Extracting Player Tracking Data from Video Using Non-Stationary Cameras and a Combination of Computer Vision Techniques

Neil Johnson

ESPN Analytics
johnson.4205@osu.edu | @neilmjohnson
Basketball and Open Source
1548747

Abstract

This research paper looks at the feasibility of parsing player tracking data from a single non-stationary camera like the one typically used in sports broadcasts. Parsing player tracking data from broadcast video opens up a plethora of applications that allows the technology to be scaled downwards and across sports in addition to capturing events that could otherwise not be captured. This approach uses an array of open source computer vision applications including pose estimation and template matching. Early tests show the accuracy of this new method in placing players within a foot of their true location at 94.5%. Making player tracking data more accessible lowers the barrier of entry and increases the timespan for which advanced methods of analysis can be practiced. Additionally, the pose estimation data itself provides an additional new frontier of data analysis that can increase the fidelity of analysis that relies of player tracking data.

1. Introduction

Player tracking datasets have been a game changer for a number of sports over the past decade. Knowing the exact location of each player on the field as well as the ball is extremely powerful. Unfortunately the fiscal costs to create player tracking data provides a barrier that prevents fans, academics and outsiders from gaining access to these immensely valuable datasets.

Additionally, while the industry leading tracking systems do an exceptional job at providing data for the games their systems are set up to capture, the methodology prohibits the operators from being flexible and agile with what games they want to capture.

The player tracking data methodology presented in this paper aims to augment current tracking system methods by focusing on a new approach that is inherently modular and not dependent on significant financial backing. This approach, called CV Tracking[i], uses open source computer vision libraries on a non-stationary camera video feed like the one used in television broadcasts. This approach allows anyone to create their own player tracking data for any game with adequate source video.

[i] CV stands for Computer Vision

There have been attempts in the past to create player tracking data from video. Back in 2006 Beetz et al [1] created ASPOGAMO[ii], or Automated Sport Game Analysis Model, which was a system used to derive player tracking data from the 2006 FIFA World Cup broadcasts as a part of a competition held during the 2006 RoboCup. Their approach was extremely novel and way ahead of its time, and created player tracking data by estimating the camera parameters and identifying players by detecting blobs that did not match the field color. Football inherently appears to be one of the easier sports to create player tracking data given how large and homogenous the playing surface is and what kind of perspective the broadcast camera must use to capture the game.

2. Approach

The first requirement to create this data is to have video from a game. This paper will use a basketball game in its explanation but this technique can be applied to any sport. Since this method is fairly modular the minimum acceptable video quality will vary depending on the implementation.

In order to create player tracking data from video, we must first break the video up into distinct continuous shots of action. Depending on your source video, the camera angle may never change or may change very frequently. For this process we want to break down the source video into smaller video segments that are distinct continuous shots of action that in aggregate show as much of the contest as the source video contains. Video that does not show game action or does not show enough of the playing surface or players in it can be discarded. For the explanation of this methodology I will focus on applying CV Tracking to one video of continuous action, which can then be applied to every video segment that compromises the entire video available for this given game.

Once a series of videos is created that compromises the duration of the game, we will then apply three computer vision techniques. The output of all three methods will be combined and transformed to create a rough version of player tracking data. Each method can be replaced with a different implementation than the ones described in this paper. From there a series of automated and manual scrubbing techniques will be applied to create the final version of player tracking data.

2.1. Parsing Player Information

The first computer vision technique identifies where each player actively participating in the game is in each video frame[iii]. To do that, we will apply a computer vision technique called pose estimation which identifies specific body parts which compromise the pose of a person.

To do that, we must use a multi-person pose estimator. There are numerous open-source multi-person pose estimators, but after some testing the one that worked best for this application was AlphaPose [2]. AlphaPose is a python-based multi-person pose estimator that can run in real-time, and has a PyTorch implementation that makes it easy to set up[iv].

[ii] The project page for ASPOGAMO is still online: http://www9.cs.tum.edu/projects/aspogamo
[iii] A video frame consists of a single image taken from the video.
[iv] AlphaPose is available to download here: https://github.com/MVIG-SJTU/AlphaPose

Using the pre-trained models that come with AlphaPose, the first step is to execute AlphaPose frame-by-frame on our video segment[v]. Depending on your installation, you can utilize CUDA-enabled graphics cards to speed up the process as well as utilize command-line parameters to further optimize performance. Without the use of a CUDA-enabled graphics card, running AlphaPose can take a significantly long time for a video that is only a few minutes in length and is of typical 720p quality.

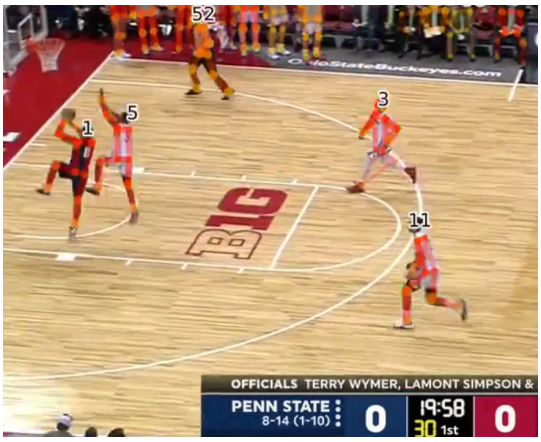


Figure 1 is a cropped still from a processed video frame showing the pose estimation overlays as well as their unique identification numbers.

Once AlphaPose completes, it will output a JSON file which consists of every pose it discovered for every frame within the series of video frames. It can also be configured to output the video frames with the pose superimposed onto the image, as seen in Figure 1, which can help you interpret the JSON output.

Since each video frame is processed independently, we must then run the AlphaPose output through PoseFlow [3]. PoseFlow is AlphaPose's pose tracking software which tracks people frame-by-frame[vi]. PoseFlow can also track people when they leave the video frame or are occluded by other people.

[v] To extract frames from a video, I simply used another free and open-source software called FFmpeg to extract images from the video file at 15 frames per second.
[vi] PoseFlow is available to download here: https://github.com/MVIG-SJTU/AlphaPose/tree/master/PoseFlow

Figures 2, 3, and 4 are cropped video frames outputted from PoseFlow sampled roughly one second apart to show the unique identification number for each player persisting over video frames.

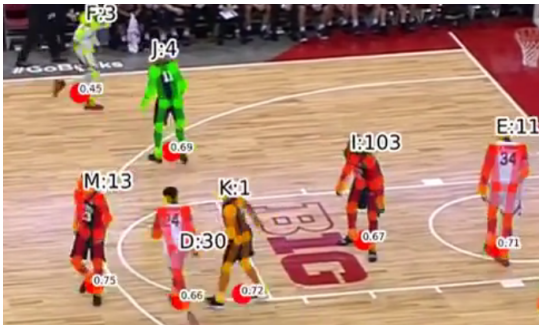
Once PoseFlow completes it will output a new JSON file that attaches unique identification numbers to each pose it finds within the series of video frames. Additionally PoseFlow can also output processed frames with pose overlays as seen in Figures 2, 3, and 4.

Using that, we can manually map those numbers to our own identifiers[vii] for each player, so we know who each player is and what team they belong to. There are some potential methods to automate the identification of each player, but for now this process is done manually. At this point, you will notice some players' poses are not identified in every single frame. The only poses you want to map are ones that are definitively identifying the players of interest. Small gaps in data are to be expected and will not cause an issue.

Once you have identified all of the players within the PoseFlow output JSON file, you can now transform that JSON file into one that excludes all other poses and maps each pose to the player's unique identifier. Additionally we can add center-of-mass calculations for each player's pose. To do this my calculation aims to provide the average coordinate value of each players' ankles, but your definition of center-of-mass may differ. For instance, Second Spectrum's center-of-mass appears to be an average coordinate value of hip location, but their method is proprietary and this is just speculation. Regardless of your definition of center-of-mass, you can calculate it using the body parts each pose estimation provides. The pose estimation data consists of an x-coordinate, a y-coordinate, and a confidence interval.

[vii] For my unique identifiers I just simply used letters.

```
{0,  "Nose"},
{1,  "LEye"},
{2,  "REye"},
{3,  "LEar"},
{4,  "REar"},
{5,  "LShoulder"},
{6,  "RShoulder"},
{7,  "LElbow"},
{8,  "RElbow"},
{9,  "LWrist"},
{10, "RWrist"},
{11, "LHip"},
{12, "RHip"},
{13, "LKnee"},
{14, "Rknee"},
{15, "LAnkle"},
{16, "RAnkle"},
```



Figures 5 shows which body parts the pose estimations from AlphaPose provides. Figure 6 shows the center of mass calculations for players by depicting a red dot and a confidence interval.

2.2. Parsing Court Information

We want to know where specific parts of the court (also known as court features) are within the video so we can then take the coordinates of the court and the people in the video to transform that into the standardized coordinate system typically used for basketball player tracking data.

To do this, CV Tracking will apply a fairly basic computer vision algorithm called template matching. Template matching "is a technique for finding areas of an image that match (are similar) to a template image." OpenCV [4] implemented a template matching function that is very easy to use for this process.

Figure 7 is taken from the OpenCV documentation and visualizes the template matching algorithm.

To create template images we must find unique court features within the playing surface. To maximize effectiveness, what must be done first is prepping the video frames by applying a pre-processing method. An effective pre-processing method helps distinguish the lines on the playing surface from everything else[viii]. This only needs to be done once, and will work across multiple broadcasts with varying video quality. For this example, I created a batch script which called the photo editing software Gimp[ix] which converted the image to black-and-white, where every color close to white was converted to true white and everything else was converted to black, as seen in Figure 8.

[viii] One thing to note is that this process must be done once for each unique playing surface.
[ix] Gimp is a free and open-source photo editor available at https://www.gimp.org/

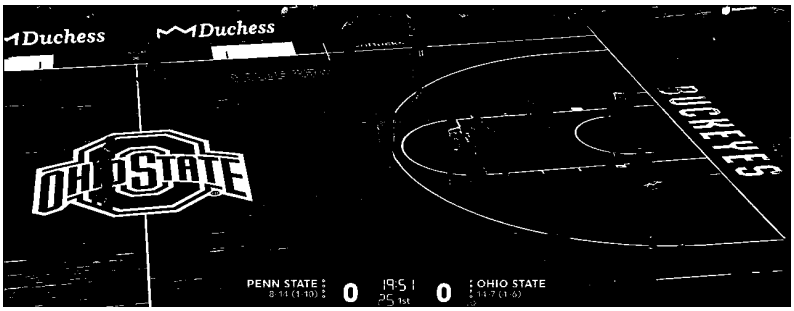


Figure 8 shows the output from the pre-processing batch script.

Once that has been completed and every video frame has been transformed, we must identify template image candidates. Typically the paint area and the center court circle on basketball courts are good candidates. While this paper explains the approach for one template, ideally you would want to identify a series of template images within each video frame to maximize accuracy.

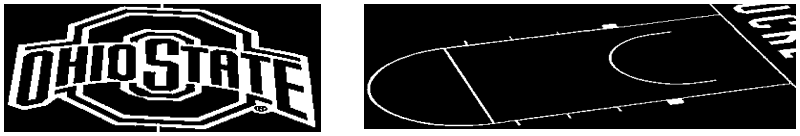


Figure 9 and 10 are two examples of template images.

Once you have created template images, you must apply the template matching function using those templates on each video frame. The result will be the coordinate locations of the template image within each video frame. This function will return its best guess, even if its confidence interval is extremely low because the template image does not exist in the source image.

Using this you can re-visualize the video with the outline of each template image superimposed onto each video frame. This can help you confirm the template matching function is working and help you identify a minimum acceptable confidence interval threshold for each template image. For this paper, I created seven template images, applied them to each video frame, and then created a mapping file which consists of the templates that were correctly found by the script using their confidence value. Then another script eliminates any remaining erroneous template matches, recompiles the video visualization, and allows us to confirm this process has been completed successfully. Figure 11 shows an example of the video visualization, showing two templates being correctly identified within the frame.

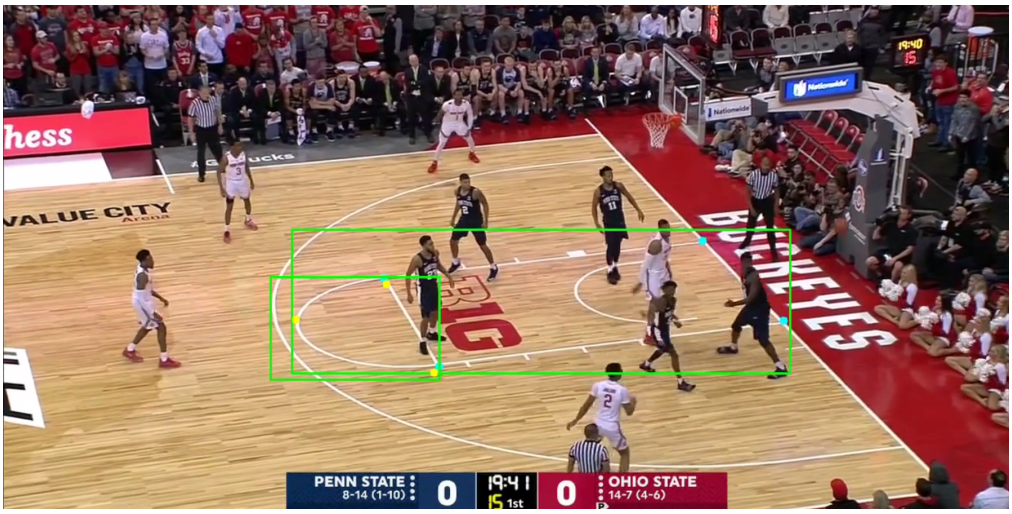


Figure 11 shows two template images being matched within the video frame by overlaying their outlines onto the original image.

2.3. Parsing Game Context

Parsing game context simply requires applying optical character recognition (OCR) to the on-screen scoreboard. If you are using video that is not from a broadcast you might not be able to capture this.

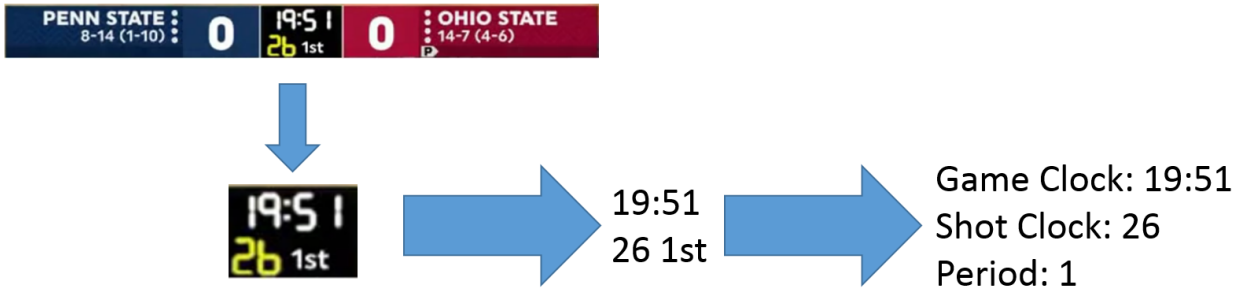


Figure 12 shows step-by-step how OCR can help parse game context from broadcast video.

For each video frame, crop the image so that only the relevant information is shown: game clock, shot clock, and period. Any OCR library should work, but PyTesseract was used for this paper. Accuracy of the OCR function depends on the optimization parameters and the font used on the scoreboard overlay[x]. A simple script can transform the OCR output text into serialized game clock, shot clock, and period information. Given the structure of game clock and shot clock time – both start at a certain time, decrement and always use numbers with a consistent format – scrubbing out erroneous OCR output is fairly trivial.

[x] Some fonts are easier to interpret than others.

2.4. Compiling Player Tracking Data

Once the previous three sections are complete, you have everything to create player tracking data for the video segment: player location data, court location data, and game context data.

Since we are working with two independent planes – the video frame plane and the player tracking data plane – we can simply create a function to convert a set of coordinates from one plane to the other. Since we know the exact locations of the court features, we can use that to create a series of affine transformation functions that we can then apply to the player location data, thus giving us player tracking data that is in our player tracking coordinate system.

Figure 13 shows the coordinate system range for a video frame.

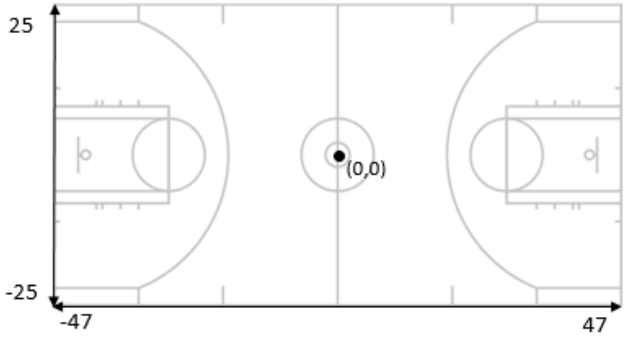


Figure 14 shows the layout of the coordinate system typically used for basketball player tracking data.

The standard player tracking coordinate system is as follows: center court is (0, 0) and each unit in this coordinate system represents one square foot in real life. The standard NBA and college basketball court is 94 feet by 50 feet, so that gives our player tracking coordinate system a range of -47 to 47 on the x-axis, and -25 to 25 on y-axis.

Using that coordinate system, we can map the distinct court features within our input video to what their exact locations are in the output player tracking data. For instance, if you use the outline of the paint on the right side of the court as one of your template images, you can map the four corners of the paint to their exact coordinates within that template image to their values in the output player tracking data coordinate system, which would be the coordinates (8, 28), (8, 47), (8, -47), (8, -28). In Figure 15 those map to A, B, C, and D respectively.

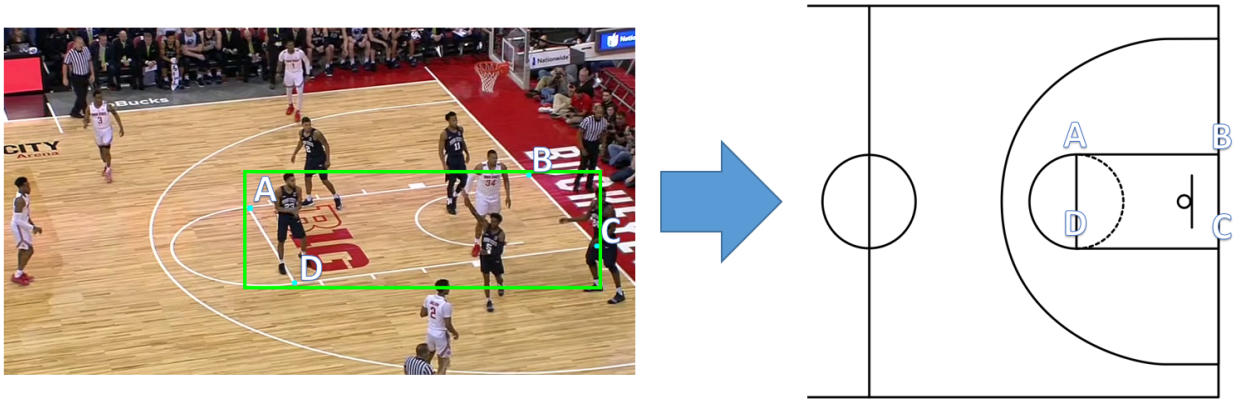


Figure 15 visualizes the affine transformation of coordinate data from the input video frame coordinate system to the output player tracking data coordinate system.

Using coordinate mappings with each court feature creates a series of affine transformation functions for each template image to be applied across all video frame. Each available affine function can then be applied to each player's center-of-mass coordinate. Once that is complete you can output your results into a JSON file, which is the very first draft of player tracking data for this given video segment.

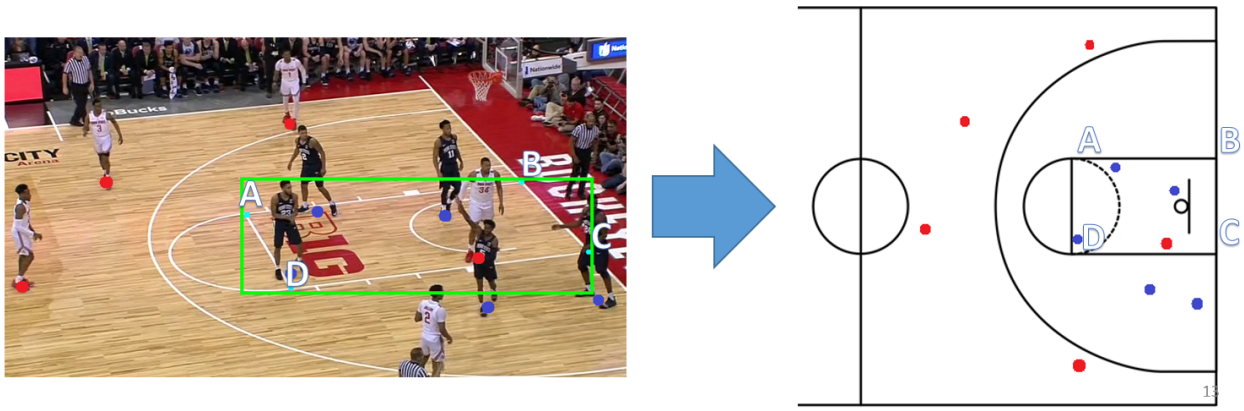


Figure 16 shows the center-of-mass coordinates being translated from the input coordinate system to the output coordinate system.

The first draft of the output player tracking data is going to be fairly noisy and missing chunks of player location data. To improve this we need to apply a series of scrubbing methods.

The first step is to apply a Kalman filter to remove the jitter a player experiences on any given trajectory. This was accomplished by using PyKalman and it is recommended to configure the function to ensure optimal output.

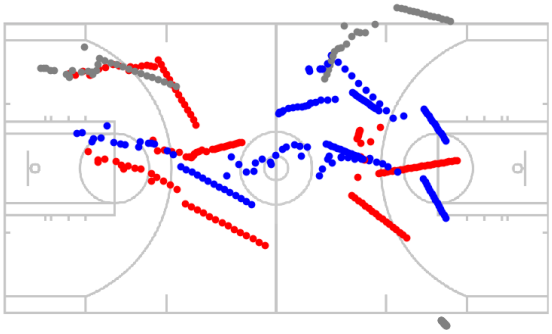


Figure 17 shows the raw player trajectories.

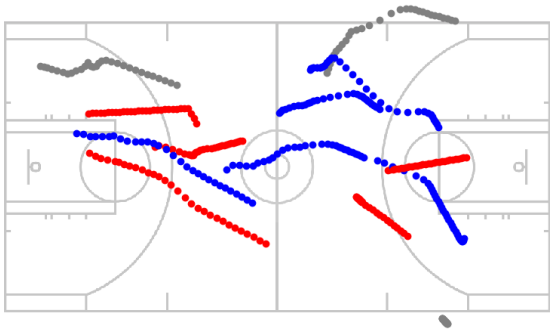


Figure 18 shows what player trajectories look like after applying a kalman filter.

For gaps in a player's trajectory simply applying a linear fit between the two known points around the gap and using a time series function between those two known points produces surprisingly good approximations.

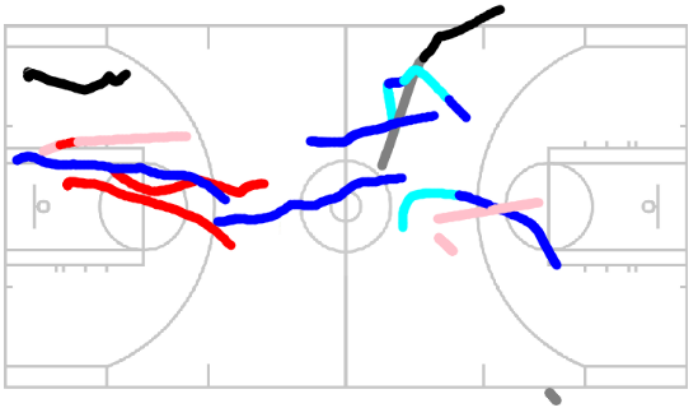


Figure 19 shows player trajectory with observations in red, blue, and black and approximations in pink, cyan, and grey respectively.

Unless your source video captures all ten players on the court for all live game action there will be gaps in the observations for a given player. The typical nature of a basketball game broadcast requires that the camera focus on the most important parts and thus means that the data that is approximated is less critical than the data that is observed.

At this point in the process, some manual intervention will likely occur to help ensure the data quality is up to your desired standards. The process of creating player tracking data for this video segment is complete, and the process can be repeated on the remaining video segments until the entire game is processed. As mentioned before in the player parsing section, there will often be gaps in observations. Fortunately in most instances the gaps are not for long periods of time thus the linear interpolation between the two known points tends to work well and can even be used to automatically suppress anomalies in the observed data that could not physically happen.

3. Results

The results of CV Tracking is a player tracking data file produced only using open source python software. Given the modular nature of this method the data file results can vary in quality, but if you

have adequate source video which comes from a camera that stays on the main action for the entire duration of the game and you have enough processing power to apply the aforementioned computer vision techniques in a timely manner then the resulting data is on par with industry standard player tracking systems and is reasonably feasible to produce.

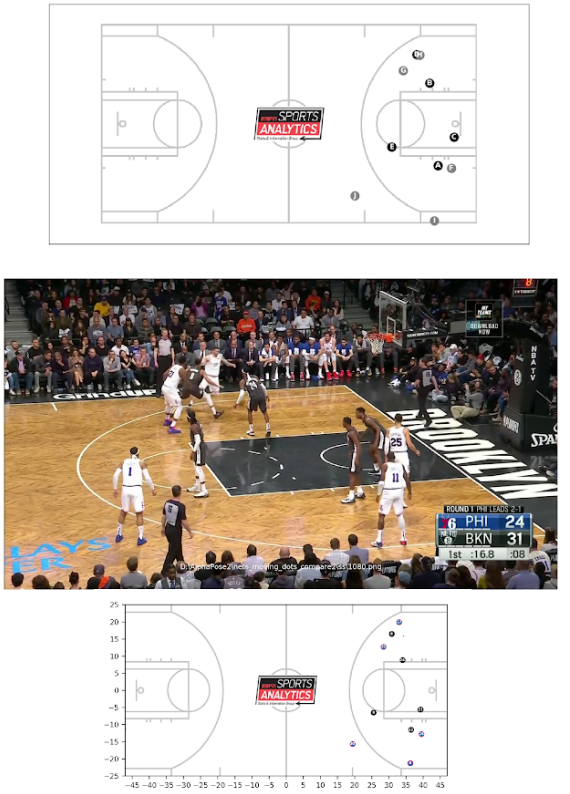


Figure 20 shows one of the images shown in the accuracy test.

While it is difficult to find an ideal way to evaluate the accuracy of this system it is probably best to still attempt to find one. The first test I did was poll a random sample of colleagues and friends. Using still frames that visualized the player tracking data alongside the original frame I simply asked the group: were the dots presenting each player approximately within one square foot of where they truly were given what you can see in the video frame? After nearly 300 samples, the poll results determined that the player tracking data created from CV Tracking was within one square foot of the ground truth 94.5% of the time.

Using the exact same video frames, I mapped them to the Second Spectrum player tracking data and I asked the group the same question. To my surprise, the results were that after nearly 300 samples the player tracking data from Second Spectrum was within one square foot of the ground truth 92.1% of the time.

Now I will mention that CV Tracking is extremely unlikely to be better at creating player tracking data than a system that has at least six stationary cameras constantly recording during the course of a game. To me this speaks to the likelihood that 300 samples is too small of a sample size to produce meaningful results. Ideally the amount of samples evaluated would probably need to be ten times that, if not more, but finding people to voluntarily evaluate 600 samples already took over a month and was the amount I could get before having to complete this research.

For the sake of transparency and due to the lack of ample resources to do thorough testing, I implore all readers to download the raw data created as an example for this paper and to look at the results for themselves. The zip file at the following url contains the player tracking data from a 2 minute video segment, player mapping information, a moving dot animation video, and the actual raw source video itself that the player tracking data was created from:
http://espnanalytics.com/cv-player-tracking

4. Next Steps

While the method to create player tracking data in this paper lays the foundation for an agile, modular, and low cost approach there are numerous fronts on which it could be improved.

The biggest missing piece is obviously tracking the ball. Through the research done to develop CV Tracking it became fairly clear that identifying the ball is a more difficult problem due to a number of a factors. By nature, the ball moves vertically on the court significantly more than players. This means correcting the raw tracking data from a one camera perspective is much more likely to be skewed and will require more rigorous corrective measures. Additionally the ball is much more frequently occluded by players since it is a smaller object. I believe the approach to parsing ball tracking data will have to be significantly different. A potential approach that may be effective is utilizing pre-existing player tracking data that includes the ball combined with CV Tracking data so that an algorithm can be developed to interpret the poses of all 10 players on the court and infer where the ball is given the input pose estimations.

The process of pose estimation will identify everyone in a video frame, including all of the fans in the shot. A simple improvement to this process would be to eliminate poses that could be classified as “sitting” and otherwise eliminate poses that a player would not be in while participating in the game.

Using a linear function between two known points for a given player to approximate their location for the time they are missing from the player tracking data works surprisingly well, but could still be improved. The “Bhostgusters” model introduced by the Siedl et Al [5] does a good job at mimicking realistic player movement given the input from five players on one side of the ball. Using their approach one could create a series of models to approximate the locations of missing players given the amount of known player locations.

The process of mapping pose estimation to player information is the most tedious manual part of CV Tracking. To help minimize this one could develop an array of methods to automatically identify people. Using the pose estimation data you could attempt to find distinct gaits for each person as well as attempt to identify the numbers on their jersey whenever they are directly facing toward or away from the camera’s perspective.

One new area for development is to utilize the pose estimation data itself. This can help increase the fidelity of analysis in numerous areas such as identifying contested shots, identifying defensive stances, and much more.

References

[1] Beetz, M., Gedikli, S., Bandouch, J., Kirchlechner, B., Hoyningen-Huene, N., Perzylo, A. Visually Tracking Football Games Based on TV Broadcasts. *International Joint Conferences on Artificial Intelligence Organization.* 2007.

[2] Fang, H., Xie, S., Tai, Y., Lu C. RMPE: Regional Multi-person Pose Estimation. *International Conference on Computer Vision*. 2017.

[3] Xiu, Y., Li, J., Wang, H., Fang, Y., Lu, C. Pose Flow: Efficient Online Pose. *British Machine Vision Conference*. 2018.

[4] "Template Matching." *OpenCV 2.4.13.7 Documentation*, https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html.

[5] Siedl, T., Cherukumudi, A., Hartnett, A., Carr, P., Lucey, P. Bhostgusters: Realtime Interactive Play Sketching with Synthesized NBA Defenses. *MIT Sloan Sports Analytics Conference*. 2018.

Appendix

Using video demonstrations is the most effective way to show results, but in the absence of that here are some images of real comparisons between the source video and the output player tracking data that was created during the research done for this paper.