos are more challenging for court detection, this is the case when the court is partially out of the frame, when the brightness is either very low or very high or when the player occludes a large portion of a court line. Similar to all the artificial intelligence related algorithms we have implemented, it is directly run on the server.

4.3.1 Related Work

One of the first algorithm detecting court lines was developed in 2003. Most algorithms followed the following steps: extract white pixels, find lines using RANSAC [5] or Hough Line transform [6], associate found lines on the image to each court line.

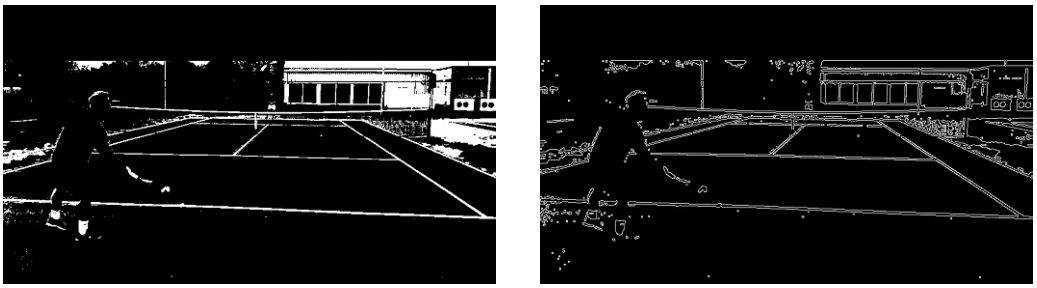
Our supervisor and his PhD student Silvia Mora had previously work on an algorithm [7] for court detection which has then been improved by Tom Titcombe. First the court pixels are discriminated into darker ones and lighter ones. The threshold is found by taking the average brightness of a court which is computationally very fast. The black and white image is then pre-processed with Gaussian blur [8] to reduce noises and Canny edge detection [9] to find the edges. Finally the algorithm applies a Hough line transform to find intersections of white lines and compare them to pairs of all possible combinations of corners in a tennis court, choosing the ones which



(A) First step: frame is pre-processed with white mask (B) Second step: applying Gaussian blur and Canny edge detector

FIGURE 4.6: Here we can see two steps of the previous court detection algorithm on a frame from a tennis game. The white mask threshold to discriminate pixels is found by taking the average brightness of a court but in some cases (such as this one) too many points would turn white, and in some others too many would turn black, because average brightness is not a perfect indicator to separate pixels belonging to the court lines. We can see that after the white mask, the Gaussian blur and the Canny edge detector have been applied there are too many edges, and the court lines are not clearly apparent, hence the Hough line transform could likely return edges that are not court lines.

match with the most white pixels. The Hough line transform returns the edges' coordinates. To reduce computing time, the court is detected every ten frames, this number can change depending how smooth one wants the court lines detection to be. Last year's group composed of Lun Ai, Elliot Gunton, Sunghun Jung, Andy Li, Eric Ruan Zhu and Jason Tsa, improved the algorithm to find a suitable threshold. In the case where the white mask is too restrictive (not enough pixels are considered white), no court line can be detected as there are not enough edges. The algorithm was improved by passing the frame through a more permissive white mask until some court lines are detected.



(A) First step: frame is pre-processed with white mask (B) Second step: applying Gaussian blur and Canny edge detector

FIGURE 4.7: The new court detection algorithm computes a more suitable white mask. Figure (A) shows the frame after the white mask has been applied, one can clearly see the court line edges. Figure (B) shows the next step; a Gaussian blur is performed to reduce noises, a Canny edge detector is applied to find edges and finally apply a Hough transform to get the edges' coordinates. Since the threshold discriminated well the pixels, the court lines would correctly be detected.

4.3.2 Enhancements

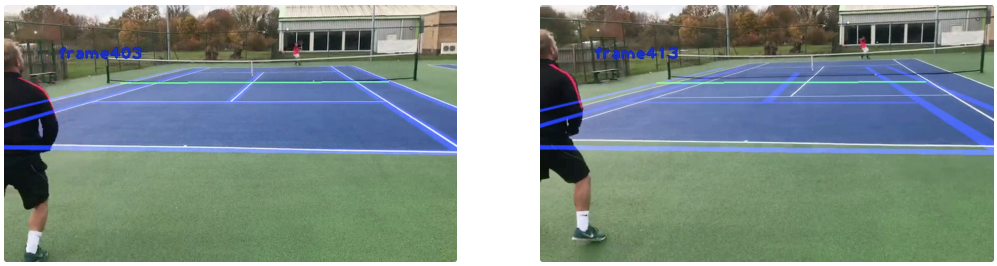
The algorithm was improved by finding a more suitable threshold to better discriminate light and dark pixels. The old threshold would almost always be too permissive, thus the new threshold is an average between the old threshold found by repeated relaxation and white, which makes it more restrictive.

As lines would not get correctly detected in some frames, we needed to remove the bad court line candidates. In order to avoid detected court lines from jumping from frame to frame, we tracked the position of the net and allowed it to move a certain displacement between each frame. If the displacement is too large between two frames, we assume the detection has gone wrong and instead use the court lines from the previous frame as the actual court lines should not have moved too much.

4.3.3 Limitation

The algorithm is subjected to the premises that the court lines are distinguishable and not largely worn out. Depending on the different types of surfaces (grass, clay and hard), performance would also vary from courts to courts.

One of the other assumptions this algorithm builds upon is that the displacement between frames should be subtle. If the camera moves rapidly, the algorithm might filter out new court candidate as a detection failure, for it not being within a certain range to the previous candidates. The algorithm then keeps the court line from previous frames, despite of the fact that it is unfitting to the actual courts.



(A) Frame 403 (B) Frame 413

FIGURE 4.8: Side by side comparison of two frames, before and after cameraman moves camera by a large displacement, that shows the failure to update the new displaced court accordingly.

The algorithm did not consider the influences from factors such as brightness, as the protocols of the application advise users to only use this feature on videos that are neither overexposed nor underexposed. Weather condition, as one of the protocols, is assumed to be sunny. Puddles of water on the court incurred by rain cause specular reflection of light to the camera, which may result in wrongly detected court.

4.4 Ball Tracking

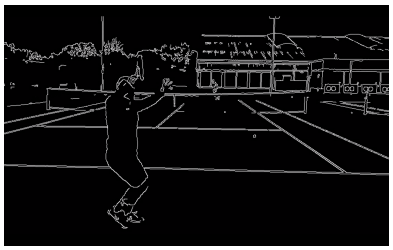
This module works together with the court detection module, in order to detect the landing points where the tennis ball bounces off the court. Based on this piece of information, whether the ball landed within the legal margins of the court can be inferred.

4.4.1 Initial Attempt

Initially we tried to detect tennis balls by picking up RGB color of a certain range and filter out the real ball from noises, and it worked reasonably well with frames where green color is relatively rare. However, we quickly realized that it is almost impossible to gain the ground truth value from frames where the ball has nearly the same colour as the environment, as shown below (too many points were qualified as tennis balls).

Then we switched our algorithm to detect edges from a frame and look for any enclosing contours, graph below shows the edges detected. However, from this intermediate image we came to the conclusion that there could be too many enclosing contours and the results wouldn't be much better than the last ones so we decided not to move on with it.

(A) Ball detection using pixel extraction



(B) Ball detection using Canny edge detection

FIGURE 4.9: Ball detection, initial attempt

4.4.2 YOLO-V3

We finally turned to the methods of detecting tennis balls by machine learning. Since a tennis ball is a common object that many models are trained on, we simply looked for pre-trained models that suit our algorithm. In order to find the best one, we compared two state-of-the-art networks that satisfy our needs of tracking tennis balls: Mask-RCNN and YOLO-V3, both of which track the positions of balls within a frame but YOLO-V3 is far quicker because it does not need to mark the balls precisely. Therefore, our ball detection is mainly based on YOLO-V3.

What we need to do in this section is to filter out the balls that are static and lying on the floor, because we want the ball only when it is flying in the air.

First of all we need to trace the ball. That is, we need to identify which points in the subsequent frames are actually the same balls as the given point in the frame. The approach to identify this is to search for some subsequent frames, in our case we empirically chose the next three. Then we find the coordinate with the smallest distance in the next frame: if it is within a threshold, we regard it as the same ball; if

not, we search for the next frame. In order to prevent the case that the same coordinate is being used twice, we only consider the non-traced ball when looking for the same ball in the subsequent frames. The search will be stopped if there is no point within the threshold in the next 3 frame. Therefore, our assumption is that the ball movement is slow enough so that in the next frame it will not move so far from the original point.

For example, here are some ball coordinates for same frames, and DIST_THRESHOLD = 30.0, and we use the Euclidean distance to calculate the distance between 2 points. We want to trace starting from point {101, 102}.

..., [{101,102}], [{200, 201}, {304, 308}], [{115, 109}, {113, 107}, {201, 202}], [{102, 103}], ...

distance({101,102}, {200, 201}) = 140.01

distance({101,102}, {304, 308}) = 289.21

which are all larger than 30, so there is no same ball to trace in the next frame.

distance({101,102}, {115, 109}) = 15.65

distance({101,102}, {113, 107}) = 13.00

distance({101,102}, {201, 202}) = 141.42

so point {113, 107} with distance 13.00 is within the threshold and is minimum, so we choose this as the coordinate to be traced, and use this coordinate to trace for the subsequent frames and repeat the same algorithm shown above, until we cannot find a coordinate within the distance threshold in 3 frames or we reach the end of the video.

Also, after the point {113, 107} is traced, we will not consider this point anymore. For instance, when we are tracing {200, 201}, we will not consider this point but will only consider {115, 109} and {201, 202} in that frame.

However, the limitation of this algorithm is that it is possible that in the next frame, another ball is even closer to the ball in this frame than the ball that we should classify as the same ball, so our algorithm might misidentify the ball. Nonetheless, the possibility for this is quite low.

After tracing, we have several tracks, each of which contains the coordinates of the same ball in some frames. Note that, the coordinates in the different tracks does not necessarily mean different ball.

Now we want to interpolate to remove the discontinuity. The algorithm is exactly same as the one used in pose interpolation in 4.5.9, which will be given more details.

Finally, we want filter those static balls and leave the balls that is being played. For each ball coordinate in all tracks, we convert it into court coordinate. For each

track, we get the average distance between 2 points. If the average distance is beyond some threshold, we regard it as dynamic ball. Things will work better if the camera is not moving too much. Because after converting into court coordinate, the distances between balls in the air in 2 continuous frames will give ridiculously large number, we can just exploit this feature to find the dynamic balls that are being played. We found this when we used the previous naive method to calculate ball velocity.

For example, here are 2 tracks, and coordinates are court coordinates in meter

[. . . , null, null, {7,8}, {7.5, 8.3}, {6.7, 7.4}, null, . . .]

[. . . , null, {9.3, 10.6}, {17.5, 18.9}, {53.8, 72.3}, {102.6, 207.7}, null, . . .]

in which ". . . " are all "null", and the coordinates are not necessarily true values that will be obtained from a real video but random values I choose for demonstration.

distance({7,8}, {7.5,8.3}) = 0.583

distance({7.5,8.3}, {6.7,7.4}) = 1.204

so Average Distance = (0.583 + 1.204) ÷ 2 = 0.894, which is small and likely to be the static ball, and we need to filter this out.

distance({9.3, 10.6}, {17.5, 18.9}) = 11.667

distance({17.5, 18.9}, {53.8, 72.3}) = 64.570

distance({53.8, 72.3}, {102.6, 207.7}) = 143.926

so Average Distance = (11.667 + 64.570 + 143.926) ÷ 3 = 73.388, which is large and likely to be the dynamic ball, and we need to leave it.

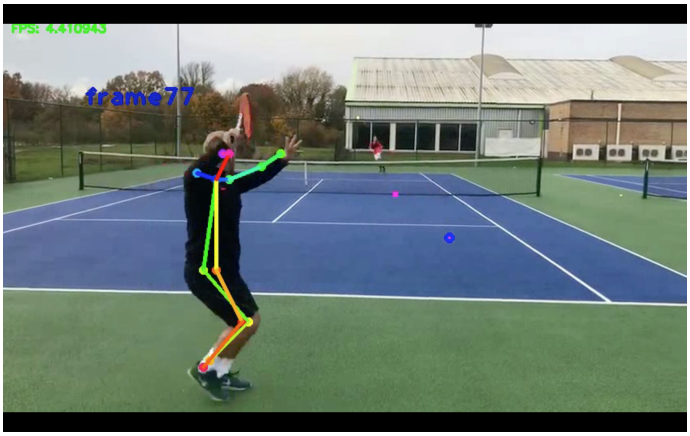


FIGURE 4.10: Ball detection by machine learning. The ball candidates before filtering are indicated with pink color. After the filtering, the ball left is represented in blue.

4.4.3 Limitation

Although it gets more robust compared to the pixel extraction methods, there are still certain problems we could not solve. One being that the model does not recognize balls that are blurred due to poor filming quality. The model is trained mostly with still objects, therefore it tends to lose track of ball when it moves too fast and the camera could not capture a vivid image. Possible way to address this issue could be feeding more images with blurred tennis balls to the network and retrain the model.

Another limitation is caused by occlusions of tennis ball. Depending on the angle of camera and position of players, the tennis ball could be occluded from time to time, resulting in loss of important metrics. In order to cope with this problem, interpolation is used to predict the location of the ball in those undetectable frames. After the interpolation is done, although some frames are still unable to have a ball prediction due to lack of information of previous or subsequent frames which successfully detect the ball. The interpolation method is also used in generating certain metrics and will be explained with more details in 4.5.9.

4.5 Metrics Gathered

4.5.1 Ball Bounce

Ball bounce detection is done as previous work. In general, the algorithm keeps tracking on the direction of the ball, and any negates in direction of the ball is considered as a bounce. However, the previous work did not find a way to separate ball being hit by player and ball bouncing on the ground since the algorithm treats these two cases as one. As an enhancement, through using different methods listed in the next section 4.5.2, we are able to obtain the ball hit frames. Whenever the algorithm detects a bounce at a certain frame, we will check if the frame is one of the hit frames, therefore successfully differentiating hit and bounce.

The limitation is that sometimes a hit may fail to be detect, so that it will still be falsely considered as a bounce. In addition, if the ball track is not good enough, the ball bounces will be poorly detected as well.

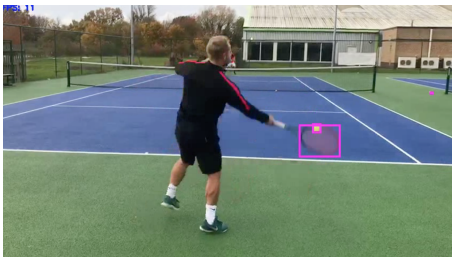
4.5.2 Ball Hit

The detection of ball hit was a rather difficult challenge. The principle was simple from the first look: Since the previous algorithm has already obtained all the ball bounces and hits mixing up together, one will only need to manage to distinguish them from one another. Following this principle, different attempts were made.

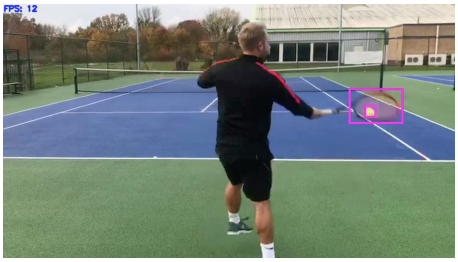
Detection from YOLO

During processing some of the videos using YOLO mentioned above, it comes as a surprise that when the player is hitting the ball, not only the ball was being detected as a square, but also the racket was being detected as a square well. Since YOLO is also capable of detecting many other things, and racket might be one of them. The two squares enclose with each other in most of the frames that the ball is

being hit. Therefore the goal can be completed with direct help from YOLO.



(A) Frame A



(B) Frame B

FIGURE 4.11: Extracted Frames from YOLO ball detection

However, it was then found out that the enclosing squares fail to appear in most of the other videos. The main reason for this is that the YOLO model is not well trained enough to detect racket as accurately as ball. Therefore, the improvement is made to this measure.

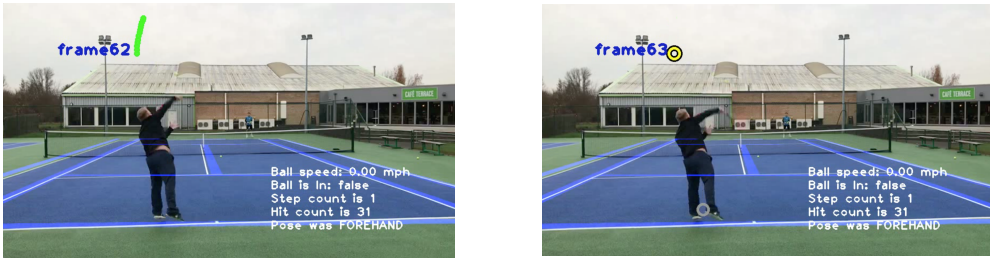
Using Pose Estimation

Another thought came to mind that the hit ball must be closer to the player than the bouncing ball, since the player has to move closer in order to hit the ball. In addition, the hit balls events and the bouncing balls events are happening in a rotation. In other words, it is impossible to have the ball bouncing without being hit before, or vice versa. Ideally, by taking raw bouncing balls(which mixed up with hit balls) pair by pair with anticipating every pair containing a hit ball frame and bouncing ball frame, one is then able to compare the the pair with each of its Euclidean distance to the player's body part(right wrist or left wrist) from pose estimation, it is then possible to differentiate the hit ball from the bounce.

However, it is proved to be a rather naive thought as it is difficult to check which ball is closer to the player from only one certain view perspective by the phone camera. In addition, the ball detection is not perfect either, sometimes a bounce ball will not be detected, so one can not assume that every bounce is followed by a hit.

Path Vanishing Algorithm

By observing different processed videos, we found that when the ball is being hit, it will have very high speed, usually becoming untraceable by YOLO. Attributed to this, we came up with the idea that whenever a ball, has being detected accurately and continuously for a certain period of frames, suddenly disappears from the YOLO detection, it is then highly likely being a hit. We obtained help from path detection of previous work, which leaves the most likely track for where the ball was in preceding frames. Once the track(green shown in Fig4.12) is gone, it means that the ball is lost in track, therefore being hit. Thus the endpoint of the path(the last spot where the ball is still getting detected) should also be the position where the ball is getting hit.



(A) Frame A (B) Frame B

FIGURE 4.12: Frames demonstrating the path vanishing algorithm. At the time that the path vanishes, the endpoint of that path will be drawn, being the last location that the ball was detected, which is considered as the hit.

4.5.3 Homograph Matrix

It is necessary to introduce the homograph matrix, which can be seen as a conversion factor that is also involved in the court detection part from previous work. Basically, It is a transformation matrix that can map any coordinates within the video domain to the actual coordinates in the tennis court from an aerial view, and vice versa. The court is also drawn this way: First we detect four points from the video that fit to four corners of the court lines, obtaining the homograph matrix. Then we just draw the actual court from an aerial view out and project the court on to the video using the homograph matrix.

4.5.4 Ball In and Out

With the help of homograph matrix, we could convert any points in the video to the actual coordinates on the court. Therefore, by simply comparing the coordinates of the ball at the bouncing frame with the known dimensions of the court, we will have a relatively accurate result on whether it is a valid shot.

However, there are also limitations. Since the homograph matrix is generated during court detection, it will depend on the quality of detected court. If the court is detected badly, not only to mention that the drawn court lines will not fit with the actual court to a good extent, but also any transformation results using the homograph matrix will be erroneous as well. Moreover, if the bounce fails to be detected, the in/out result will not be given either.

4.5.5 Hit Map

Using the homograph matrix, we are able to generate a hit map, which is the tennis court from aerial view with all bounces that take place in the video.

All the bounces recorded during the game are later used to calculate a score of the shots quality for the player at the far side.

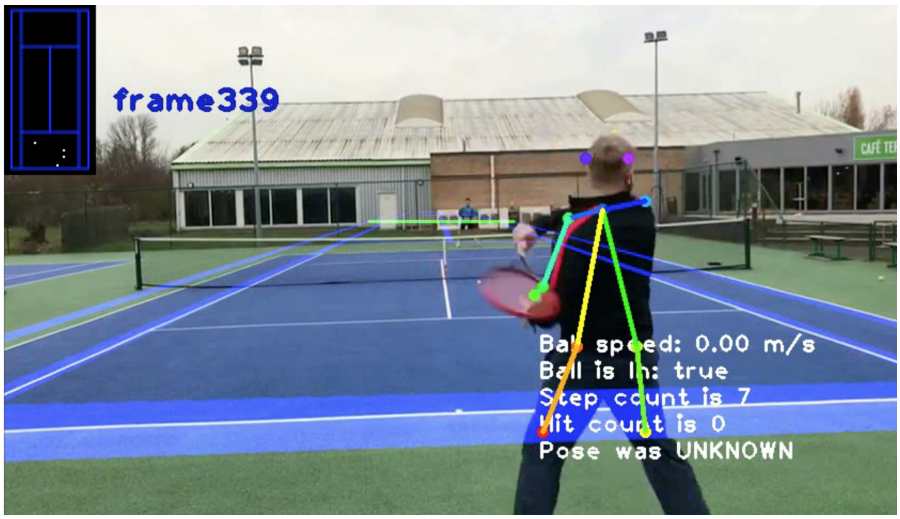


FIGURE 4.13: Hit map shown at top-left corner.(White dots represent the currently recorded bounces)

4.5.6 Score

The score takes the average from each of the shot. The quality of the shot depends on where it lands. If the ball is not in the court, the score will be zero. Otherwise, the harder for the player to hit the ball, the higher the score. To be specific, the farther the ball is from the net (comparing y coordinate), the score rises; The closer the ball is from the sidelines (comparing x coordinate), the score rises.

4.5.7 Speed

Here, it is specifically defined as the speed of the ball after it bounces the ground. It is calculated in several steps:

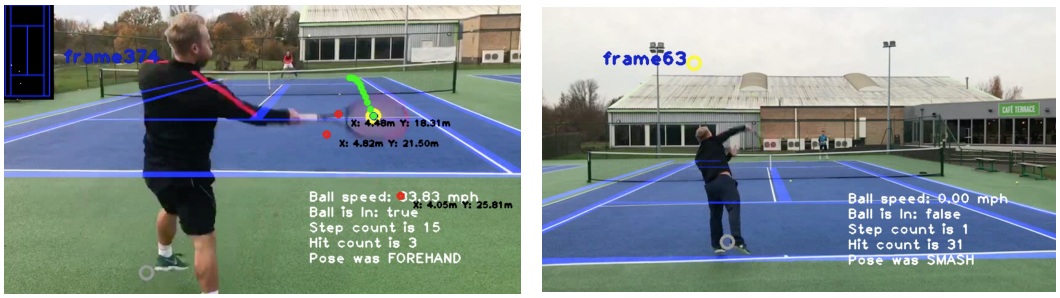
- Record the actual court location of a single bounce using homograph matrix. The coordinate is recorded as a 3D point with z set to 0 since the bounce is at the ground level. In addition, the bounce frame number is also recorded.
- Once a hit takes place, derive the actual court location of the center point of player's feet at that frame using data from pose estimation as well as homograph matrix. Record this location also as a 3D point with z set to half of the player's height that will be entered by the player when using the application. z is set to half of the height since one assumption is that the player is hitting the ball at half of his or her height in average. The hit frame number is recorded as well.
- Calculate the distance between two 3D points with $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$
- The time is calculated by $\left(hitFrameNumber - bounceFrameNumber\right) \times FPS$
- Divide distance by time to determine the speed.

The limitation is that this speed has great dependency on how accurate the bounce and hit are detected. Moreover, we assume that the bounce will happen exactly at half of the player's height, while it is not the case. This becomes a limitation because it is challenging for this project to detect the third dimension with the only help

from homograph matrix, which only transforms points in 2D. In future progress, it is likely to integrate other advance technologies into this project in order to achieve the additional dimension measurement.

4.5.8 Shot Recognition

Shot recognition, in other words, is to determine which type of shots the player uses when hitting the ball, whether forehand, backhand, serve or smash. This is done by simply comparing the position of player's certain body part and the ball at the frame that the ball is being hit. To be specific, at the hit frame, we compare the coordinates of player's nose with the coordinates of the ball. If the ball's y coordinate is greater than that of nose, then it is a smash. Otherwise, we compare the x coordinates. If the x coordinate of nose is greater than the ball, then ball is at the left side of the player, thus being a backhand shot. Otherwise, it is a forehand shot.



(A) Forehand pose (B) Smash pose

FIGURE 4.14: The algorithm distinguishes poses(shown at the bottom-right corner)in two different frames

However, the limitation is that shot recognition depends heavily on whether the hit is detected. If the hit is not detected, the shot recognition will not give a result. In addition, due to only having one view perspective of the camera, sometimes the algorithm may fail to recognize the correct pose.

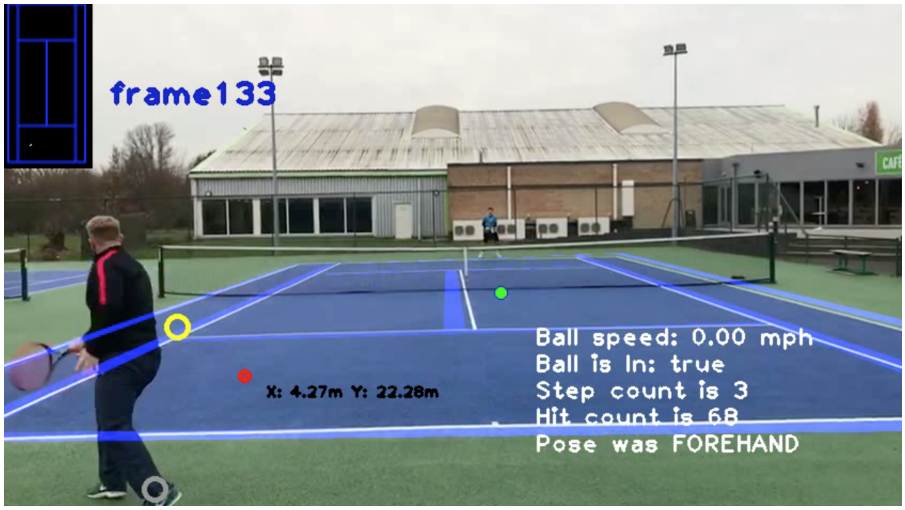


FIGURE 4.15: The algorithm fails to recognize a backhand shot

4.5.9 Step Count

The presumption is that the coordinate of 2 ankles of player is known. Our approach is that if distance between left ankle and right ankle is close enough, we increment the step count.

Use of Interpolation

However, since there are some frames in which one or more body parts cannot be recognized, we need to fill these frame using the information that we already know. That is, the coordinates of that body part in the previous and subsequent frames. The specific method is easy: we interpolate using the point of N-section if 'N-1' frames are missed. We will not interpolate frames initial and final points if they are missed because there are no previous and subsequent points to interpolate.

For example, in a video there are 9 frames, (This is just an example aimed for simplicity, in reality we will not have the video as short as this.) the points below are the coordinates of left ankle being recognized by the model for each frame, and "null" means "unrecognized".

null, null, {101, 102}, null, {103, 104}, null, null, {109, 110}, null

After our algorithm, we have:

null, null, {101, 102}, {102, 103}, {103, 104}, {105, 106}, {107, 108}, {109, 110}, null

because:

$$101 + (103 - 101) = 102$$

$$102 + (104 - 102) \div 2 = 103$$

$$103 + (109 - 103) \div 3 = 105$$

$$104 + (110 - 104) \div 3 = 106$$

$$103 + (109 - 103) \div 3 \times 2 = 107$$

$$104 + (110 - 104) \div 3 \times 2 = 108$$

Formally,

n-th frame: $\{x_1, y_1\}$, ..., k-th frame: $\{x_2, y_2\}$, where "..." are all "null"

$(n+i)$-th frame can be interpolate as $x_1 + (x_2 - x_1) \div (k-n) \times i, y_1 + (y_2 - y_1) \div (k-n) \times i$, where $1 \leq i \leq (k-n-1)$

However, such interpolation has limitation: when the body part is really outside the edge of video, it will still consider it inside the video, and the body part will "stick" on the edge of video, which is not same as the ground truth.

After interpolation, things become easy. What we need is to obtain the distance between 2 ankles in each frame, and count the number of times it falls into a specific

threshold. However, we need to choose the threshold carefully: if the value is too large, 2 or more steps will be considered as 1 step; if the value is too small, steps will be missed because the distance versus frame index is a discrete function, and it will simply "jump over" that small value.

4.6 Server Set-up

All the video processing steps mentioned above were run on Azure server instead on smart phones locally due to the limitation of computational power of smart phones. We used machine specialized in machine learning to achieve such task, with following specs:

- 6 cores CPU (E5-2690v3)
- 1 x K80 GPU (1/2 Physical Card)
- 56 GB of RAM
- 380 GB SSD

Machine with such powerful processing power is still more than enough of what we need at current state as we have not fully exploited the hardware for concurrent processing, that is, every single video is processed in serialized manner and only one frame of a video is processed at a time.

Regarding the budget, we had around $1000 in total, and NC6 costed around $1 per hour. Since we have strictly controlled the hours when we turned on the server, there should be plenty credits left.

4.6.1 Server Structure

We used Apache HTTP server as a reverse proxy which receive requests from clients and redirect them to the backend server, then retrieve the responses and send it back to the clients.

On the backend, we employed a combination of Gunicorn [10] and Flask [11] to serve each request. Gunicorn is another HTTP server that uses pre-fork worker web server model, it means that the server creates processes for handling multiple requests, hence serving multiple clients at the same time. Currently, the server creates three workers for serving three request asynchronously, it is possible that the server could take care of more requests but that is not the focus of our development at the moment.

For the web framework, we used Flask which is quite light-weight yet efficient. There are two main endpoints we used for uploading and retrieving video: /upload_video_0/tennis and /processed/tennis.

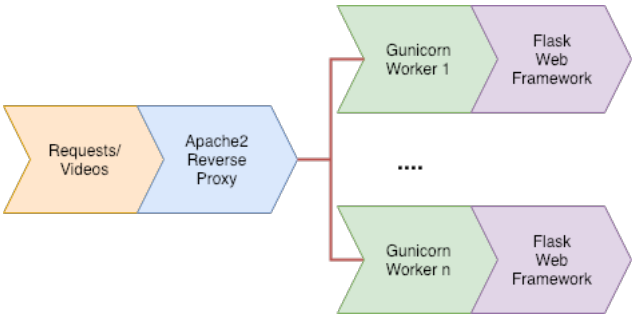


FIGURE 4.16: Flow chart of serving requests

4.7 Client Platform (Android)

4.7.1 Server Communication

The communication between the mobile app and the server is conducted using HTTP requests. OkHttp[12] is an efficient HTTP client, as it supports connection pooling so that connection can be reused for other requests to reduce latency, hence it is used throughout the client program. The communication includes sending POST requests to upload the video to the server, and GET requests to download the video from the server after it gets processed.

After user selects the video to be uploaded from gallery, the video gets converted to Java class `InputStream`, after which is further converted to a byte array. The byte array for the video is then attached to the request body of an asynchronous POST request. When the upload completes, server would respond to this POST request in JSON format, where "progress" specifies the url link to check for completion of the processing, "skeleton" to access the skeleton data from Pose Estimation module, and "vid_name" to the name of processed video. The following is an example response to an upload event:

```
{
"progress": "http://51.140.44.139/progress/figure/tennis/p_t1.mp4.log",
"skeleton":"http://51.140.44.139/json/tennis/p_t1.json",
"vid_name":"p_t1.mp4"
}
```

Originally, server responds after the processing has finished, and client has to maintain the connection throughout. This approach was then replaced due to the long processing time: a 13-second video would take well over 5 minutes to finish, whereas sending the video to the server is done in a matter of seconds. Maintaining such HTTP connection that is idle in most of its lifespan is not resource-conservative, the current approach, therefore, is used and works as follows. Client sends GET request to the link in attribute "progress" periodically, and would get back a floating-point number ranging from 0 to 1.0 representing the progress of the processing. Client will not start retrieving the video specified in attribute "vid_name", until the number reaches 1.0.

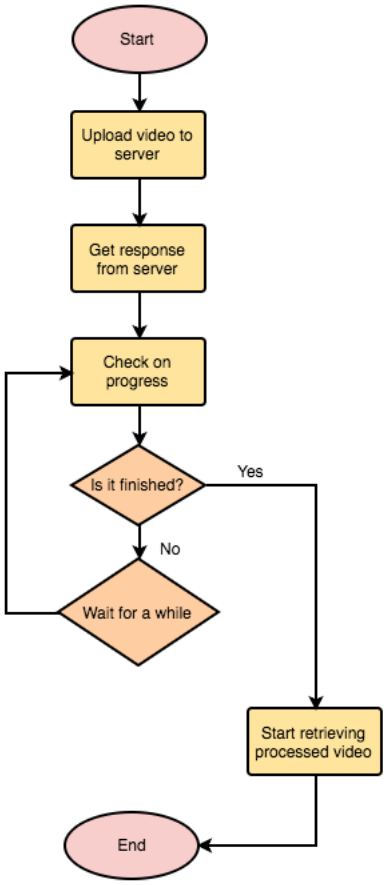
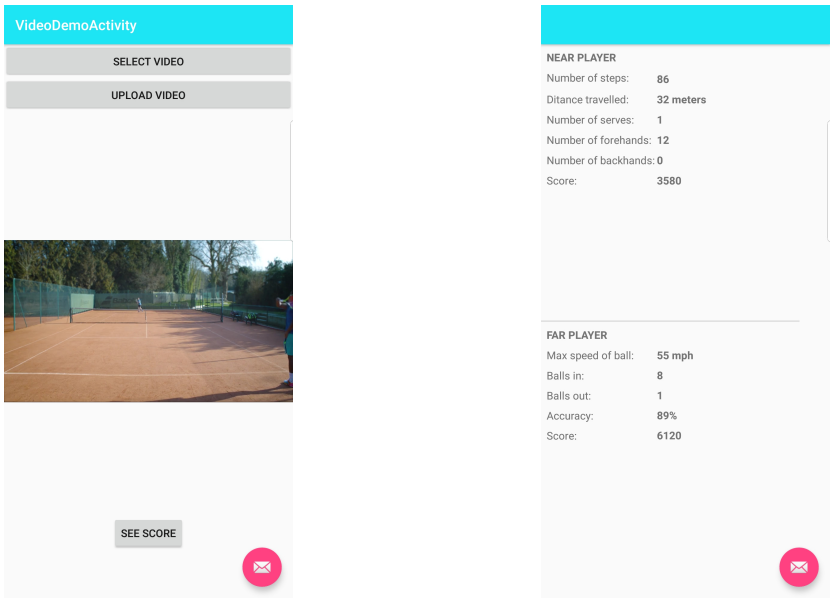


FIGURE 4.17: Flow chart of client communicating with server

4.7.2 User Interface

The two screen-shots shown above are the prototypes of user interface. It supports selecting video from the gallery and uploading it to the server. After the result transmits back to the app, the media player changes the original video to the processed version, with skeleton, court, ball and also metrics annotated in the video. The user is then able to click on the "SEE SCORE" button to go into the metrics interface, shown in FIGURE 4.13(B), to view the overall statistics.



(A) Video select and upload Interface (B) Metrics display interface

FIGURE 4.18: User Interface on Android

Evaluation

5.1 Testing Procedures

Each algorithm works independently which allows us to test them separately. System testing is performed on Android Studio with actual mobile devices connected to ensure the server and client work within the system as a whole. Department of Computing provided us with a Samsung S6 and a Samsung S8. With both phones we could check for any OS compatibility issues. See Appendix A for a snippet of Logcat file saved during a successful system test. Since the end results to all detection algorithms are in video content, the evaluation of the algorithms is conducted in the integration test before the system test.

In order to visualize the results of our algorithms we drew an overlay directly on the frame being analyzed. LV8Sport helped us greatly by sharing with us a bank of tennis videos and they also recorded with a smartphone camera tennis drills as we asked them to test specific scenarios, environments and edge cases. This saved us a great amount of time that we could dedicate to other tasks instead.

5.2 Quantifying Success

5.2.1 Court Detection

On the first video filmed with a smartphone, out of 40 court lines detected, 6 were wrong. That is an accuracy of 85%. All of those 6 sets of wrong court lines were flagged and the court lines from the previous frame were used instead. On the second video also filmed on a smartphone, out of 80 court lines detected, 9 were wrong (accuracy of 88%) but all correctly flagged as wrong and replaced with the previous frame's result. From Titcombe's report, the running time from computing average brightness to finding court lines' coordinates is 0.0467 ±0.0005s. This portion of code excluding computing average brightness (which takes virtually not time) is run every time the threshold for the white mask needs to be relaxed, which can be from 0 to 60 times (60 is an empirically chosen upper limit). Hence, finding the court lines for a single frame can take from about 0.0467s to about 2.802s.

5.2.2 Ball Tracking & Hit Detection

We tested our algorithm on a video of a single tennis rally and the results are generated empirically. Occasionally, the algorithm fails to track the ball or hit but if

anything is detected it is generally close to ground truth:

Bounce detection actual frame compared to detected frame											
	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th
Actual	76	166	268	355	447	538	626	728	817	903	986
Detected	84	165	268	354	446	537	623	730	816	none	none

Hit detection actual frame compared to detected frame												
	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th
Actual	0	93	186	278	373	466	551	644	746	833	920	1000
Detected	none	93	186	278	373	466	none	644	747	833	920	1001

Reflection & Future Extensions

6.1 Reflection & Future Extensions

6.1.1 Time consuming challenges

One entirely new challenged we faced was communication with the client. At the beginning of the project we had spent a lot of time and effort on converting the project to TensorFlow Lite, which is necessary for an offline machine learning project. We concluded after the first milestone that it was unrealistic and we lacked components and time to go through with the idea, but we did not know how to present it to our client. The project would have benefited from deciding it before, and negotiating with the client a cloud solution earlier. We were advised during our Software Consulting session to present our issue to the client in the most honest way and suggest a sub optimal solution to it. We would keep a couple of team members trying to figure it out while the rest of the team could go on and develop the next features. LV8Sport team was very comprehensive and we arrived to an agreement, our project would use a server for online computations. This has the disadvantage to consume bandwidth to our users but it allowed us to move on and create a viable product in the end.

6.1.2 Improving Algorithms

Our tool relies on four artificial intelligence related algorithms: pose estimation, court detection, ball tracking and shot recognition. They are state-of-the-art for open-source projects but further improvements in accuracy and computation speed could be made given more time.

Court detection would greatly benefit from a better white mask. We improved the color threshold at which a pixel is considered white but the fewer frames wrongly detected are mainly because of a threshold too restrictive. The second base line is in general less bright and can be missed by the white mask.

6.1.3 Improving User Experience

Our main priority for the project was to deliver top-performing algorithms for a minimum viable product. UI design could be reworked, enabling users to click fewer times, ideally less than three times for any core feature such as filming a drill. User experience features could be added to incentivize tennis players to improve more. Possible features could be to track one's progress over time and display it in digestible graphs and histograms, to have sharing options to show one's drill to

other users or even to have a leaderboard of the best performances.

Another way to improve the user experience would be to remove the bandwidth consumption due to computations run on the server. At the moment the main obstacles to analyze the videos directly on mobile are the algorithms using the Tensor Flow API. Tensor Flow offers an option for machine learning projects embedded on Android system called Tensor Flow Lite. The technology is very recent, thus has some bugs, lacks documentation and requires the latest phone models. However, in the near future it should be possible to port those algorithms directly into an Android Studio project which would enable the app to work fully offline.

6.1.4 Server Improvements

In order to further leverage the computing power and gain improvement in processing speed, we may divide a single video and run the algorithm on each fragmentation, furthermore, as the scale of users increases, it is preferable to run distributed system on the backend to reduce the processing time.

Other than reducing time for analysis of videos, it should be preferable to use some CI tools to minimize risks of server failure caused by a specific commit and pack the whole environment into a docker image in order to reduce costs of migrating the server.

6.1.5 For Other Sports...

Tennis was chosen to showcase how technology can become a sport analysis tool. Two other teams were working with our client LV8Sport as well and created a digital assistant for fitness and football. Some algorithms are needed to analyze most sports, for instance pose estimation, and thus can be reused directly. Some algorithms are useful in a more narrow domain, most sports using a ball can use the ball tracking algorithm, and can be reused for those specific sports. Finally, some of our algorithms need some tweaking before being used, for example the court detection needs to be slightly adjusted to work for other sorts of courts. Most of the technology has been built and at most a couple of algorithms need to be specifically written to create an analysis tool for a completely different sport.

6.2 Ethical Considerations

6.2.1 Data Privacy

We are very concerned about data privacy. All the data we are using is the videos sent by the users. They upload themselves each video they capture, thus they are consenting to the app analyzing it. We do not use any other user data.

6.2.2 Environmental Impact

As we use a GPU server, this is not most environmental friendly application, but it is the only way to run the machine learning models. The maximum consumption is at 149 W/h and the average is around 100W/h when used. To make sure our server is not too resource-consuming, we turn it on only when needed.

6.2.3 Inclusive Design

We made sure the app can be used equally well by any tennis enthusiast. The language of the app is English, which allows a large portion of the world population to use it, moreover the user interface is very intuitive and even non-English speakers could use it easily. Using our application does not require any expensive investment. Tennis players need to have an Android phone, which is about 88% of all phones, allowing as many people as possible to use the app. Tennis players should already have a racket, tennis balls and access to a court so that is not a further investment required to use our technology. People with disabilities who can play tennis can also hold a phone to record a tennis drill. Hence, our app supports equality by being accessible to anyone playing tennis, regardless of social status, gender, ethnicity, language, or disabilities.

Project Management

7.1 Methods

7.1.1 Agile Software Engineering

Out of the three agile methods introduced in the Software Engineering course, we decided to follow Scrum [13].

Scrum is more of a product management method when put alongside with Extreme Programming(XP) and Kanban. Scrum members have designated roles during the process, where Scrum Master monitors the development and ensures that the planned features are correctly delivered at the end of each Sprint; Product Owner decides the direction of development process and specifies features to be built, acting as a bridge between engineers and customers. Since we are collaborating with an organization outside the campus, following this technique best helps align the goals of stakeholders with our development: one of our team member, Pinchu Ye, was assigned the Scrum Master role and Matt Willcocks, the founder of LV8Sport, will be the Product Owner leading the weekly meetings every Wednesday. Each sprint lasted two weeks, to fit nicely with each milestone.

In order to keep everyone on the same page, daily scrums were held either in labs or over Slack [14] to recap what has been finished and the current obstacles, and to set out works for the following day. A Slack team was setup on the website with a channel for our group and the client. Another channel was used to communicate between our group, the client, and the other two groups doing fitness and football. This second channel's purpose is to coordinate tasks common to the three groups such as pose estimation and creating the server. We used the online Trello board [15] for item backlogging that we updated at the meetings to help visualize the velocity of development. Each sprint ended with a sprint review and sprint retrospective that will help us prepare the milestone review by our supervisor every Friday. Sprint review lasts two hours and we review what has been completed and prepare the demo for the stakeholders. Sprint retrospective lasts about an hour and a half and we identify and agree on continuous process improvement actions.

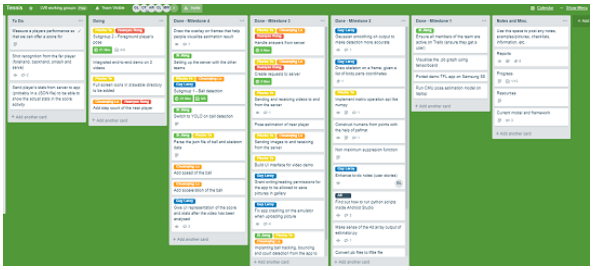
The Trello board offers us a way to decompose tasks into sub-tasks and track their progression. Each task is attributed tags indicating which members of the team are responsible for it. We introduced the following columns to separate each step of a task's cycle:

- To Do: tasks we plan to complete but that are not started yet, either because they are not to be prioritized over current tasks or because they require some

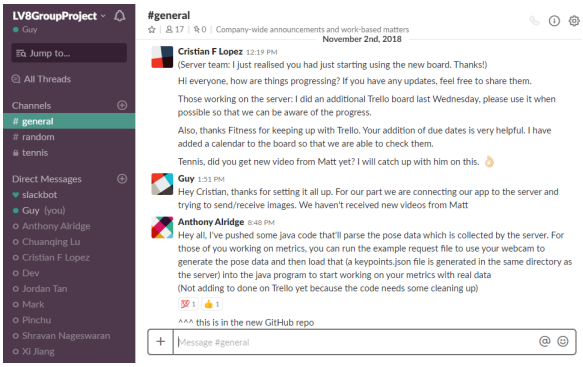
other tasks to be completed beforehand.

- Doing: tasks that some members of the team are working on currently.
- Milestone 1: tasks completed during the first milestone.
- Milestone 2: tasks completed during the second milestone.
- Milestone 3: tasks completed during the third milestone.
- Milestone 4: tasks completed during the fourth milestone.

Separating items done in each milestone helps us track our progression over time and deliver more precise reports. There often is a logical sequences of tasks over several milestones.



(A) Online Trello board used for item backlogging.



(B) Slack page to communicate with all concerned parties.

FIGURE 7.1: Slack and Trello board, two tools we used to work more effectively.

7.1.2 Version Control & Continuous Deployment

The main project is stored on Github [16] repository owned by another group that shares the same project, and since we have a different set of metrics to calculate, we created a separated Gitlab [17] repository for the metrics algorithms. Upon each push events on Github and Gitlab, the server will be notified and automatically update itself to the latest commit.

Different branches were used to keep the master branch bug free when implementing a new feature. Our project made use of continuous integration, this means we regularly merged our branch with the master one (a couple times a week), so that

everyone was writing code upon the latest working changes.

7.2 Planning

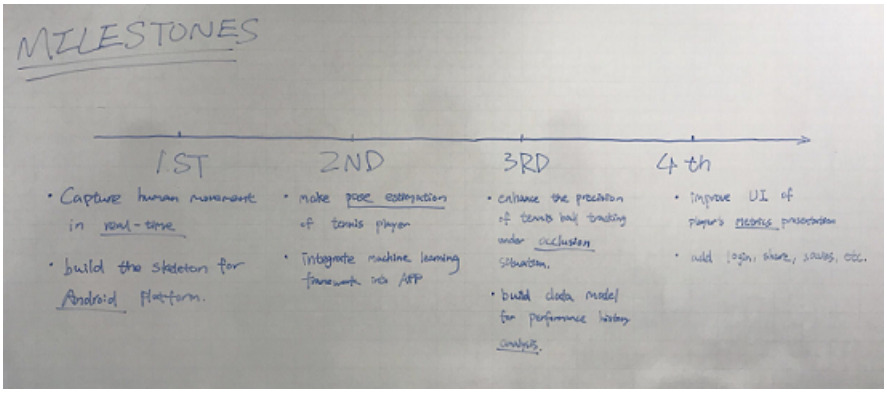


FIGURE 7.2: Sketch on blackboard of the work to complete for the milestones ahead during one of the first meetings.

Every two-week iteration, we presented our supervisor two new important features, as well as the updated project plan with work completed, the updated list of risks, the Responsible Technology worksheet, and the user feedback gathered. Before starting the project we planned the following:

	Plan outline
Milestone 1	Build skeleton of the app Capture video in real time from the app
Milestone 2	Integrate machine learning frameworks (such as TensorFlow Lite) into Android Studio Pose estimation working Enhance the precision of the tennis ball tracking under occlusion situations Build data model for performance history analysis
Milestone 3	Improve the User Interface (UI) to display players' metrics
Milestone 4	Add social features such as login, share, saves, import and export

TABLE 7.1: Rough plan outline at the beginning of the project, before further research.

As one can see, the plan changed significantly. We re-evaluated the short and long term goals regularly. Thus, throughout the milestones the project's tasks were updated due to unforeseen constraints and new ideas. In the end, what we achieved for each milestone is the following:

	Completed
Milestone 1	Research on pose estimation models Research to convert TensorFlow projects to TensorFlowLite.
Milestone 2	Implementation of multi-person pose estimation
Milestone 3	Setup server to run computations and handle requests/responses Implement ball tracking Implement court detection
Milestone 4	Enhance UI to visualize the models' output Polish previous models and combine them all Track several metrics and display them

TABLE 7.2: Main tasks completed at each milestones.

Android Logcat Snippet

```
Call actual request
Response from server received
Filename: p_1.mp4
http://51.140.44.139/progress/figure/tennis/p_1.mp4.log
Request successful
Not started yet, response is: file not exist or the processing hasn't
    ↪ started (/var/www/posegpu/storedData/tennis/p_1.mp4.log)
http://51.140.44.139/progress/figure/tennis/p_1.mp4.log
Request successful
Percentage at: 0.029375765
http://51.140.44.139/progress/figure/tennis/p_1.mp4.log
Request successful
Percentage at: 0.14810282
...
Percentage at: 0.8604651
http://51.140.44.139/progress/figure/tennis/p_1.mp4.log
Request successful
Percentage at: 0.93268055
http://51.140.44.139/progress/figure/tennis/p_1.mp4.log
Request successful
Percentage at: 0.998776
http://51.140.44.139/progress/figure/tennis/p_1.mp4.log
Request successful
Percentage at: 0.998776
http://51.140.44.139/progress/figure/tennis/p_1.mp4.log
Request successful
Percentage at: 1.0
p_1.mp4
Start retrieving video
buffer(okhttp3.internal.http1.Http1Codec$FixedLengthSource@310ae70).
    ↪ inputStream()
Start buffering to file
CREATED A TEMP FILE
File already created in Project root directory/data/user/0/com.elavate.
    ↪ test/filestemp.mp4
Buffered to file
Change videoview
```

Bibliography

[1] Bloomberg L.P. *This 200 AI will end tennis club screaming matches*. URL: https://www.bloomberg.com/news/articles/2017-03-02/this-200-ai-will-end-tennis-club-screaming-matches (visited on 01/03/2019).

[2] Hawk-eye innovations. *Hawk-eye innovations tennis*. URL: https://www.hawkeyeinnovations.com/sports/tennis (visited on 01/03/2019).

[3] Zhe Cao et al. "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *CVPR*. 2017.

[4] *Optimizing for mobile | TensorFlow Lite | TensorFlow*. URL: https://www.tensorflow.org/lite/tfmobile/optimizing (visited on 01/03/2019).

[5] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: http://doi.acm.org/10.1145/358669.358692.

[6] Intel Corporation. *Hough Line Transform*. URL: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html (visited on 01/03/2019).

[7] Silvia Vinyes Mora and William J. Knottenbelt. "Deep Learning for Domain-Specific Action Recognition in Tennis". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017*. 2017, pp. 170–178. DOI: 10.1109/CVPRW.2017.27. URL: https://doi.org/10.1109/CVPRW.2017.27.

[8] Intel Corporation. *Smoothing images*. URL: https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html (visited on 01/03/2019).

[9] J. Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (Nov. 1986), pp. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851.

[10] Benoit Chesneau. *Gunicorn - Python WSGI HTTP Server for UNIX*. URL: https://gunicorn.org/ (visited on 01/03/2019).

[11] Armin Ronacher. *Flask (A Python Microframework)*. URL: http://flask.pocoo.org/ (visited on 01/03/2019).

[12] *An HTTP HTTP/2 client for Android and Java applications*. URL: http://square.github.io/okhttp/.

[13] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN: 0130676349.

[14] Slack Technologies. *Slack*. URL: https://slack.com/ (visited on 01/03/2019).

[15] Atlassian. *Trello*. URL: `https://trello.com` (visited on 01/03/2019).

[16] Github. *Github*. URL: `https://github.com/` (visited on 01/03/2019).

[17] Gitlab. *Gitlab*. URL: `https://about.gitlab.com/` (visited on 01/03/2019).