UNIVERSITAT DE BARCELONA

TREBALL DE FINAL DE GRAU

Table Tennis Ball Tracking and Bounce Calculation using OpenCV

Author:
Christian José SOLER

Supervisor:
Dr. Eloi Puertas

June 22, 2017

Universitat de Barcelona

Abstract

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Double Degree in Mathematics and Computer Science

Table Tennis Ball Tracking and Bounce Calculation using OpenCV

by Christian José SOLER

Table tennis is an extremely fast sport which makes it hard for coaches to supervise many trainings at the same time. In this dissertation, a system is presented to solve that problem using a camera over the ping-pong table and a computer connected to the camera to process that information. This system outputs all gathered data of the match which are the trajectory the ball followed and the places where it bounced. The constructed system finds the ball's trajectory satisfactorily, however not its bounces, mainly because the implementation was done using a camera with low frame rate and resolution.

Acknowledgements

I would like to first thank my professor, Dr. Eloi Puertas, for giving me the opportunity to participate in this project. It has been what I enjoyed most doing during the whole degree. It made me realize computer vision could be my vocation as a future CS and Math graduate.

I'd also like to thank the members of CAR for facilitating all the tools for this project and for letting me visit CAR whenever I needed to record videos or to calibrate the camera. I sincerely hope my work in the future will help you in your trainings, Ramon.

Moreover, I'd like to thank my friends and roommates for supporting me throughout the project and for listening to me every time I needed to do rubber duck debugging to solve problems I encountered during this project.

Contents

Introduction

Table tennis, or simply ping-pong, is a sport that has been around since the 19th century. Over time it became more and more popular and as many other sports, it started requiring scientific and engineering tools in order to improve the athletes' performance, the arbitration and the view of the spectators.

In this sense, the institutions and organizations that sponsor this sport have taken care of the improvements that can be done on this elements to make the sport more professional to the players, referees and observers. The Centre d'Alt Rendiment (CAR), being the first sport center in the world that let athletes keep on with their studies while training, is within this same line and has incorporated scientific and engineering approaches for this objective. Located in Sant Cugat del Vallés, near Barcelona, CAR has a sports research center that works in many projects that mainly allow to record the performance of the athletes while they are doing the sport activities, for example the development of an autonomous device for the technique analysis of aquatic sports.

Trying to implement technological solutions to the problems and demands that are emerging from the different sports, arises a specific need in table tennis: the coaches need to supervise multiple tables at the same time which is practically impossible given the distance they have to walk between each table and how fast players play. CAR realized that they need some form of engineering in order to devise a system that fulfills their requirements, and have been some previous attempts in trying to address that issue like the MIT's PingPong++ project, but although it was adapted for CAR's necessities, it failed because of physical limitations. That is why it is proposed another solution: A camera-based approach that grabs the information of the game using a camera in order to do statistics with that information.

The main purpose of this TFG then is to devise a system from the viewpoint of computer vision and image processing to help table tennis coaches supervise multiple tables at the same time by showing training statistics for each table connected. This system would not only be useful for coaches, but also for athletes who could configure training programs for themselves and see how they did.

However, given this project is limited in time, the plan is to perfect the data gathering of the ping-pong games as much as possible, in order to do statistics later on. This data mostly includes the trajectory of the ball and its bounces. Ideally, the output will be a heatmap of bounce areas shown on a TV updated in real time while players are playing.

As for the structure of this dissertation, first an analysis of the surroundings of our problem will be done. From here a system design will be extracted and explained in high-level to solve the problem of data collection of a ping-pong match.

This design will be then implemented and presented to the reader in a more low-level format. Then, to see how accurate the algorithm is, some experimental results will be shown and conclusions will be drawn from these in the end.

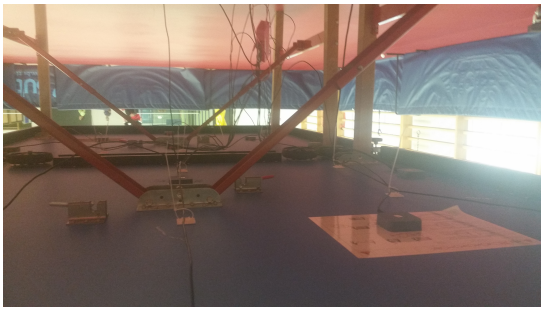
Analysis

The main preceding of this project is PingPong++ (PingPongPlusPlus) from MIT. In this project, they implemented PingPong+ from 1998 using open-source hardware and software interfaces so that anyone could continue creating their projects based on it.

In essence, both projects' aim is to see where the ball bounces using 8 microphones. Since there can be no objects on the table, these microphones are placed below the table and are coordinated so that they detect the bounce using the sound the ball makes upon hitting the table. This data is then recollected and can be represented on the table itself using the projector placed above of it. The MIT project also counts with a Visualization API to create new forms of visualization on the table and a Crowd Data API to do data analysis on the data collected by the games played on the interactive table.

At CAR, an implementation of PingPong++ was done to create a training program using this MIT project, as seen on figure 2.1. However, it failed mainly because of detection errors. One of the reasons having to do with how sound sensors work, since these sensors do a triangulation using the sound waves the ball emits upon hitting the table, which is extremely imprecise. Also, ambient noise, which is normal in a room with twenty ping-pong tables, can add new waves that sound sensors detect falsely. With these reasons that cannot be remedied, the detection by sound was discarded.

FIGURE 2.1: Sound sensors for PingPong++ implementation at CAR



The game of ping-pong itself consists of hitting a lightweight ball back and forth across a table to gain points. Two players (or two pairs of players, however, for this analysis two players will be assumed) use rackets in their hands to hit the ball from the side of the other player, who tries to hit it back before the ball bounces twice on his side or falls to the floor. If this happens, the player opposite to the side where the ball fell wins a point. Each match begin by one of the players serving and continues as explained. In case one of the players fail to hit the ball back, the other player gets

a point. The number of points to win varies depending on the type of match played, but typically are eleven.

As for the system that needs to be designed, since it revolves around ping-pong matches, not only does the design has to contemplate the flow of a ping-pong match but also the elements of the game must be presented: the table, the net on top of it, the ball and the racket.

The table, which must be rectangular with white borders, 2.74m long, 1.525m wide, and it must lie 0.76m above the floor. It is where most of the game occurs, so nothing can be on top of it except for the net, which is 15.25cm over the table and is white. The net is what divides the side of one player from the other one's. The net can be hit before entering the other player's side during the game, but not when serving and because of the ball's velocity, the net is deformed when hit. Since the net is so short, no camera can be put on top of it, because it would be hit by the ball.

The ball must be spherical, with 40mm of diameter and must weigh 2.7g. Since it is so lightweight, it cannot be painted or modified (like, for example, put a GPS on it), since that would change its physical behaviour. Also it technically could be another color, but at CAR white balls are used. It is the most dynamic element of the game and it is hit by the rackets that players hold to change its direction and send it back to the other player. It moves extremely fast because of its weight and the agility of players.

The racket can be of any size, color and shape, but its blade must be flat and rigid. Also, the 85% of the blade in thickness must be made of natural wood. Players use it to return the ball to the opponent. Players can wear any clothes (including white) and move fast around the table, which is why the system must be designed taking into account possible confusions with the ball.

The system's design will need to take into account not only the previous attempts' results but also the context of the ping-pong match itself, such as assuming the ball is white or that the ball usually moves from left to right or from right to left. In the next section, the design of a system will be presented taking into account all the analysis done beforehand.

Design

In this chapter, a high-level description of the system for the problem of ball tracking and bounce calculation is presented. The problem is tackled using image processing and computer vision techniques. If the reader is familiar with this kind of projects, then the system's design should be familiar too. Broadly speaking, images are produced by a device, the camera, which are then preprocessed to remove noise. The image's pixels are then treated as signals in order to extract some valuable information. In this case, postprocessing is also needed for data visualization.

The system's flux begins with the camera that produces frames to work with. The camera is located at a fixed position around 5 meters high from the floor to see as much as possible of the scene, since table tennis players tend to use up a plenty of space while playing. The camera is of 60fps, has a 1024 x 576 resolution and is connected by an Ethernet cable to a computer or a Raspberry Pi. Over the camera, there is a projector that would be used for showing the results of bounce coordinates. This is reflected in 3.1a.

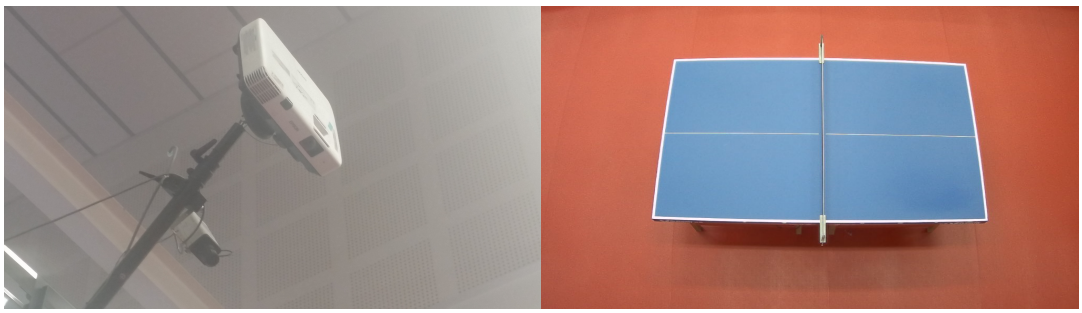
After the frame given by the camera is grabbed, it is passed on for preprocessing. The first frame is used to calibrate the camera, i.e. remove the fish-eye effect from it as seen in figure 3.1b, to detect the table and to discard all parts of the scene that could later on be confused by a ball. This last one is a countermeasure for false detections. For the rest of frames, this extracted information is used to correct and remove noise from the image.

After all the preprocessing has been done, the ball needs to be found. This is where computer vision and image processing comes in; techniques from these fields are used to extract this information. If the ball has been found on the given frame, then the system proceeds to bounce calculation which is based on the idea that the

FIGURE 3.1: Camera

(A) Camera and the Projector above it

(B) Image of the table from camera

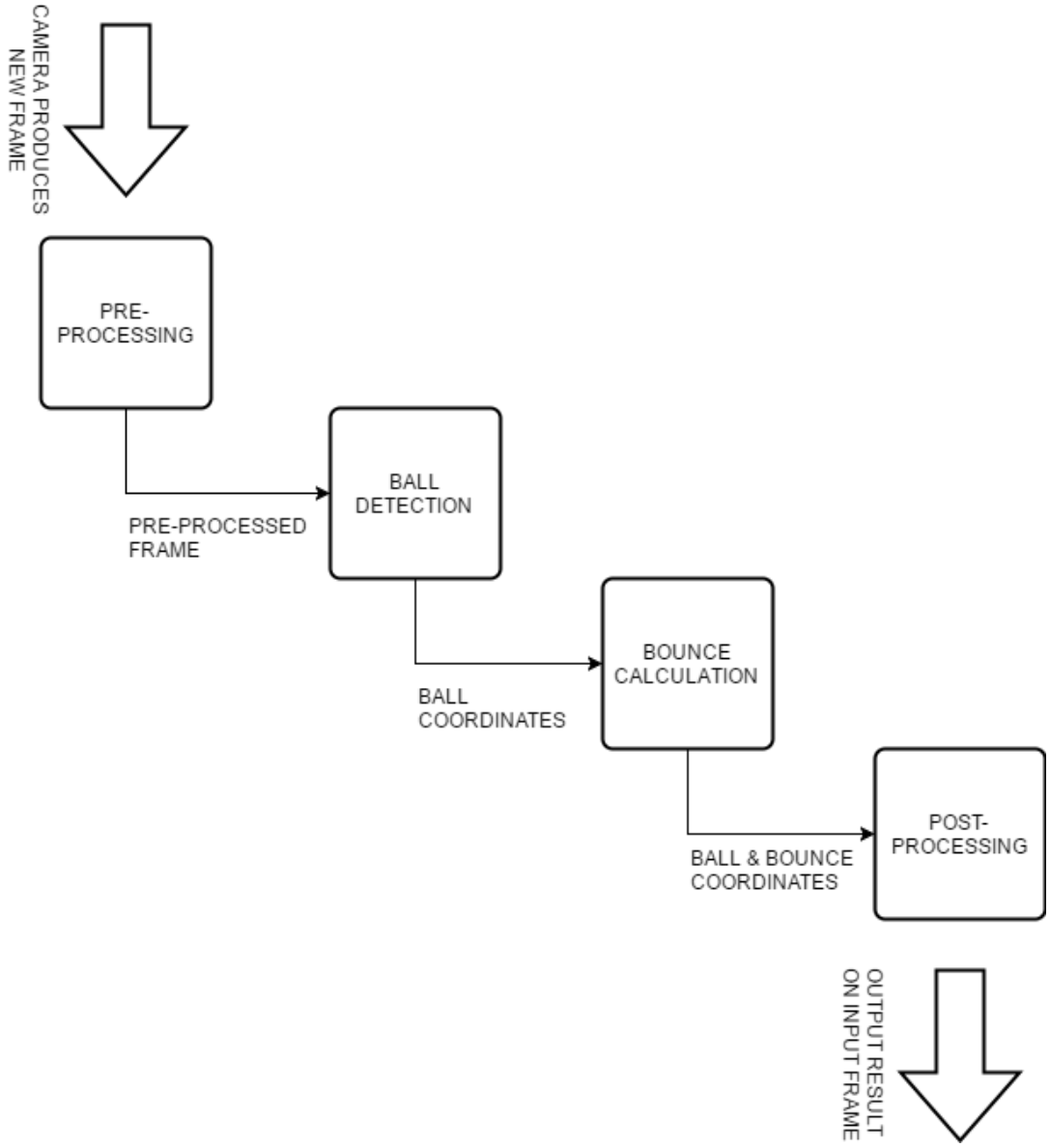


bounce will never be detected exactly since the camera has a finite sampling rate and table tennis is fast game; bounces will only be detected after they happened.

Using the information obtained by ball detection and bounce calculation, the frame is post-processed for data visualization. This is done real time and the result is shown on the original frame with fish-eye effect removed by painting the trajectory of the ball and marking its bounces. This post-processed frame will be shown on a TV for table tennis trainers to see.

This system design solves the problem of ball tracking and bounce calculation with the extra of data visualization to present the data to table tennis trainers. The flux explained beforehand is summarized in 3.2.

FIGURE 3.2: Flux of the system



Implementation

After having done the design of the algorithm, an implementation using Python with OpenCV and numpy libraries was done. This implementation is now presented to the reader. A single video was used to adjust parameters of the algorithm.

4.1 Preprocessing

Preprocessing is initialized using the first frame. It includes calibrating the camera, detecting the table and creating a mask to remove all initially white pixels from the scene.

4.1.1 Camera calibration

Images from the camera come with a distortion that needs to be corrected. The approach to achieve that is by training a camera calibrator and use that information to undo the distortion experienced.

The camera undistortion takes into account 2 types of distortion: radial distortion that corresponds to the fish-eye effect and tangential distortion which occurs because the image taking lenses are not perfectly parallel to the imaging plane. Tangential distortion correction crops the image. In order to correct radial distortion, the focal length, the principal point and the 3 coefficients of radial distortion need to be found. To correct tangential distortion, the 2 coefficients of tangential distortion need to be found.

OpenCV's camera calibration approach was followed. To follow this approach, captions of a 10x10 chessboard were taken on top of the ping-pong table at different positions as seen in 4.1. However, since OpenCV's chessboard detection functionality was not powerful enough for this project, after some investigation, MatLab's Camera Calibration module was found better and the camera calibrator was trained using 7 training images. An example of these training photos is presented in 4.1b.

FIGURE 4.1: Chessboard calibration

(A) Chessboard (B) Training image

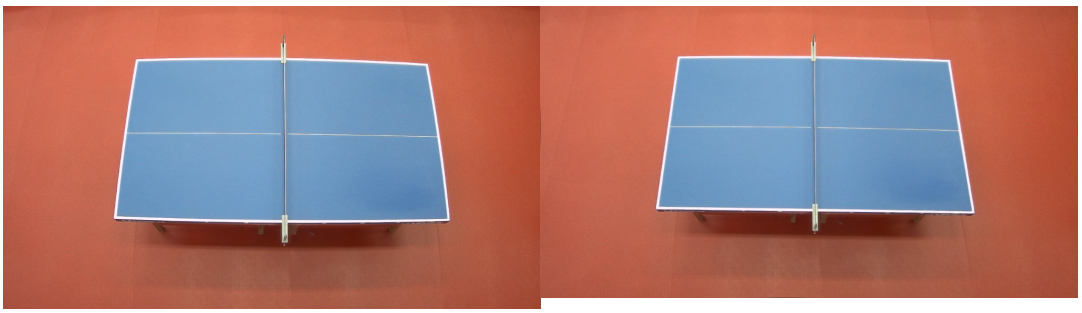
TABLE 4.1: Undistortion coefficients and error using MatLab

Focal Length	$(1.0424 * 10^3, 813.7234)$
Principal Point	$(485.1304, -89.7278)$
Radial Distortion	$(-0.3692, -0.0680, 0.1172)$
Tangential Distortion	$(0.1360, -0.0018)$
Mean Reprojection Error	18.27%

The image resulting from using these coefficients for undistortion is presented in 4.2. The reader might observe that the undistorted image is clearly smaller; that is due to the cropping process after correcting tangential distortion.

FIGURE 4.2: Camera calibration result

(A) Original (B) Corrected

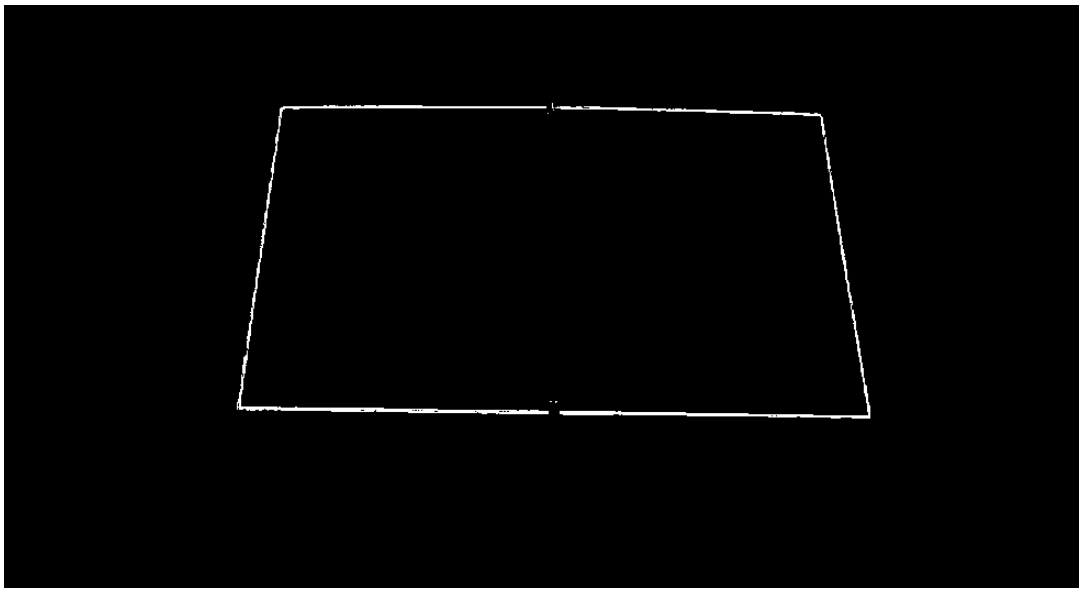


4.1.2 Table detection

As a pre-step of table detection, in order to prevent false positives in all detection algorithms, using the first frame all initially white regions are detected and removed.

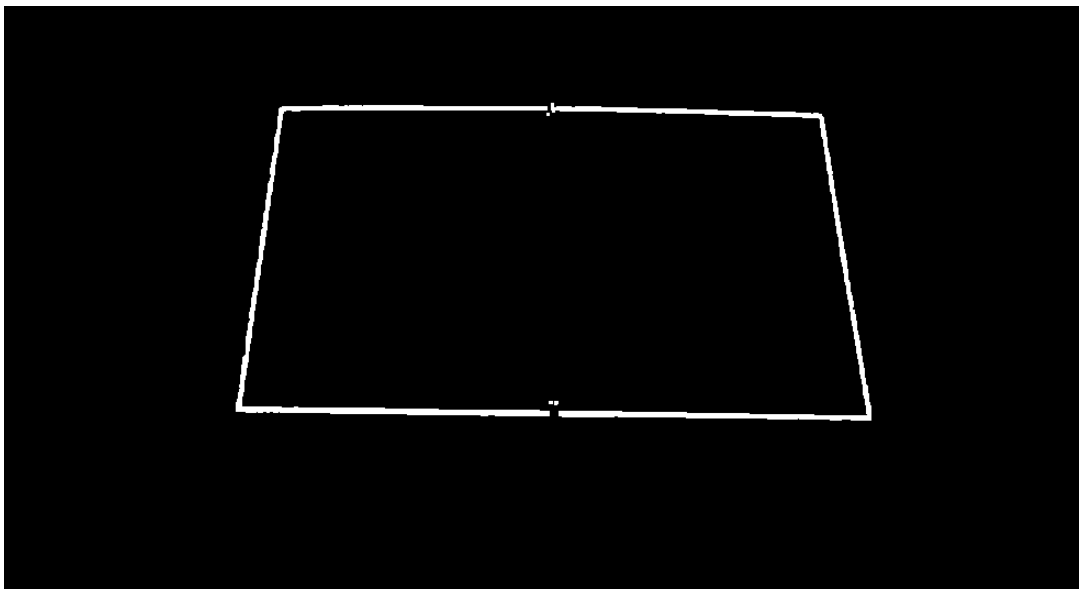
First, all colors are passed to HSV color space which enables working in a more continuous color space, unlike RGB. Let s be a variable that defines white sensitivity. A binary filter is passed on the image to remain only with pixels with colors between $(0, 0, 255 - s)$ and $(255, s, 255)$ in HSV space. In this application, $s = 15$. The result is only the white pixels of the original image.

FIGURE 4.3: Filter on white pixels



The resulting mask is then dilated to make the white region wider, as prevention. Let the resulting mask be M_{white} and its inverse $M_{\text{non-white}}$. The M_{white} is the region where the table will be detected and $M_{\text{non-white}}$ is where the ball search will be restricted; it prevents confusion of the ball with other white regions of the image such as the white line of the table or irregularities in the net.

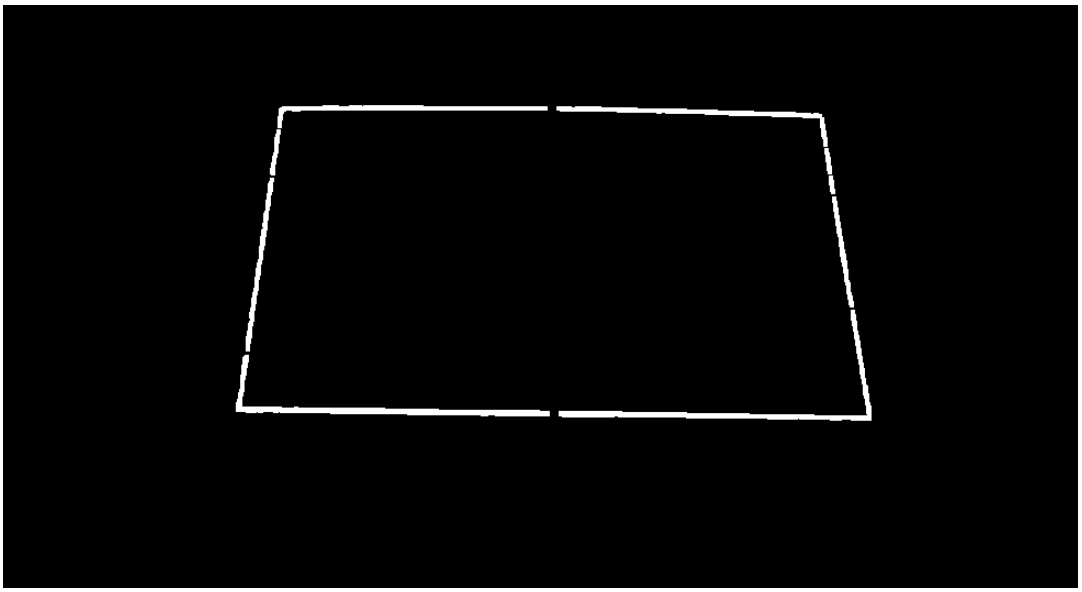
FIGURE 4.4: Resulting M_{white} after dilation



The inside and outside of the table need to be defined, which will correspond to the masks $M_{\text{in-table}}$ and $M_{\text{out-table}}$, respectively. To do that, the table's white border has to be found and its inner and outer lines need to be extracted.

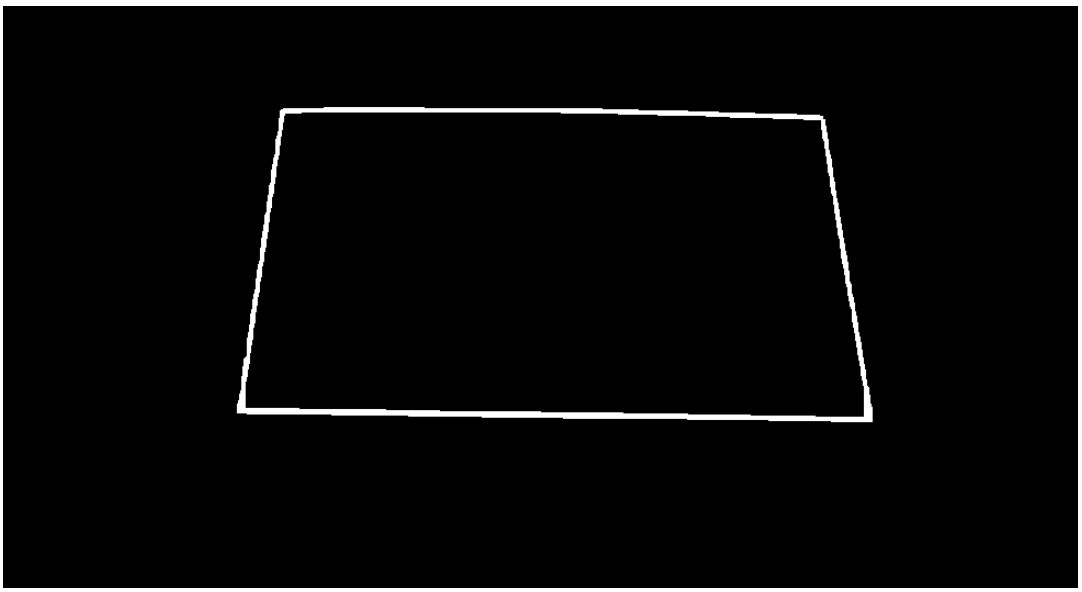
Observe on 4.4 that the contour of the table is not well defined yet, it still needs some more noise removal. The image is opened with a 4x4 kernel to remove the white region above the bottom hole which corresponds to a portion of the net.

FIGURE 4.5: Opening $M_{\text{non-white}}$



Notice on 4.5 that the white regions that do not follow the white line, disappeared. Now, using closing morphological operation with a 20x20 kernel the holes of the white line will be removed. 20x20 is needed, because the whole is of a considerable size.

FIGURE 4.6: Closing previous mask

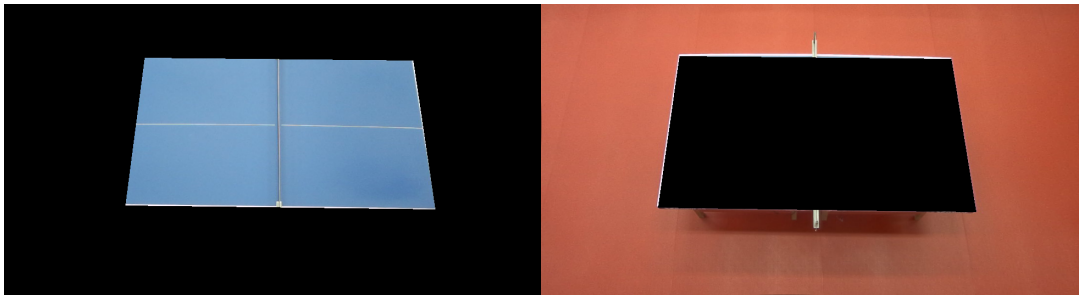


Having removed all noise from the mask, the approach to find the table borders is using contour search, where a contour is a boundary of a possible object. Using contour search with OpenCV's *findContours* function all the table's contours are found organized in hierarchies, because both lines of the border of the table belong to the same hierarchy. Since the table is the white rectangle with border of greatest area, then simply the outside of the border the table is the contour of greatest area of the mask and its inside is the second one. The masks $M_{\text{out-table}}$ (which represents the image outside the table's borders) and $M_{\text{in-table}}$ (that represents the image inside of

table's borders) are created from these results and shown in 4.7 applied to the initial frame.

FIGURE 4.7: Resulting table masks applied to first frame

(A) Mask $M_{\text{in-table}}$ (B) Mask $M_{\text{out-table}}$



4.2 Ball detection, Bounce calculation and Post-processing

In this section, the problem of ball detection and bounce calculation is tackled. Also, the results from this step are post-processed to visualize them to the user of the system.

The first step of the detection is seeing whether the ball is inside the table. Balls inside the table are assumed to be the white objects of greatest area. The detection space is restricted in various stages to improve precision of the detection. Frame number 328 is used as an example to simulate how the algorithm behaves when the ball is inside the table. It is clear on 4.8 that the ball is inside the table.

FIGURE 4.8: Frame 328 unprocessed

First $M_{\text{non-white}}$ is applied to the frame to remove all white pixels of background. It prevents confusion of the ball with other white pixels of the scene.

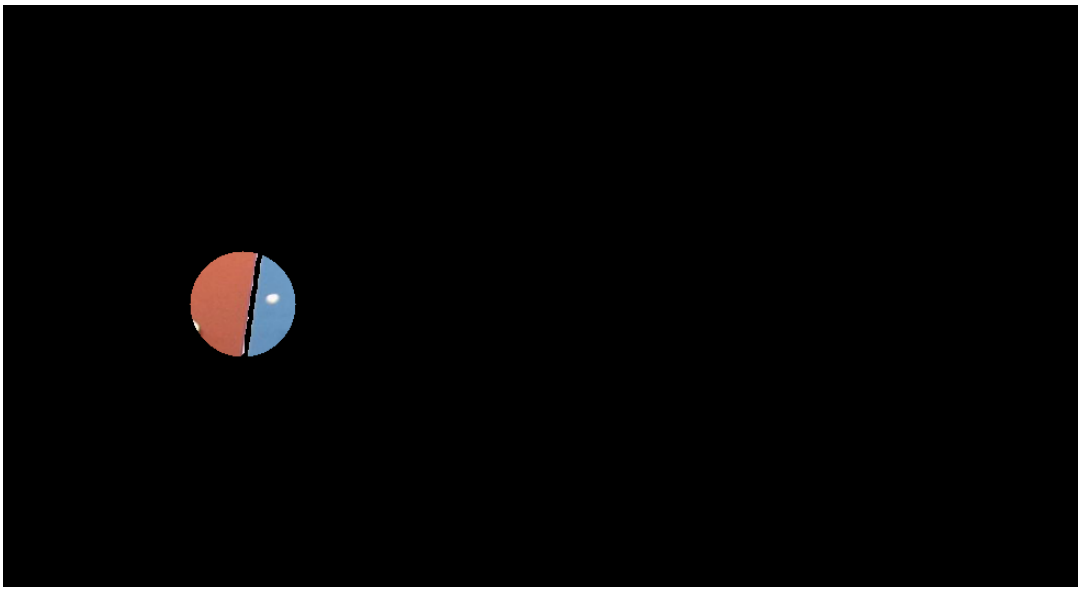
FIGURE 4.9: Removed all white pixels

Since the change between frames is almost instantaneous, the assumption that ball doesn't accelerate more than twice from one frame to the next one is made. That means that if there are at least two consecutive detections of the ball, then the next detection shouldn't be further than twice the distance between these two. This makes sure that once the ball has enough consecutive history, it is tracked properly without getting lost in detection elsewhere.

So the search of the next ball is restricted to the ball centered at the last detection, with radius twice the distance between the last two detections. Let the resulting mask be called $M_{\text{neighborhood}}$ as seen in 4.10. If the ball is not found in this neighborhood, it might mean that it is covered by some other scene object or the detection failed, however false positives will be avoided.

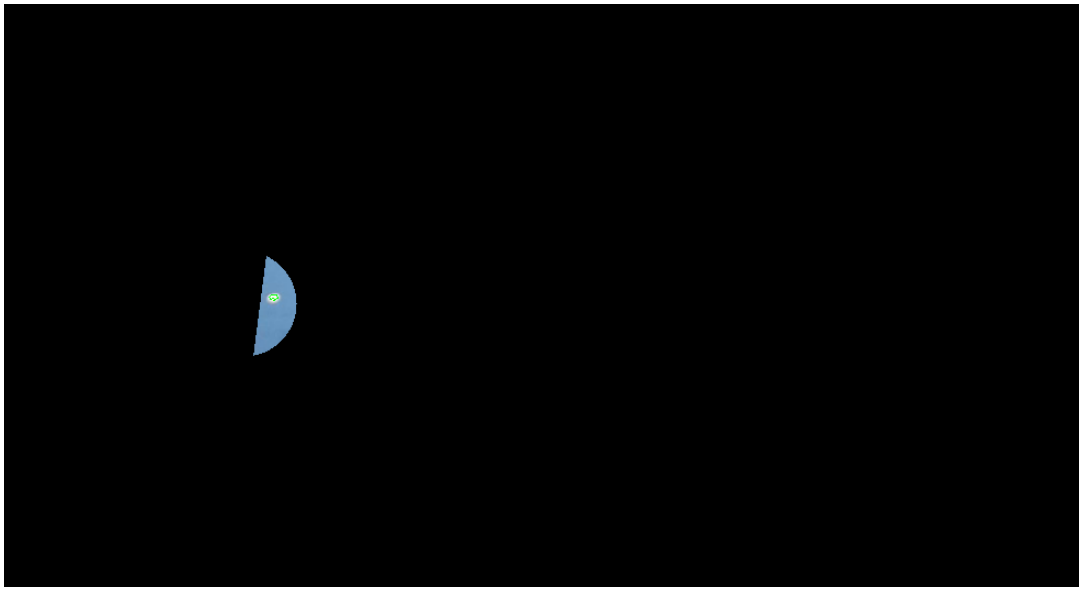
Note that this also improves mean performance, since it reduces detection space to a much smaller one if there is enough history, which should usually be the case.

FIGURE 4.10: Applied $M_{\text{neighborhood}}$



Afterwise, the $M_{\text{in-table}}$ is applied to the resulting image. Note how the part that is outside the table disappears with this mask. After having passed a white mask on the image, contours are searched in the image. The contour search is done using OpenCV's *findContours* library function to find outer layer contours, i.e. if a contour is inside another, then the outer one is returned. The center of the ball is the contour's center of mass. The green dot in 4.11 is the ball detected. Usually, it is the only contour found, but in case of ambiguity, the one of greatest area is returned.

FIGURE 4.11: Apply $M_{\text{in-table}}$ and search contours



If the inside search returned failure, the ball is checked if it is outside the table. The algorithm is different from inside search since if the same was used, other white objects of the scene would also be detected like shoes or T-shirts, which is not desirable. Another approach has to be applied. For this explanation, frame number 390 will be used. It is clear on 4.12 that the ball is outside the table. White pixels are first removed from background, but in this case, that does not affect detection, so it will not be represented.

FIGURE 4.12: Frame 390 unprocessed

Next, the same way as in inside detection, a suitable neighborhood around the ball is done taking into account previous positions applying $M_{\text{neighborhood}}$. Notice again that the ball is clearly inside this neighborhood. The frame is restricted to outside the table using $M_{\text{out-table}}$, but notice that it stays the same in this case.

FIGURE 4.13: Applied $M_{\text{neighborhood}}$

The next step is background subtraction, which is implemented using OpenCV's *BackgroundSubtractorMOG2* class with a history of 15 frames, a maximum distance of 9, and to not detect shadows. The distance used is between a pixel and the distribution of the pixels of the history of the background subtractor. The filter it returns is then passed through a median filter with 5x5 kernel to remove salt and pepper noise. The result is presented on 4.14.

FIGURE 4.14: Background subtraction without neighborhood

Experimentally, contour search was seen to not work as well outside the table as it does inside, which is why blob detection is used rather than contour search. A blob is a group of connected pixels in an image that share some common property. The blobs are filtered by colour, area, circularity, inertia and convexity. It is implemented using OpenCV's *SimpleBlobDetector* class with parameters: blobColor = 255 (white), minArea = 100, minCircularity = 0.75 (very circular object), minConvexity = 0.9 (very convex object), minInertiaRatio = 0.08 (object that is in movement). If multiple blobs were found with these characteristics, the whitest of them is returned. If there are no blobs that fulfill the specified requirements, this process will return failure.

FIGURE 4.15: Background subtraction with neighborhood and Blob search

Suppose now both inside and outside detection resulted in failure. In these cases, extrapolation is used. It is useful for predicting balls that were undetected either

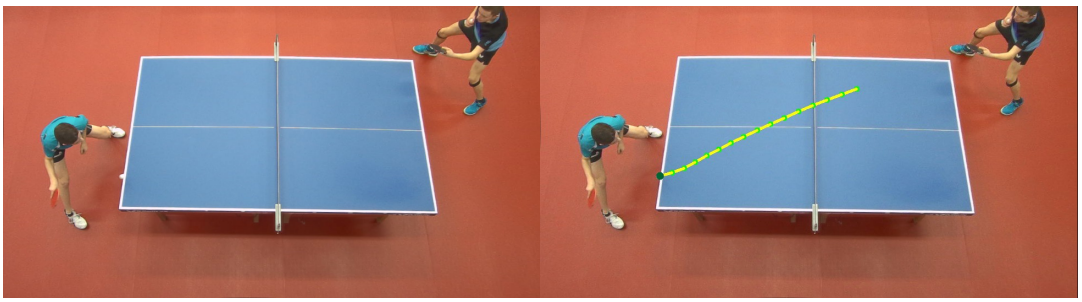
because they were covered by some scene object or because the previous steps had a miss. Extrapolation cannot be done if there are not enough non-extrapolated values in history, which for this application is four. The main idea behind extrapolation is that if the ball follows a parabolic trajectory, it is reasonable to think that if a parabola is fitted on the previous ball positions, then the next position of the ball should lie on that parabola.

More precisely, a parabola P is fitted on last 4 ball positions. Let (x_1, y_1), (x_2, y_2) be the coordinates of the last and before last positions of the ball, respectively. Define $\Delta x := x_1 - x_2$ and let $N = x_1 + \Delta x = 2x_1 - x_2$ be the translation of the x component of the last ball with velocity Δx. To calculate the extrapolated position, P is evaluated at N, which gives $(N, P(N))$ as predicted ball position. Moreover, in order to remain realistic, extrapolation is not done more than once in a row, since the behaviour of the ball can change rapidly. Already extrapolated values are not going to be taken into account in order to extrapolate either, however they are used for bounce detection, since it is one of the extrapolation's main purposes.

An example is shown from frame 638. The ball is not found because it is over the white line, which is one of the main reasons the extrapolation was created. In 4.16 the dark green ball is the extrapolated ball.

FIGURE 4.16: Frame 638 before and after post-processing

(A) Unprocessed (B) Processed



After a new ball detection, the coordinates of the ball are passed on for bounce test i.e. the ball is checked for being post-bounce or not.

Bounce is assumed to be the moment when the ball's trajectory moves from one parabola to another, without changing direction. With this in mind, a parabola is fitted on some of the last centers and seen if the ball's trajectory could possibly be parabolic. If it cannot, it means that the ball already started following the next parabola, so it must have bounced in between. More precisely, a parabola is fitted on the last 4 centers (since a parabola can always be interpolated on 3 points or less) and if the quadratic error of that parabola is higher than a threshold, in this application is 0.6, then a bounce happened.

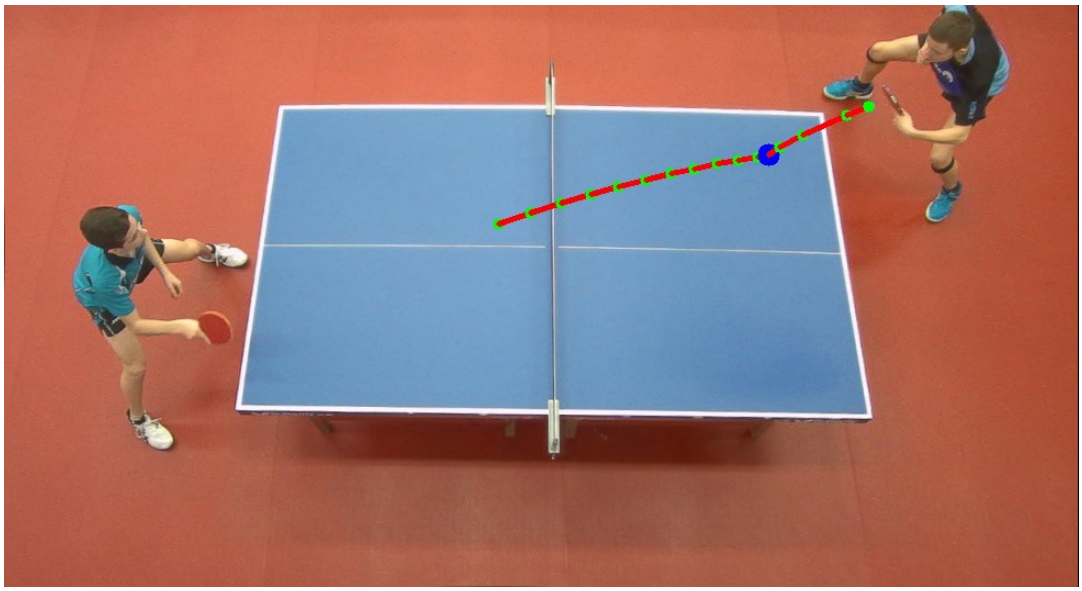
To calculate the position of the bounce, since the bounce test is done after each consecutive detection, then it is confirmed that the bounce happened between the last and before last of the balls of history. If the bounce happened before, the bounce calculator would have seen it in a previous post-bounce check. With that in mind, let (x_1, y_1), (x_2, y_2) be the coordinates of the last and before last positions of the ball, respectively. Now let $M = \frac{x_1 + x_2}{2}$, be the x coordinate midpoint of these.

Then the position of bounce when detected is calculated by fitting a parabola P the last 4 balls excluding the last one, since the last position is after the bounce and is

part of the new trajectory of the ball. Then, this parabola P is evaluated at M, which gives $(M, P(M))$ as predicted bounce coordinate.

Also, when a bounce happens, the history is erased except for the ball after the bounce, since the trajectory is a completely new parabola. In 4.17, the blue dot represents the found bounce coordinates.

FIGURE 4.17: Bounce calculation



Post-processing is done then using the obtained ball and bounce coordinates. These are painted on the undistorted frame, as seen on 4.16b and 4.17. The light green dots correspond to ball positions, red or yellow line to the trajectory of the ball (color indicates if it goes left to right or right to left), dark green dots correspond to extrapolated balls and big blue dots to bounce detection.

Originally, heatmaps of bounce frequency were planned to do, but this representation resulted being much more visual for testing the accuracy of the algorithm, which is what the next section will be all about.

Experimental results

The system has been implemented, however, its accuracy needs to be measured. By accuracy the ability to guess on which frame the events happen is meant, not the precision of the exact coordinates. The latter would be too inaccurate to test manually.

The software has a testing option for this use case so that given the list of frames numbers where each event occurs, it calculates the number of false positives, false negatives, true positives and true negatives. In order to use the testing module, the whole testing video of 16421 frames has been parsed manually and written down all the frame numbers where bounces happen and the ball is in movement. The results are presented in the form of confusion matrix for each of them. In case of ball tracking, all frames were taken into account, however with bounce calculation only the ones where the ball is in movement, otherwise the results would be misleading.

n=16421	Predicted NO	Predicted YES
Actual NO	7513	257
Actual YES	1900	6751

(A) Confusion Matrix

Accuracy	86.86%
Precision	96.33%
True positive rate	78.03%
False positive rate	3.31%

(B) Rates

FIGURE 5.1: Test results of ball tracking

The confusion matrix 5.1a, in itself does not let us conclude whether the algorithm was successful, some meaningful rates using the confusion matrix are presented in 5.1b. The accuracy and precision are high, which is what a ball tracker needs to have. The false positive rate is low which is even more important, since if balls are falsely detected, then a false trajectory could lead to false bounce detections.

n=8651	Predicted NO	Predicted YES
Actual NO	8123	302
Actual YES	97	129

(A) Confusion Matrix

Accuracy	95.39%
Precision	29.93%
True positive rate	57.08%
False positive rate	3.58%

(B) Rates

FIGURE 5.2: Test results of bounce calculation

Again, the confusion matrix 5.2a alone does not give enough information, so

rates were calculated and presented in 5.2b. The accuracy is high and false positive rate is low, but these are meaningless in this dataset, since there are around 250 bounces for 8651 frames examined. As for the other two, the precision is low, however true positive rate is a 57.08% which is more than half of the real bounces detected.

They might seem bads results at first for bounce calculation, but since the ball tracker shows good results, it either means that the frame rate and resolution of the camera is too low or the algorithm of bounce calculation algorithm needs adjustments, however experimentally, it was clear that most of the bounces that went undetected were the ones that corresponded to faster balls, meaning there are less samples to calculate bounces from. Also, the 57.08% could be improved at the cost of losing precision; the decision was to try to remain around 30% of precision.

Conclusion

The proposed objectives for this TFG were: track the ball, calculate its bounces and generate a heatmap as output that is updated real-time with the most frequent areas the ball bounces.

The ball tracking has a high accuracy and precision which makes that objective fulfilled. However, bounce detection's results are not as satisfactory: The precision is low and the true positive rate is around 60%. Seeing that the tracking is satisfactory, the camera's low resolution and frame rate are blamed for that, because most missed bounces are when balls travel fast, i.e. trajectories that are made up of a small set of points and with a lot of distance between them because of the slow sampling rate. Also, given the camera is around 5 metres high, the small resolution makes detection more imprecise. As for the heatmaps, they have been substituted by painting the ball's trajectory and bounce on the original frame since they were more visual for testing. For the oral presentation though, heatmaps might be generated for better data visualization to the general public.

All in all, the predecessor of the system of training program has been created with one table and one camera above it. It is still of great help for coaches, since if the output is connected to the TV, they can see with some delay what happens on that table, i.e. what trajectory the ball follows and where are its bounces. It is still imprecise, but since the ball tracking is satisfactory, only the bounce calculation needs to be adjusted, which is a much more simplified problem.

As for the future works, there is still a lot to be done. First of all, the camera needs to be changed: it either has to be improved to have a higher frame rate and a better resolution, or another camera could be added in order to recreate 3-dimensional space to be able to find bounces by intersecting the ball's trajectory with the plane of the table. Moreover, the algorithm is designed to be real-time, however, the processing is around 5x slower than the original frame rate, which does not correspond to a real-time output; if the algorithm is remade using for example, CUDA, optimization could be done. Furthermore, the bounce calculation's quadratic residue threshold is static, it should depend on the curvature of the trajectory instead.

Once the bounce calculation and ball tracking are both satisfactory, the data obtained by the system can be used to obtain any statistics the coaches need for their analysis, for example, training programs could be made to practice a particular type of serving and then system counts the accuracy of the pupil with the exercise that was proposed, amongst many other possible training programs.

As soon as the system is perfected for one table, it could start escalating to more tables at the same time all with their own statistics of each exercise that could be shown to the tablet or phone of the coach so he can just go to the tables that really need help with the exercise.

Bibliography

[1] Docs.opencv.org. (2017). *OpenCV modules.* [online] Available at: http://docs.opencv.org/3.2.0/ [Accesed 1 Jun. 2017]

[2] Docs.python.org. (2017). *Python 3.5.3 Documentation.* [online] Available at: https://docs.python.org/3.5/ [Accessed 1 Jun. 2017].

[3] Docs.scipy.org. (2017). *Contributing to NumPy — NumPy v1.12 Manual.* [online] Available at: https://docs.scipy.org/doc/numpy-dev/dev/ [Accessed 1 Jun. 2017].

[4] Experimedia.eu. (2017). *Augmented Table Tennis | EXPERIMEDIA*. [online] Available at: http://www.experimedia.eu/2014/10/10/augmented-table-tennis/ [Accessed 1 Jun. 2017].

[5] Fctt.cat. (2017). *Federació Catalana de Tennis de Taula*. [online] Available at: http://www.fctt.cat/ [Accessed 20 Jun. 2017].

[6] Ishii, Hiroshi, et al. "PingPongPlus: design of an athletic-tangible interface for computer-supported cooperative play." *Proceedings of the SIGCHI conference on Human Factors in Computing Systems.* ACM, 1999.

[7] Xiao, Xiao, et al. "PingPong++: community customization in games and entertainment." *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology.* ACM, 2011.

User Manual

In here, the use of the program is explained. It is mainly a script that accepts certain parameters and has some package requirements.

A.1 Requirements

Python 3.5 with libraries (creation of virtualenv is recommended to run):

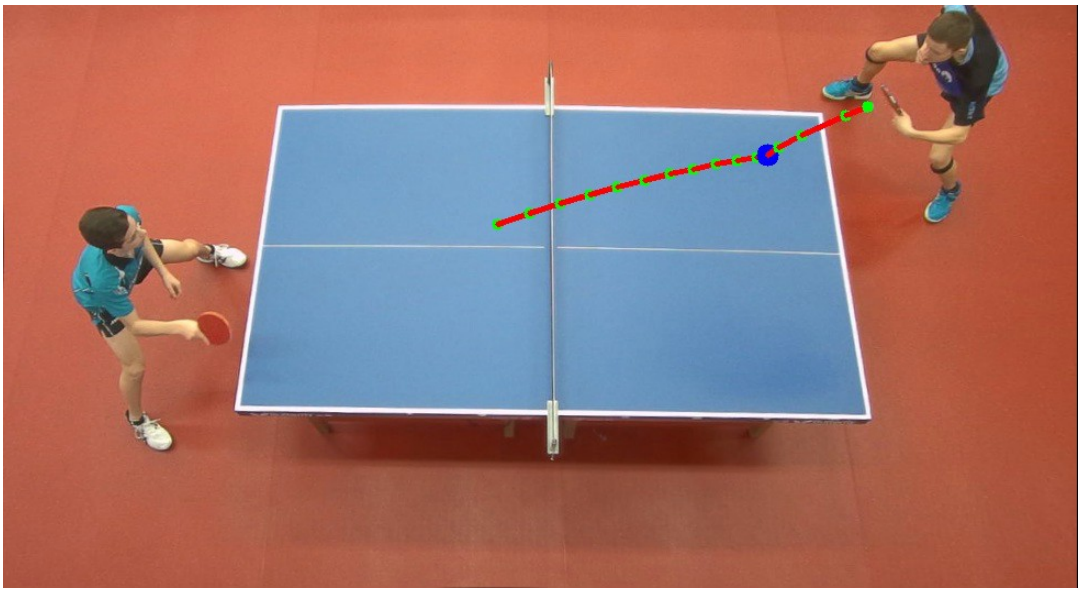
- Naked 0.1.31
- PyYAML 3.12
- appdirs 1.4.3
- mpmath 0.19
- nose 1.3.7
- numpy 1.12.1+mkl
- opencv-python 3.2.0.7
- packaging 16.8
- pandas 0.19.2
- pika 0.10.0
- pip 9.0.1
- progressbar2 3.18.1
- pyparsing 2.2.0
- python-dateutil 2.6.0
- python-utils 2.1.0
- pytz 2017.2
- scipy 0.19.0
- setuptools 18.1
- six 1.10.0
- sympy 1.0
- wheel 0.29.0

A.2 How the software works

The parameters of the script are as follows. The whole program is fitted for one video which is attached to the TFG.

- -v: Path to the input video file. If not present, connected camera is assumed
- -o: Path to the output video file. If not present, no output is produced, instead the processed frame is shown on screen. If present, a progress bar is shown to know how much is left.
- -f: If present, output video frames per second. Default is 20.
- -s: Ball tail size. How many balls should be painted on the frame at a time maximum. Default is 16.
- -l: Whether to loop to video or not. If not present, when the video ends, so does the program.
- -t: If present, the program enters testing mode, ignoring all other parameters. It is a mode for the developer to calculate testing metrics for a particular video. Currently, it is adjusted for a particular video, match3.avi, so the test inputs should be changed inside the code. A progress bar is shown to know how the test is doing.

FIGURE A.1: Output match3_track.avi of the program with match3.avi as input



In figure A.1 one of the frames of the processed video are shown. In here, the red line indicates the ball moves from left to right, if the movement was right to left, then it would be yellow. Each light green dot corresponds to a ball position detected and the big blue dot is a detected bounce. Darker green dots correspond to extrapolated position, meaning they are not really seen, but the algorithm knows where they are.

A.3 Information for developers

The main class of the application is PingPongApp.py, however main.py is the file that reads command line arguments. PingPongApp.py has its mock equivalent PingPongAppMock.py which runs all tests of the video when -t is passed to the script as parameter. The folder classes correspond to the files of the normal execution of the program, while test_classes corresponds to the mock of the ones that need to be tested.

Moreover, in order to run testing with other videos, the BounceCalculatorMock and BallTrackerMock classes need to be modified to change the real frames where the events occurred.