Using Mobile Vision for Basketball Shot Tracking and Prediction

Anand Kapadia
aakapadi@andrew.cmu.edu

Gaurav Lahiry
glahiry@andrew.cmu.edu

Abstract

Out project is a basketball shot analyzing iOS app that detects and tracks the ball and hoop, fits a parabola on the shot trajectory in real time, and makes a prediction on the accuracy of the shot. Additionally, the app reports the angle at which the ball enters the hoop. Major challenges faced include accurate ball detection, fast ball tracking with background noise, obtaining real time results and feedback, and dealing with a non-stationary camera. Accuracy of detection and speed of analysis were specifically challenging on an iOS device.

1. Introduction

To perform its task of analyzing a basketball shots arc and accuracy in real time, this project uses various computer vision algorithms. Specifically, the app performs basketball detection using a fHOG (fast histogram of gradients) detector from the dlib library, a HSV color tracking and background noise reduction tacker, and linear least squares with RANSAC to best fit a parabola to the shot arc. The final app also uses the HOG detector to find the basketball hoop and Lucas-Kanade based optical flow to track the hoop after discovery.

It is useful for a real-time basketball tracking app to be implemented on mobile as the target audience of this app includes basketball players who wish to improve their shooting ability at the gym, where any device more powerful than a phone is not available. With the increasing processor speed on mobile, it is reasonable to have a tracking system. However, this project cannot be trivially implemented through opencv as object (basketball) detection is still relatively slow and a simple detector is not fast enough to accurately capture enough data points of a basketball shot arc to parametrize a parabola. Additionally, object detection can be somewhat involved in the training process. Hence, a smarter mix of computer vision algorithms have to be implemented to achieve the task at hand.

Most of the goals set forth in the proposal were accomplished and the app is close to being able to analyze the shot accuracy and angle in real time.

2. Background

Shot angle, while not the most commonly thought of aspect of a basketball shot, is one of the most important for the sole reason that the angle a ball approaches the hoop changes the size or accuracy region of the hoop (size of the margin of error). However, a shot with an angle that is too high is very hard to control due to the additional forces on the ball. Studies have been performed on shot angles, and it has been determined that the perfect shot angle is based on distance to the basket – at the foul shot line it is 51 degrees, and at the three point line it is 45 degrees. Additionally, there are three main components of a basketball shot: release (follow through), backspin, and shot arc. Of these three, shot arc is by far the hardest to judge as a shooter. Hence, this project focused on making an app that measures shot arc and angle in hopes of creating something that is both fun and useful to basketball fans, giving the user key feedback without needing to invest in an expensive smart basketball. [1]

2.1. dlib

The histogram of oriented gradients (HOG) is a feature descriptor for object detection. The technique finds gradient orientation within small subsections of an image. The fundamental thought is that objects can be defined by many small intensity gradient vectors put together to form one larger object or image. This gradient is compiled through a histogram generated for each subsection of the image/template, and the descriptor is all these gradients and histograms put together. Intensity based normalization in subsections can also improve accuracy by providing shadow and light invariance. [2]

This was important for accurate ball and hoop detection as otherwise shadows and changes in lighting, both of which are constantly affecting the scene of the basketball court, affect the ability to maintain the same level of detection. The dlib library was utilized for its HOG detector, which was trained to detect basketballs and basketball hoops. It was also used to track the center positions of the ball as it moved. [3]

2.2. OpenCV

Optical flow is the the motion of objects in an image between two separate frames in a video. It consists of a displacement vector showing the movement of points from first frame to second. [4] The LucasKanade method is based on partial derivatives of the image signal and the affine transformation. It assumes that the "optical flow" of these key features will remain constant within a fixed region, and uses least squares to calculate their movement. [4]

The OpenCV library was used to implement RGB and HSV tracking, and matrix structures were used to work with the parabolic points. Lucas-Kanade optical flow tracking was used to find good features around the hoop and maintain a detection region without suffering massive slow downs due to continuously running the object detector. [5]

2.3. Starter Code

The face landmarking iOS project on GitHub, developed by user zweigraf, provided basic Swift code for capturing and handling the video input. The captured input buffer was then passed to the dlib wrapper function, by way of a bridging header file in the project, in order for the C++/dlib code to modify the image appropriately, using its own 'array2d' datatype. The session handler uses the AVFoundation framework to capture video input from the iPhone camera. The algorithm specific aspects of this starter code were taken out and replaced with code relevant to the basketball shot detection project. [6]

3. Approach

The algorithm used for basketball shot tracking can be split into a few main subsections: ball detection and tracking, parabola tracking, hoop detection and tracking, and shot detection. Most optimizations were performed at the software and algorithmic levels as it was the amount of math that was taking time, no so much the computation efficiency. Resizing was an optimization done at the hardware level. Additionally, due to conflicts between the dlib and armadillo libraries, opencv was used. This proved to be reasonable as opencv was not the portion of the code with the major performance hits.

3.1. Ball Detection and Tracking

3.1.1 Detection

To start ball detection, a fHOG classifier was tested and trained on the computer using the dlib library. First, many images of basketballs were taken both in the gym with a mobile device and off the Internet. These images were then manually labeled with bounding rectangles around each basketball and fed into the fhog trainer. This trainer was mostly borrowed from dlib's face detection example code with a few major parametrization changes.

- A 50x50 sliding window was used (closer to a window size the basketball would appear in)
- Image pyramids of down-sampling ratio 5/2 were used. This causes the training to be more scale invariant (small basketballs and big basketballs are detected) as the trainer runs the data on the training images as well as resized versions of the training images
- Through experimentation, a C term of 9 was used. The classifier is created based on the machine learning concept of support vector machines (supervised learning hence the training data), which is essentially a linear classifier – ball or not ball – but in a high dimension. The C term in this classifier is important as it distinguishes between a hard margin and a soft margin. Increasing the C value will lead to over-fitting of the data while decreasing the C term will lead to allowing more errors (looser fitting due to outliers) of the data.
- The epsilon value, or "the risk gap" as dlib calls it, was kept a 0.01, the minimum value
- The match ratio was changed to 0.48 from 0.5 due to some bounding labels being slightly offset from others due to slight ball obstructions in the training images and the various image sizes.

After satisfied with the results of the trainer on the computer the results file (.svm) was ported to a xcode project derived from zweigraf's work. [6] This project was used as a starter as it provided the interface between the camera and the dlib library.

3.1.2 Tracking

After porting to iOS, it was discovered using the tools that XCode provides (as discussed in the results section) that running the HOG detector as a tracker on an iOS device is extremely slow. This was due to the combination of using a high quality image and having to scan the image through a small window multiple times (due to image pyramids). Changing to a smaller image size resulted in a massive speed increase but ball detection accuracy and consistency was hurt due to heavy pixelation and noise. Hence, an image size of 1920x1080 was chosen as a compromise. This resize was performed at the hardware level by changing AVCapture settings resulting in the fastest resize possible. Yet, the ball detection was too slow even at this compromised image size, and a combination of hoop and ball detection was unusable (1.2-0.7 fps).

Therefore, two compromises were proposed (of which the second was implemented).

- Option 1: Given that the HOG detector works efficiently for small images and given that a basketball

will only move a small amount between frames if the image processing is fast enough, a ball can be detected searching the entire image initially. The consecutive frames can start by looking for the ball in only a cropped subsection of the image close to where the ball appeared in the previous frame. Additionally, knowledge of gravity and physics can be used to search deeper in directions that the ball is more likely to be moving quickly in. If the ball is lost, search the entire image again.

- Option 2: Use the HOG detector to generate a template for a faster tracker. If the ball is lost, generate this template again.

After both were implemented and tested, it was determined option 2 was superior. While there is a performance hit during the calibration period (while the ball and hoop have not been found, frame rates can be as low as 1 ever 1.5 seconds), tracking after initialization is real-time.

Furthermore, three options were tested for ball tracking after a template was generated:

- Option A: Lucas Kanade/Optical Flow. Find descriptors/features within the template found and run an opencv based algorithm (calcOpticalFlowPyrLK) to track optical flow of the key features using Lucas Kanade. Track these features frame by frame until lost.

- Option B: Use an RGB detector to find the average color in the template. Track this color throughout the image. Due to noise as many object appear maroon or orange in a gym, apply a similar cropping technique as was applied in Option 1 of the initial tracking algorithm to increase consistency.

- Option C: Use an HSV detector to find the most commonly occurring HSV pixel value in the template, and track this color throughout the image (as hue is theoretically a more light and brightness invariant color representation than RGB). Again use the cropping technique.

Due to a ball rotating in the air, Option A was eliminated as an option as the Lucas-Kanade algorithm relies on corners, not edges (which is exactly what the circle that the ball appears to be is).

Between RGB and HSV, HSV was selected due to its more robust detection. Yet neither color tracking algorithm was optimal and if there was more time in this project, a correlation filter or other robust tracking algorithm would have been additionally tested. This project could be made significantly better with a more consistent ball tracker.

3.2. Hoop Detection and Tracking

A dlib HOG detector was trained for hoop detection using manually labelled basketball hoop images. Similar parameters were used for training as those used for the ball detector. Interestingly, it was found that the features of the hoop including the net were not picked up as easily by the mobile camera, even though the gradient image output on the computer showed proper training, including the net. This was due to the lack of distinctiveness and sensitivity of light on the net itself, leading to it not being detected as easily on a live video feed. Thus, it was decided to retrain the detector using just the rim (top of the basket), as this provided a sharper feature area and was solid enough to be easily seen by our camera.

As talked about in the basketball detection section, the Lucas-Kanade optical flow methodology of tracking was chosen for the hoop. As the hoop is stationary and has good corner features, this proved to work far more accurately than it did on the ball. Initially the HOG detector was used to find a template but once a sufficient number of good points were found, template based tracking was used allowing for hoop tracking from various angles around the court. Along with an increased portability, this also drastically sped up the frame processing rate and were able to achieve running speeds close to real time.

3.3. Parabola Tracking

3.3.1 Methodology

With the numerous tracking algorithms that were tested, the implementation of a robust parabola parametrization algorithm became vital. The major question was "even with a spotty detector, what is the best way to get a an accurate parabola?" To do this, three steps were taken:

1. Only twenty points of historical ball location data points were stored at any given time. Through experimentation, it was determined that no ball would stay in the air for longer than what twenty data points could not capture.

2. Linear least squares was implemented to best-fit a parabola. This will be talked about in detail below, but Random Sample Consensus (RANSAC) was used to eliminate outlier data points.

3. Given that gravity acts on the ball when it is shot and other reasonable physical limits to what a basketball shot can look like, a parabola parametrization was only accepted if the coefficient of the x^2 term in the parametrization fell within a reasonable range (negative and not too steep or too flat).

3.3.2 Theory

Given a data set of n points, linear least square can be implemented using the fact that $Ax = B$ where in the case of a parabola, A is a nX3 matrix of coefficients to the equation $ax^2 + bx + c = y$, x is the matrix $\begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix}$, and B is an nX1 matrix of the y coordinates of the parabola. This equation can then be solved using

$$x = (A^T * A)^{-1} * A^T * B$$

3.3.3 RANSAC

While using the full set of 20 points to run linear least squares, the parabola found was relatively accurate. However, RANSAC was implemented for cleaner results as RANSAC reduces the impact outliers. The specific implementation used was for every frame, the best fit parabola generated using 6 random data points (the parabola with the most in-liers) out of 500 iterations was used. This resulted in the correct parabola being found quicker, the parabola equation being more stable, and the parabola holding for longer.

3.4. Shot Prediction

Once the parabola of the shot trajectory and the rectangular region around the hoop were calculated, shot analysis was performed. First, the top edge of rectangle (spanning the top of the basket) was exhaustively searched for the presence of any parabola points within a threshold of 25 pixels. With the equation of the generated parabola, intersections between the parabola and the basket were searched for. If detected, the points starting from the point of intersection were colored in green to represent an accurately shot ball. Frequently, the app can declare a shot as accurate before the ball is on the downwards portion of the shot arc but due to lack of time in tuning, the accuracy of this feature is not very high and many of the though-to-be accurate shots hit the rim and bounce out.

Lastly, the angle of entry of the basketball into the hoop was calculated. To do this, the equation of the parabola was differentiated at the key points of intersection found earlier. This gave a tangent line which was used to calculate the slope downwards towards the basket. Using the slope and basic trigonometry, the relevant angle within this triangle was calculated and displayed onto the video feed using OpenCV functions.

4. Results

The video cited here provides the specific points of comparisons for this application's implementation of each feature against other possible implementations. It also shows the application in action in real time, which is the easiest way of understanding and visualizing the workings of the implementation. [7]

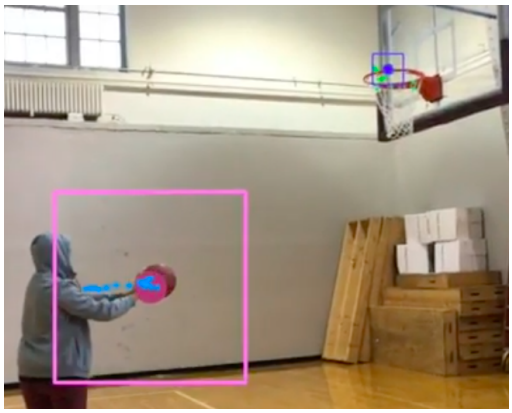

Figure 1. HSV tracking and Lucas-Kanade optical flow in action (ball and hoop respectively)

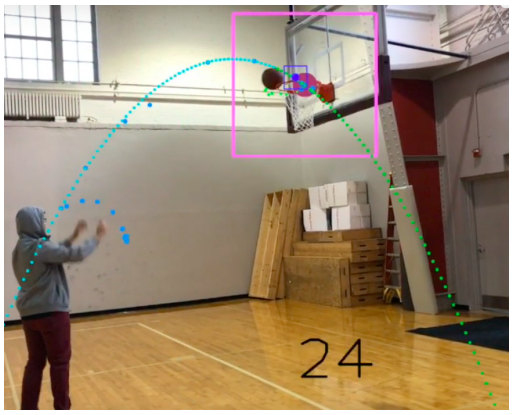

Figure 2. Full fledged tracking with generated parabola, positive shot detection and displayed angle of entry

Figure 1 and figure 2 are example of the finished product, where the HOG detector has detected the ball and HSV ball tracking is used to follow its shot. Simultaneously, the application has found the hoop (also with a HOG detector) and then found good features around the detection window that it could use as tracking descriptors. It is then, in the image, performing Lucas-Kanade optical flow on the hoop to track its position relative to the camera.

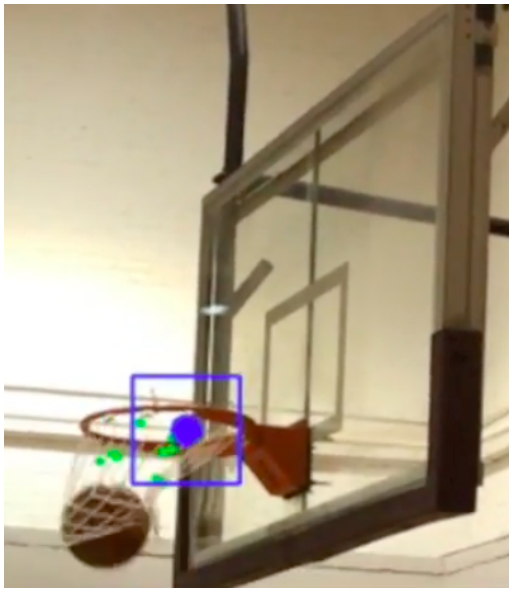

Figure 3. Lucas-Kanade method finds good features around the hoop and forms a box to be used for parabola intersection

Figure 3 shows a closer look at the Lucas-Kanade being performed on the hoop. It also illustrates that a ball being shot into the basket does not disturb the detection and tracking of the hoop. Using this method, it was also possible to view the basket from different angles around the court, without sacrificing accuracy of shot detection and parabola tracking.

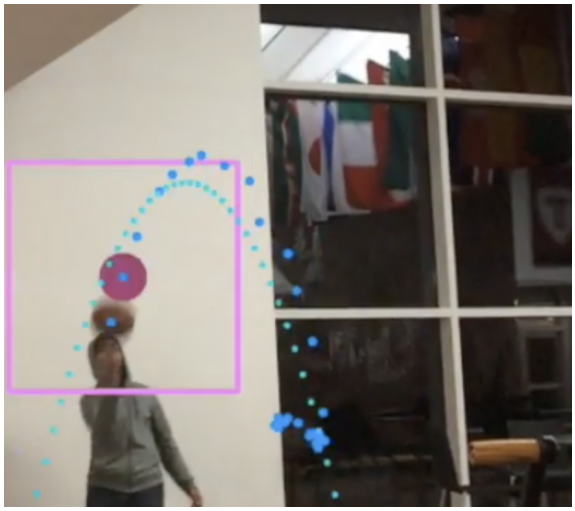

Figure 4. Pure linear least squares parabola tracking

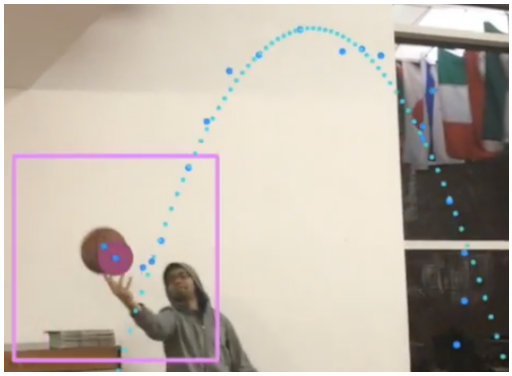

Figure 5. RANSAC optimized parabola tracking

Figure 4 and 5 show the difference between RANSAC (Figure 5) and simple linear least squares (Figure 4). As discussed in the approach segment of this report, RANSAC is able to ignore outliers and hold an accurate parabola while naive linear least squares varies significantly.

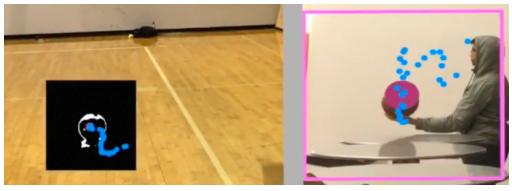

Figure 6. A comparison between RGB tracker (left) and the HSV tracker (right)

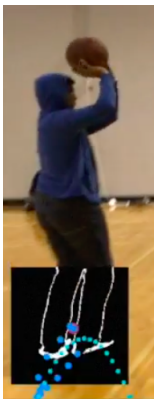

Figure 7. Issues with the RGB tracker: Often getting caught on similar skin tones to the basketball

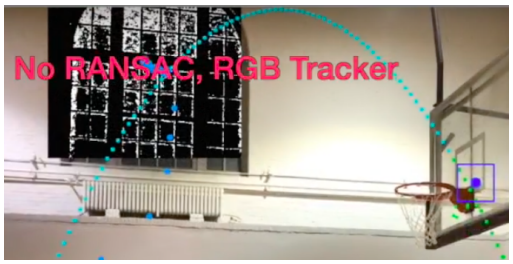

Figure 8. Issues with the RGB tracker: Drawn away by objects or changes in lighting from the background

Figure 6 shows a comparison of the RGB and HSV trackers. As illustrated on the left side of the figure, the RGB tracker only found a small portion of the basketball while the HSV found a larger portion. Additionally, Figure 7 demonstrates the RGB trackers inaccuracy as it transitioned from the ball to the pants/shoes as they were similar colors. Figure 8 continues with this by showing the rgb tracker leaving the ball and settling on the changing lighting from the window.

Figure 9 shows the instruments tab of XCode, which was used to determine where the speed was being lost. As can be seen, over 54 percent of the time was spent in object detection. As such, after implementing a successful ball detector, most of this project was spent trying tracking solutions that did not involved the HOG detector. Additionally, 20 percent of the time was spent converting images to and from dlib. This is one potential reason to move away from dlib in iOS applications in the future. A majority of these assign images were eliminated, but many were still deemed necessary.

5. List of work

Equal work was performed by both project members.

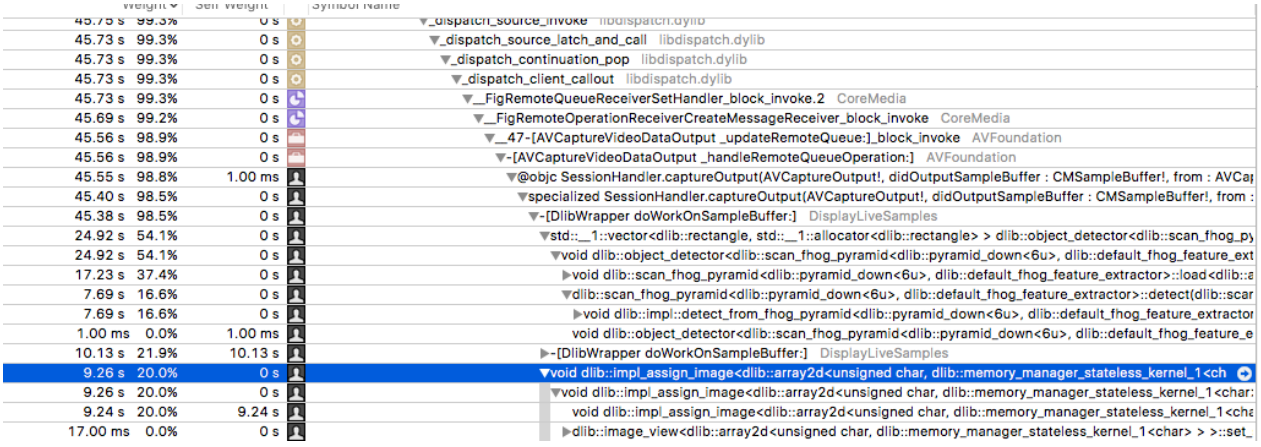

Figure 9. The CPU breakdown of processes running in our application, using XCode Instruments

6. Github

https://github.com/gaurav183/Swish

References

[1] Fisher Sharp Shooters. The physics of shooting, 2017. http://www.secretsofshooting.com/the-physics-of-shooting/.

[2] Wikipedia. Histogram of oriented gradients, 2017. https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients.

[3] Davis E. King. dlib, 2017. http://dlib.net/.

[4] Bruce D. Lucas & Takeo Kanade. An iterative image registration technique with an application to stereo vision. *Proceedings of Imaging Understanding Workshop*, pages 121–130, 1981.

[5] Itseez. Opencv, 2017. http://opencv.org/.

[6] zweigraf. face-landmarking-ios, 2017. https://github.com/zweigraf/face-landmarking-ios.

[7] Anand Kapadia & Gaurav Lahiry. Basketball shot detecting app, 2017. http://youtu.be/ROy2NNFsKAM.