Predicting the Trajectory of a Basketball Shot

Luke Fahmy
CSE 190
A13849289

Abstract

In this paper, I set out to predict the trajectory of a basketball shot using OpenCV in Python. The goal was to have a short video of the first portion of a shot (before the ball reaches its peak), and predict if the ball will go in or not. I used background and foreground segmentation, blob detection, and least-squares estimation to achieve this result in 2D. I found that the environment of the shooter has an enormous effect on the results, and the ability to detect the ball. All in all, this project requires much more work to be practically useful and not just interesting.

1 Introduction

Computer vision is one of the most popular and most useful aspects of Artificial Intelligence. I sought to combine that with one of the most popular sports in America (and my favorite sport): basketball. In particular, I love the detail required to be a good shooter in basketball. Naturally, my passion for the two subjects encouraged me to work on their intersection. Shooting is a very difficult skill for most people to improve in because of the attention to detail required. When seeking to provide some solution to help players in improving their shooting, I've found that having some shot tracking technology is the first step.

2 Related Work

One inspiration for this project is the paper "Predicting Shot Making in Basketball Learnt from Adversarial Multiagent Trajectories" done at Northwestern University. It indicated to me that this is a feasible project for an undergraduate student, even though our ideas and methods were vastly different. Also, the general path of the sports world and the NBA in particular is towards the fields of Computer Vision and Artificial Intelligence in general. SportVu, the company that provided much of the data for the aforementioned paper, also provides similar data to researchers from many different backgrounds. Professional teams then use this research to further the training and development of their players, as well as youth players with the hopes of becoming professionals.

3 Methodology

I used OpenCV 3.4 with Python 3.6 on OS X El Capitan with the occasional use of NumPy. I started with background/foreground segmentation, then moved into shape detection, and then trajectory prediction.

3.1 Background Segmentation

I used the OpenCV bgsegm MOG background subtractor. It required no special parameters. I simply passed it each frame of the video and received a corresponding frame with a guess as to what is not part of the background. Of note is the fact that the algorithm takes previous frames into account. So starting the background detection later in the video will result in having less information to genuinely discern what is background and what is not.

3.2 Blob Detection

I used the OpenCV simple blob detector to detect the ball. This is important because the instinct would be to use some sort of circle detector, but since the ball moves quickly and shadows change on the ball, it is tough to detect the entire face of it. Thus, blob detection will most likely work better, given the proper parameters. OpenCV's blob detector has many parameters to juggle, and they usually require a good amount of tinkering. For my specific implementation, I had the following:

```
params.minThreshold = 0;
params.maxThreshold = 255;
params.filterByCircularity = True
params.minCircularity = 0.7
params.filterByArea = True
params.minArea = 300
params.filterByConvexity = True
params.minConvexity = 0.1
params.filterByInertia = True
params.minInertiaRatio = 0.01
```

The most important among these parameters (the ones that led to the best result) are the minArea and minCircularity. MinArea is in pixels, and it filters out random small spherical shapes that could arise in the background model (since the background model has many irregularities). MinCircularity compares the area of the blob in relation to how it would be if it were a perfect circle. .785 is about the circularity of a square. The minConvexity and minInertiaRatio are more complicated. I put them at near minimum values to catch any shapes that slipped through the cracks because of their oddness, even though their area and circularity make them good candidates to be the ball. This is because the background model sometimes heavily distorted the ball.

3.3 Trajectory Prediction

The trajectory prediction was quite simple. I used the NumPy polynomial fit method, which takes in an array of x-coordinates, an array of y-coordinates, and the degree of the polynomial. Since we are shooting a basketball, and the trajectory of a basketball is like that of any projectile, we knew we would use a parabola (degree of 2). Thus, the polynomial fit method returns a vector with 3 elements: one each for a, b, and c in $ax^2 + bx + c$.

4 Experiment

Now we will go into the workflow of the experiment: what I tried, what worked, and what failed.

4.1 Ball Detection

The only non-trivial task to be done is ball detection. If I can detect the ball, and keep track of its center for long enough, then I will be able to plug in the saved points into a plethora of parabola estimation algorithms to get the trajectory. The next question is how to detect the ball. I eventually settled on the pipeline of MOG background segmentation to simple blob detection (where the blob is re-detected every frame). However, I played with many ideas before landing on this one.

4.2 Tracking

The first idea was to just see if the ball could be tracked well. The short answer is no, not with the methods immediately available in OpenCV without requiring too much computing power. I tried many different tracking algorithms, where I would draw a bounding box around the ball in one frame, and have the algorithm follow the ball through the other frames. In my case, the best tracker was the TLD tracker, and all of the others were close to worthless (for my purposes). In all situations,

however, the space between the ball and the bounds of the box required a decent amount of precision, and even though TLD tracked well with this precision, it still was not usable. This is because the box was too big and the location of the ball within the box varied too much between frames. This makes it difficult to track the center of the ball, which is the entire goal, and thus tracking was not helpful at first.
Later, I tried MOG background segmentation and then tracking. The results were better, but the issue of finding the center of the ball persisted, and thus I concluded that tracking in general was not precise enough for my needs.

4.3 Filtering Based on Color

Even though color is used randomly throughout the project via methods such as blob detection, the idea of detecting the basketball based on its color did not last long. The most obvious issue is that not all basketballs are the same color, and many balls change color over time due to wear and tear. Also, throwing something in the air exposes it to all the different angles of lighting in the room, making it much harder to properly assess its color. So the idea of directly using color to identify the ball was discarded without any attempt at implementation.

4.4 Filtering Based on Shape

Once tracking failed, this was easily the idea with the most potential. It first began with the idea of using a hough transform to find circles. Since basketballs are spheres, their cross-sections are circles, so this seems to make sense from a 2D perspective. However, again, the problem of implementing this in a color version was too difficult and did not seem promising. Thus, I thought to try it after background segmentation. The results of the background segmentation begged for a different method, however, since the ball would frequently change shape in those black and white frames while on its way to the hoop. Sometimes it was strictly circular, but this was rare. Blob detection became the obvious choice since it could detect all sorts of remotely circular shapes and also perfect circles. The final method became blob detection after segmenting the background and foreground.

5 Results

The results were not as good as I hoped, but they show promise, and my mind abounds with ways to improve on them. On surface level, there are many examples where the prediction is three or four inches off, which is not ideal. There are a few where the prediction is spot on. Then there are a few that can just be used to illustrate the weaknesses of the system. These ones have predictions that are nowhere near the hoop because of noise. The noise will either present something that is not the basketball but tricks the algorithm into thinking it is the ball, thus skewing the parabola. Or the noise will keep the blob detector from being able to properly recognize the ball itself. In all cases, I only used single camera videos from YouTube, so the predictions are all in 2D. Thus, I can never say for certain whether or not the ball will go in the hoop.
In general, failures of the algorithm happen because of a few main things. They are camera movement, too much movement by something other than the shooter and the ball, shadows on the ball, and a noisy background behind the ball. I attempted to predict some NBA free throws; they were mostly disastrous. In one case, a player's head was confused for the ball, so the trajectory prediction was quite far off. In another case, there is a background with posters behind the ball, so when the ball flies through that area of the video, the algorithm is unable to properly discern the ball, resulting in less points to plug into the polynomial fit. Remarkably, this case was the most accurate, most likely because there were not too many shadows on the ball. Shadows on the ball were also a slight problem, and most likely the reason for misses that were three or four inches off the mark. Many times, the blob detector would find the ball well, but fixate on the underside of the ball because of its shadow. This causes the assumed center of the ball to be off, which means that ball's point on the parabola is also off.
Another big shortcoming of the algorithm is the question of how many points to use. I have it currently at seven because that seemed to work for all of my examples to an extent. But if I want the program to use as many points as possible before the peak of the shot, I would need to have some way of knowing when the start and middle of the shot occur. Currently, the program only works well with input where the start of the video is close to the start of the shot. So if someone dribbles before

they shoot, the program would fail miserably, because it would count those ball detections as some of the first seven, and incorporate that in the parabola estimation. In all honesty, this program would be much more interesting if it were working on a live feed, but that will not be possible until I come up with a way to distinguish shooting from non-shooting activities.
Lastly, I have not yet made an attempt to locate the hoop. This means that the program cannot actually say whether or not the ball will go in. It can only show the user where the parabola goes, and the user decides if that parabola is going through the hoop or not.
In summary, the program fails because of noise, an inability to use the maximum number of points available, an inability to find the hoop, single-camera (2D prediction), and mistakes in blob detection because of shadows and other round objects. The program has success on videos that start very close to the start of the shot, show the whole trajectory of the ball, and have still cameras and still backgrounds.

6 Future Work

This section will cover a few potential solutions to problems stated above. One way to improve general ball detection is to use tracking. Currently, the algorithm re-detects the ball every frame because the readily available tracking methods were not sufficient. One way to improve this is to use information about the location of the ball. It is obvious that the ball will be relatively close to itself in successive frames. The program can use this to its advantage to never miss the ball when the ball has already been detected once. This way, it will develop an "idea" of the ball as one instance across many frames. This can also help to eliminate other problems such as dribbling. If we always know which object is the ball, we can then define shooting using some characteristics such as its motion to distinguish it from other basketball activities (like dribbling).
Another, more interesting idea is to re-introduce color. The main problem with color was its variance over frames. If we do color segmentation to make the objects in the video all a few solid colors, then it is easier to find the ball based on color, since the subspace of possible colors has expanded. (In the background segmentation method, the subspace of colors is only black and white, so the ball cannot realistically be distinguished by its color). This would also help with tracking the ball, although in this framework, re-detecting the ball would be relatively simple.
The next step in this project is to make it able to count shots in any basketball video, adjusting the predicted trajectory of the ball live until the shot goes in or out. This means that, given any video of basketball activities, the program will seemingly do nothing until someone shoots. During the shot, it will be actively changing the predicted trajectory of the ball until the ball hits the ground again. Then it will keep track of total shots taken and total shots made. This vision will require improved trajectory prediction (i.e. improving the program I have completed for this project), while also adding some way to discern different basketball activities, and some way to find the hoop and decide whether or not the ball went in.