Ping Pong Shot Charts and Training Games

Dillon Koch
Stanford University
dkoch18@stanford.edu

Abstract

Shot charts are commonly used in sports to visualize data from the game. Often times these charts are used in basketball and tennis, but they can also be applied to ping pong to show where the ball bounces during a point. To create the charts, three models from the TTNet [2] framework are replicated to track the ball, detect bounces, and locate the table. The models replicated from TTNet are trained on the same publicly available OpenTTGames dataset and perform similarly to the researchers' models. Next, this work presents a method to use the table's corners to perform camera calibration that allows points from the video to be mapped to the corresponding point in the shot chart. Once these charts are created, they can be used to create training games that encourage players to train specific skills like aiming for the corner of the table.

1. Introduction

Shot charts are a familiar tool for visualizing data in sports, used most often in basketball and tennis. The charts show an overhead view of the court combined with data from the game. In basketball, the data would be X's and O's showing where players from each team missed and made shots on the court. This way, people can look at the chart and immediately see where every shot was taken and its result. Shot charts are also used in tennis to show where the ball bounced on the court during a point. Similarly, data from ping pong games can also be displayed on a shot chart.

Much like the tennis shot chart, the ping pong chart would simply add a small dot to every location on the table where the ball bounces (Figure 1). Anyone interested in ping pong can use these to gain a better understanding of the game. In particular, players can analyze where they hit the ball most frequently. They can use that knowledge to understand their performance better and identify their weaknesses. For example, if a player notices they hit the ball to the right side a large amount of the time, they may decide to practice hitting the ball to the left more often. Players may also use the chart to learn about their opponents' weaknesses, and devise a strategy to take advantage of them.

The objective of this work is to examine a video of a ping pong game to create a shot chart. The approach used in this work involves three main steps. First, it is necessary to track the ball's position in the video throughout each point to know where it bounces. Second, a model must be trained to detect each time the ball bounces to know when it's time to add a new point to the shot chart. Finally, each time a bounce is detected, the ball's position in the video needs to be mapped to the appropriate position in the shot chart.

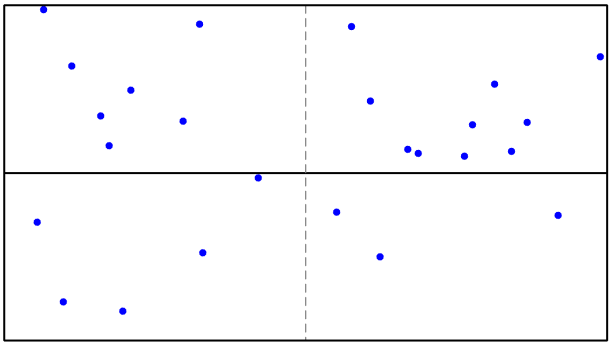

Figure 1. A sample ping pong shot chart visualizing where the ball bounced during a point.

Once the chart is created, it can also be used to create training games for players to improve their skills. Rather than just visualizing bounces, the charts can be used to encourage players to refine their accuracy by awarding points for hitting the ball in certain areas. For example, the shot chart could be used to create a new scoring system where players earn points by hitting the ball close to the corners of the table (Figure 3). The closer the ball lands to the corner, the more points the players score. As a result, the shot chart is providing valuable feedback to players as they train specific areas of their skill set.

2. Related Work

Some of the necessary work to create a ping pong shot chart can be accomplished by replicating the results of the deep learning system TTNet (Table Tennis Net) [2] pub-

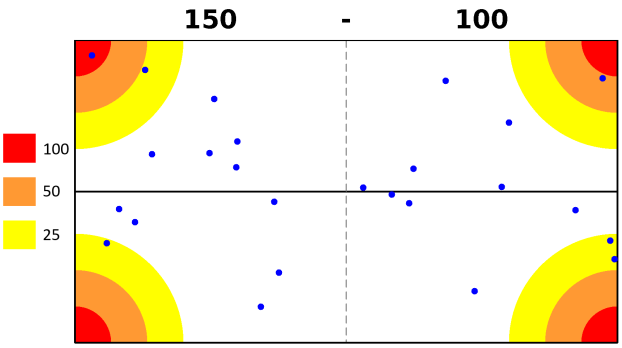

Figure 2. Sample shot chart after playing the coffin-corner challenge.

Figure 3. Empty Ping Pong shot chart diagram

lished by computer vision researchers at OSAI. The motivation for this research was to create an AI table tennis refereeing system. TTNet introduces deep learning model architectures for ball tracking, bounce detection, and table segmentation. These methods are all necessary to know where the ball is, when it bounces, and where the table is. Plus, the authors of this paper also published the OpenTTGames dataset composed of ping pong videos and labels necessary to train all three models.

Another work [1] has performed similar camera calibration involving basketball shot charts. Instead of tracking data from the game and visualizing it on a shot chart, an existing shot chart's contents were mapped onto the video of the game itself. Before mapping the entire shot chart to the video, this work began by locating corresponding points on the court between the court and chart. Once enough correspondences were identified, they were used to calculate the camera calibration matrix. That matrix was capable of mapping all points from the shot chart to the video, not just the correspondences that were easy to identify. Finally, the result of this project was the original video from the game with the shot chart added on the court. This enhanced the viewers' experience by showing the shot data and the game itself in one place.

3. Approach

The first task to create a ping pong shot chart is to make a diagram of a ping pong table that will later be populated with data. Ping Pong tables have a length of 9 feet and width of 5 feet in 3D, and the shot chart diagram (Figure 3) has the same proportions. It also includes the horizontal center line and a vertical dashed grey line representing the net.

Once the blank shot chart is created, the next step is to replicate models from the TTNet framework. First, the ball detection model is used to predict the location of the ball in each frame. Next, the event detection model identifies when the ball bounces or hits the net. The net hits are unnecessary for this project and are ignored. These models are necessary to know when and where a new dot must be added to the shot chart.

In addition to replicating these results, more features must be added to create a shot chart. Knowing where the ball is when it bounces in a video is not enough to add a point to a shot chart. This is because ping pong tables are shaped differently in videos than they are in shot charts. From the traditional camera angle of a ping pong game, the table is located in the middle, with the two players on each side (Figure 5). The table is shaped like a trapezoid in this view, with the nearest side of the table being wider than the far side as a result of their distances from the camera. However, shot charts depict the table from an overhead view, showing the tabletop's true rectangular shape we know it to be in 3D. Therefore, it is also necessary to map the table's trapezoid shape seen in the traditional camera angle to the rectangular shape used in shot charts. This mapping is accomplished by using camera calibration between the game video and shot chart. In each source, the four corners of the table are located and used as correspondences for calibration. As a result, any point on the table in the video can be converted into its corresponding point on the shot chart.

3.1. Ball Tracking

To track the ball during a given frame, the TTNet researchers trained a neural network to predict the ball's location given a stack of nine consecutive frames. The advantage of using nine frames is that the model can track the ball's movement, rather than just looking at one frame where the ball may appear blurry. The model's task is to predict the center of the ball in the last frame of the stack.

The model's architecture is split into two identical halves that perform different functions (Figure 4). The first half of the model predicts the ball's location in a downscaled version of the original frames. The input frames are downscaled from the original 1920x1080 resolution to 320x128. Once the ball's center is predicted in the downscaled stack, the TTNet researchers cropped another 320x128 image around the predicted center from the original full-resolution

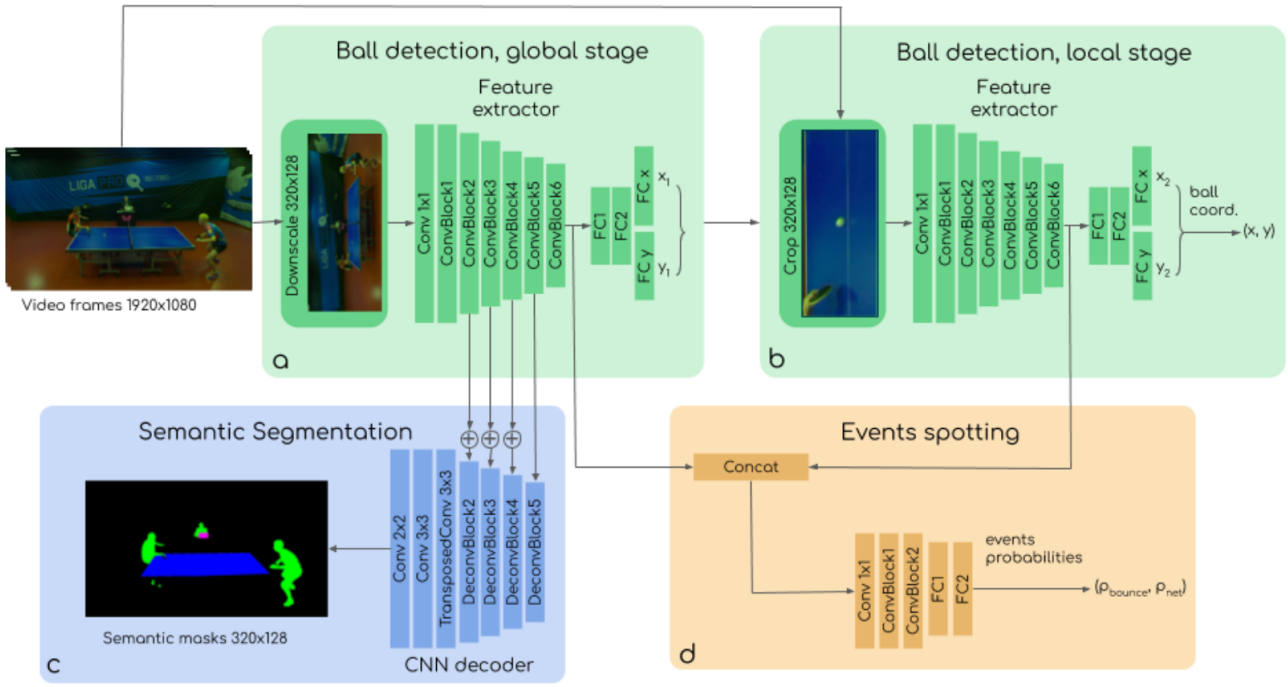

Figure 4. All three model architectures used in the TTNet framework (diagram originally published in [2])

Figure 5. Sample frame from table tennis videos in the OpenTTGames dataset found at lab.osai.ai.

frame. This newly cropped image at full resolution is fed into the second half of the ball detection model, where the ball's location is predicted again. This process of predicting the ball's center on a downscaled image, then predicting the center again on a cropped high resolution image is faster than predicting on the original frame once.

3.2. Event Detection

The event detection model is used for detecting ball bounces and net hits, but this work ignores the net hits. Similar to the ball detection model, it takes stacks of nine frames as input. It has a similar structure as the ball detection model, but with additional convolutional blocks (Figure 4). This model also uses a sigmoid activation function to output two probabilities, one for a net hit and another for a bounce. Due to the higher frequency of ball bounces compared to net hits, the TTNet researchers trained the model with a weighted cross entropy loss. They also added random frames that did not include an event to the training set to help predict the absence of an event.

3.3. Table Segmentation

The semantic segmentation model uses an encoder-decoder approach to identify three classes: humans, the table, and the scoreboard. Similar to the event detection model, this work only makes use of the table's segmentation. Also similar to the other models, the semantic segmentation model accepts downscaled frames from the videos as input. This model is trained on ground truth segmentation masks from the OpenTTGames dataset, and is evaluated using binary cross-entropy. The goal behind segmenting the table from the video is to locate its corners for camera calibration.

3.4. Camera Calibration

The final step in creating shot charts is to map points from the table in the video to the table in the shot chart. This can be accomplished by performing camera calibration using correspondences from the table's corners in the video and shot chart. Identifying the four corners in the shot chart is trivial because this work creates the shot chart. On the other hand, the semantic segmentation of the table from 3.3

can be used to find the four corners in the video.

Once the correspondences are identified, each point *p* in the video and each point *p'* in the shot chart can be represented in Homogeneous coordinates as:

$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad p' = \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \tag{1}$$

These corresponding points are related by the camera calibration matrix *H* as follows:

$$p' = Hp \tag{2}$$

This matrix *H* is a 3x3 matrix with 8 degrees of freedom. Therefore 8 constraints are necessary to solve for its unknown parameters. This means at least four sets of correspondences are needed for calibration, since each correspondence produces two equations.

The process of finding those equations starts with representing *H* as a block matrix:

$$H = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \tag{3}$$

Now the equation defined in (2) can be rewritten as:

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 p \\ h_2 p \\ h_3 p \end{bmatrix} \tag{5}$$

Converting these matrices from Homogeneous coordinates back to Euclidean will provide equations that can be used to solve for *H*:

$$u'_i = \frac{h_1 p_i}{h_3 p_i}, \qquad v'_i = \frac{h_2 p_i}{h_3 p_i} \tag{6}$$

$$h_1 p_i - u'_i h_3 p_i = 0, \qquad h_2 p_i - v'_i h_3 p_i = 0 \tag{7}$$

All four sets of correspondences can be used to create the equations from (7). Those 8 equations can then be rearranged as a matrix-vector product as follows:

$$\begin{bmatrix} p_1^T & 0^T & -u'_1 p_1^T \\ 0^T & p_1^T & -v'_1 p_1^T \\ & \vdots & \\ p_4^T & 0^T & -u'_4 p_4^T \\ 0^T & p_4^T & -v'_4 p_4^T \end{bmatrix} \begin{bmatrix} h_1^T \\ h_2^T \\ h_3^T \end{bmatrix} = Ph = 0 \tag{8}$$

The vector *h* from (8) can be solved using singular value decomposition. After the 9x1 vector *h* is solved, its contents can be reformatted into the 3x3 matrix *H*.

Once *H* is solved for, it can be used to map any of the table's pixels in the video to the corresponding location in the shot chart. This process begins with converting the bounce's pixel location into a homogeneous vector *t* with three components.

$$t = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{9}$$

Multiplying that vector *t* by the newly solved *H* matrix will produce another vector *t'* with three components that represents where a new dot belongs in the shot chart.

$$t' = \begin{bmatrix} u' \\ v' \\ z \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = Ht \tag{10}$$

Finally, the vector *t'* can be converted into a 2D pixel location in the shot chart by converting the vector from Homogeneous coordinates back to Euclidean.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{u'}{z} \\ \frac{v'}{z} \end{bmatrix} \tag{11}$$

4. Experiment

The final shot charts can be created by combining all the models and methods described above. However, it's computationally expensive to constantly run each of the models on every frame. To be more efficient, the only model that is run on every single frame in the input video is the event detection model. When that model detects a bounce, the ball tracking model is executed to locate the ball's position in the video. The ball's position is then used to add a new point to the shot chart. That way, the ball tracking model isn't consuming unnecessary computational power when there's no bounce detected.

Additionally, the semantic segmentation model does not need to be run continuously either. This model does need to be run at the beginning of the video to locate the table and its corners, since those corners are required for camera calibration. However, it is possible that the table's corners may shift a small amount over the course of the video. Perhaps someone bumped into the camera, one of the players nudged the table, or some other event could cause the table's location to move slightly. To accommodate for this, I run the semantic segmentation model periodically throughout the video to reevaluate where the corners are. This strikes a proper balance between consuming too much computations and not detecting the table's movement.

In summary, the experiment is conducted by first running the event detection model on each frame to detect bounces. Each time a bounce is detected, the ball tracking model locates the ball in the frame. After that, camera calibration is performed using the table's corners that are found by periodically running the semantic segmentation model. This process is repeated throughout the video.

4.1. Dataset

The same researchers who created TTNet also published the OpenTTGames dataset for computer vision research. It includes 5 longer videos of table tennis gameplay to use for training models and 7 shorter videos for testing. The videos are recorded with 1920x1080 resolution at 120 frames per second. Each video is captured from a similar angle, with the table in the middle of the frame and the two players to the left and right (Figure 5). All videos come with a json file indicating the frames in which the ball bounces, another json file with the two-dimensional position of the ball in each frame, and also images with the semantic segmentation masks of the table. These three files respectively serve as the labels used to train the bounce detection, ball tracking, and table segmentation models.

4.2. Evaluation

There are two main objectives that this work is evaluated on. First, the three models used in this method must be evaluated. To evaluate the ball detection model, I used the RMSE of the ball's labeled position compared to the predicted position. To evaluate the event detection model, I computed the percentage of correct events predicted. Finally, table segmentation was evaluated with Intersection Over Union (IoU). The TTNet researchers achieved 1-3 pixels RMSE for ball detection in most settings, about 97% accuracy predicting events, and nearly 93% IoU for table segmentation.

Second, the shot charts themselves must also be evaluated. Numeric metrics cannot be computed without possessing ground truth labels, so this objective will be visually evaluated. This can be accomplished by watching ping pong videos and ensuring dots are added in the appropriate place each time the ball bounces. Visually evaluating the shot charts monitors multiple aspects of the project. First, it ensures that bounces are detected at the appropriate times. If dots are added when the ball didn't bounce, or if the ball did bounce and no dot was added, then the event detection model would be at fault. It also helps evaluate the semantic segmentation of the table and the ball detection model. These methods could be at fault if a bounce is correctly detected and a dot was inserted in the wrong place. This could be due to an inaccurate prediction of the ball's location or flawed segmentation of the table. Thankfully these models are also numerically evaluated, because it would be difficult to tell which model was at fault in this situation, if not both. Finally, evaluating the training games will involve manually calculating each player's score based on the shot chart and comparing that to the score shown above the chart.

4.3. Results

The results of the three models are presented in the table below, along with the TTNet researchers' results. The metrics between the two sets of models are very similar. This is not surprising given that the models were trained on the same OpenTTGames dataset described in section 4.1 earlier.

	Me	TTNet
Ball Detection - RMSE	2.7	1-3
Event Spotting Accuracy	94%	97%
Semantic Segmentation IoU	93%	93%

Other qualitative results were measured by visually evaluating the shot charts. To do this, I placed the original ping pong game videos and the shot charts side-by-side in a new video to compare the two. The shot chart is blank at the beginning of this new video, and is populated as the bounces are detected. This was helpful to see how the shot chart was updated in real time, rather than comparing the video to all the data in the final shot chart at once. After watching those videos, it became clear that the event detection model's mistakes were almost always missing a bounce rather than predicting a bounce that didn't actually occur. At times this resulted in fewer dots on the shot chart than there should have been. However, the correctly identified bounces were consistently mapped to the appropriate place on the shot chart. This indicated that the ball tracking model, table segmentation, and camera calibration generally performed well. If bounces were detected and dots were misplaced on the shot chart, any one of those three could be at fault.

Finally, the training games also performed well. Since the input to these games is purely the shot chart data, the games' performance can be evaluated separate from the models' performance. Every time the ball bounced in a shaded region, the score was correctly updated. Many of these results can be viewed from the videos included in the supplementary matrerial section at the end of this work.

5. Conclusion

In this work, methods from the TTNet framework were combined with camera calibration to create a ping pong shot chart that visualizes bounces during a game. The TTNet models were used to track the ball's location, detect when the ball bounced on the table, and locate the table within the video. Camera calibration was also performed using the four corners of the table from the video and the shot chart. This calibration was useful for mapping locations

on the table in the video to the corresponding location in the shot chart. Optional training games were also presented that used the shot charts to help players train specific parts of their skill set.

5.1. Lessons Learned

One of the things I learned from this project was that it can be more effective to run a convolutional neural network twice on smaller inputs rather than once on the full image. This was done in the ball tracking model by passing a downscaled image first, then a cropped section of the full resolution image to locate the ball. This method performed well and will be an interesting strategy to keep in mind.

Another lesson learned is that the camera calibration can perform differently depending on the angle of the camera. All videos in the dataset present the table from a side view, but the camera has a higher altitude in some videos compared to others. In the videos where the camera is up higher, the view of the table top is closer to the overhead view in shot charts, and the camera calibration generally performed better. This makes sense because when more of the table's details are shown, it's easier to get a more exact measurement of where the ball bounces.

Finally, I learned how helpful it can be to use multiple video frames at once as inputs when training the deep learning models. The idea behind this was to track the ball's motion across a small set of frames. All the models in this work used stacks of 9 frames as input, but this could be adjusted for other purposes. For example, the TTNet researchers mentioned how detecting a serve would require larger stacks of frames since serves generally take more time than a simple bounce or net hit.

5.2. Future Ideas

Perhaps the most important piece of this work is finding the table's location in the videos with semantic segmentation. It is important to accurately identify the table to ensure the shot chart is accurate as well. However, the method proposed by the TTNet authors uses downsampled video frames that are less precise than the original 1920x1080 resolution to perform semantic segmentation. This could lead to decreased accuracy in the shot chart. Therefore, it may be worthwhile to construct new segmentation masks on full-resolution frames from the training videos to locate the table with more precision.

This work could also be improved by adding new functionality to detect when each point begins and ends. By segmenting each point from a larger video, individual shot charts can be created for each point rather than one chart per video. The overall goal of the TTNet paper was to create an AI refereeing system that used gameplay video to keep score. Although the paper didn't introduce all the necessary components for creating an AI referee, if those components were built it could help create individual shot charts.

Similarly, the training games would benefit from the AI having more awareness of ping pong rules and events beyond those displayed in this work. For example, players are required to hit the ball on their own side of the table when they serve before the ball goes over the net to the other side. In this work, that first bounce from the serve is identified by the event detection model from TTNet. The bounce is treated the same as any other bounce, which can sometimes cause problems. For instance, while running the coffin corner training game on one of the test videos, a player served the ball into the corner of his own side of the table. The bounce was correctly identified, and the server effectively scored points against himself similar to an own goal in soccer. If the AI were able to recognize serves like it does ball bounces and net hits, logic could be added to ignore the first bounce after a serve in the training games and eliminate this issue.

Additionally, there are more possibilities for training games using this shot chart. I previously mentioned games that awarded points for hitting the ball close to the corners or in the center of the table. Similar games can be created to encourage players to hit the ball near the side of the table, the front or back of the table, or even award points for hitting the ball far away from the last bounce. These new games would help players improve their accuracy in more ways.

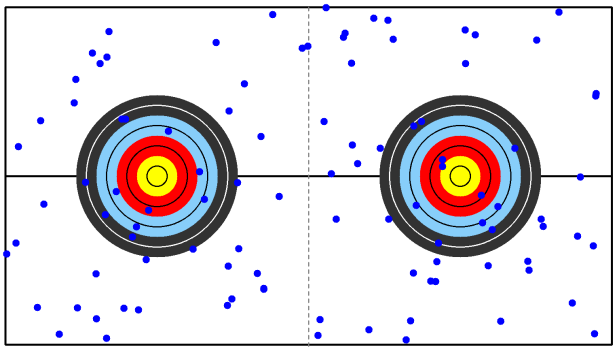

Figure 6. Setup of the Center Challenge training game

Finally, the camera calibration mapping pixel locations to the shot chart can be used in reverse. Instead of adding points to the shot chart based on bounces observed in the video, the transformation can be used to add data from the shot chart back into the video as outlined in [1]. The dots seen on shot charts could be added to the table in the video, as well as the colored regions in the training games. The semicircles in the coffin-corner challenge can be added onto the table to create an all-in-one video with both gameplay and shot chart data.

5.3. Supplementary Material

All of the code for this project can be found at `https://github.com/DillonKoch/Ping-Pong-Shot-Chart`. A video demo of the project can also be found at `https://youtu.be/zhFcnpBYkKQ`.

References

[1] N. Dixit. Adding shot chart data to nba scenes. 2016.

[2] R. Voeikov, N. Falaleev, and R. Baikulov. Ttnet: Real-time temporal and spatial video analysis of table tennis. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.