Cricket Pitch Map Generation

Contents

1 Problem Statement and Intro

The task is to develop a computer vision-based algorithm/methodology capable of accurately identifying the landing spot of a cricket ball on the pitch from a sample cricket nets video.

In this report, I've discussed my approach to this problem in detail. I've provided a comprehensive explanation using multiple visualizations, graph simulations and also included code ***generate-pitch-map.ipynb*** to detect the pitch map for a given ball as an additional aid. All of them are attached along with this solution report. Please go through each of them to understand my approach in a better way.

Figure 1: Input (left) (video feed) vs Requirement (right) (pitch map)

2 Assumptions

The assumptions used for this problem are as follows:

- The analysis is based on a single delivery's data.
- The environment is a cricket net setup.
- Basic camera assumptions made are:
  - The camera is positioned at the umpire's position, facing towards the batsman, and the height is adjusted to the umpire's eye level.
  - The camera is stable (we'll discuss the unstable case too)
  - Camera is of high quality (i.e., resolution where we can at least see/distinguish the ball during entire delivery)
  - Record frequency is high (i.e., can record multiple frames during a fast bowl. The higher the better.)
- Default assumptions for Environment:
  - Ball color: Red
  - Ball type: Leather
  - Pitch color: Red soil
  - Batter style: Right-Hand Batsman
  - Bowler style: Right Arm Fast

3 Approach

Our approach begins with gathering data in the most feasible format, which, in our case, entails a video feed of each delivery bowled.According to the assumptions, the feed should exhibit stability, fixed positioning, and clear visibility of the ball.

Next, we proceed to segment out the pitch and the ball. This involves obtaining the coordinates of the pitch edges and its dimensions, as well as tracking the coordinates of the ball in each frame to chart its trajectory as captured by the camera.

Once we have the coordinates of both the pitch and the ball, we can visualize them together to create the pitch map. Each step of this process will be detailed further to provide clarity and insight into our methodology.

Visualisations

All the visualisations that I used in this are mostly images that I provide along with explanation, code based visualisations. **generate-pitch-map.ipynb** attached along with the report.

Along with that, I used a Desmos simulation for the ball pitching and perspective analysis. Each ball can be roughly assumed to be a skewed sine line in 3D space. That's what I used to roughly simulate the environment. To check that live : **Click Here**

Do go through these visualisations and codes. The comparison of actual pitching in the simulation vs processed results using our approach is done in the end.

3.1 Object Segmentation (pitch and ball)

Assuming the feed is stable, we proceed to filter the feed to enhance clarity, particularly focusing on delineating the edges. We may need to pre-process the feed for the edge detection algorithms to work efficiently.

3.1.1 Isolating pitch

Figure 2: Example of edge detection applied to the pitch (link)

Suppose somehow we store the map of the coordinates of pitch to our own image system and get some clean mappings of pitch in **3D space**. As it would be more efficient if we can work with these pitch dimensions and the ball position data (discussed later) in an isolated environment. All this can be done by applying a low weight, threshold balanced CV algorithm, particularly, **Canny Edge detection algorithm**.

OpenCV, has such features to get the contour mappings of the edges found. This was also used in one of my earlier project (link), where I filtered only circular contours to get the number and size of coins to count the money in a faster way.

For this problem, we would need to filter long straight contours to detect pitch. Once we plot those, it would look like below.

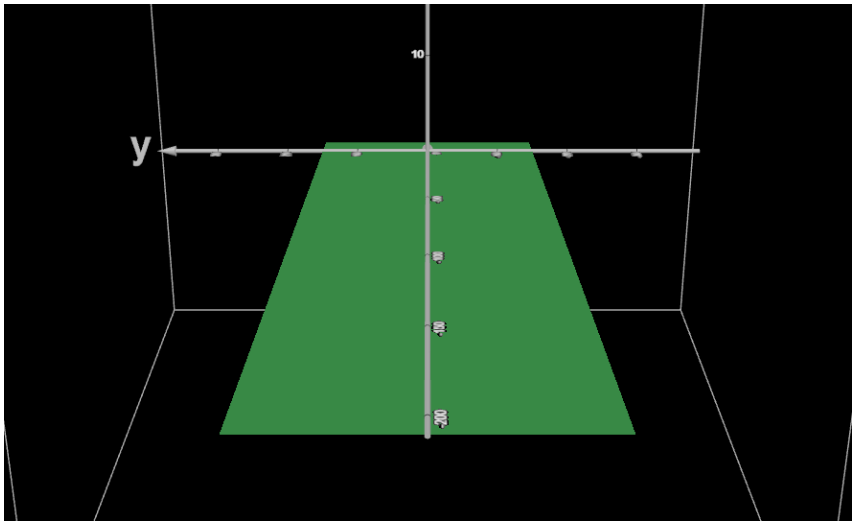

Figure 3: The coordinates of pitch from feed mapped in 3D graph space

3.1.2 Ball Tracking

From the given video feed, we need to extract the position of the ball at all times for us to detect the deflection point of the ball (basically the point where ball hits the ground), and locate it on the pitch map.

This can be done in multiple ways. Major options are:

- **Using Edge Detection**

  Similar to pitch position data but here we search for circular contours. This method is **faster** than other ways in terms of processing. But may not be accurate due to various reasons like **noise**, **poor quality**, **false positives** due to similar objects present in environment.

- **Using CNN**

  As we know, there are various object detection models available for open-source trained on specific data to detect specific objects. One of such examples are **Google's ML Kit**, **LabelBox**, etc. They are computationally a bit slower but far more accurate in terms of detecting the segmented objects.

Suppose for our example I considered using CNN way as its far more accurate. We may train and use our own CNN model for these specific tasks (preferred as its more customizable) or use vanilla object detection models like ML kit by Google, etc to detect the ball and get its position at multiple frames.

Object 0	
Bounds	(1, 97), (332, 97), (332, 332), (1, 332)
Category	FASHION_GOOD
Classification confidence	0.95703125

Figure 4: Example of parameters returned by object detection model

If we look at it's documentation, it takes input as a camera frame and **outputs the position/bounds** of the object (in our case, ball) w.r.t to input camera frame, along with many other parameters such as **confidence, category**, etc

In action it would look something like this,

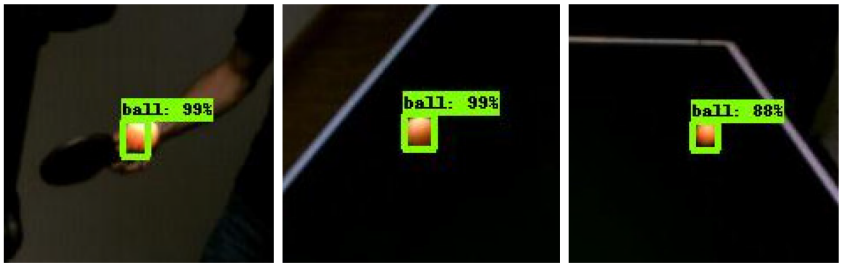

Figure 5: Example of ball detection/tracking

3.2 Data Processing

Considering we now have bounds of the ball detected by our model for each frame in the video, pitch coordinates. Now we can further proceed to the actual detection of the pitch map i.e., position of the ball w.r.t pitch.

This part is explained in ***generate-pitch-map.ipynb***. As we don't have the actual data, I just simulated the trajectory of a ball using a skewed sine wave which closely resembles its motion. [1].

More of such 3D simulations and equations for ball tracking are plotted **here.**

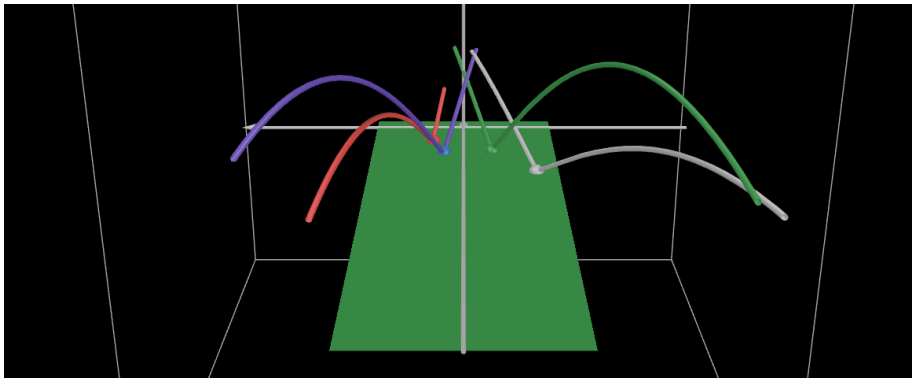

Figure 6: 3D simulation for the ball position data

3.2.1 Plotting combined data

Basically what we're doing is as we have the **data for the bounds** of ball **in each frame** of the video. We take **center** to generalise it as the position of the center of ball. We then plot all these positions in a 2D graph, along with the pitch coordinates.

It looks somewhat like below:

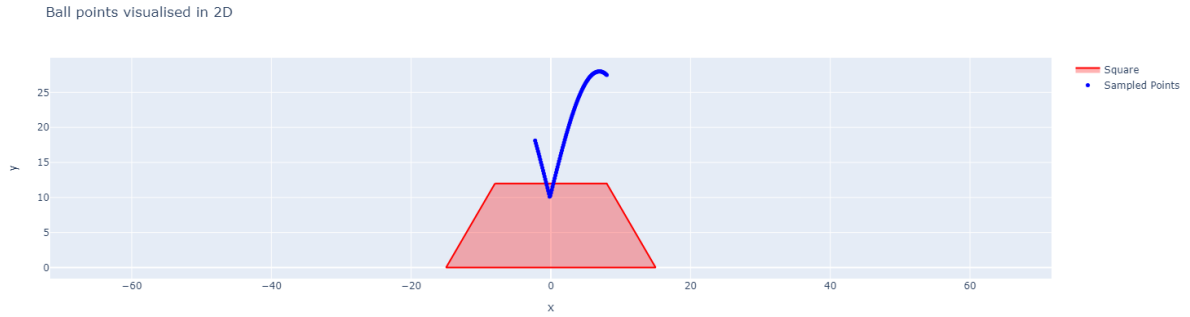

Figure 7: 2D plot of combined ball position and pitch position data

[1] As using a full fledged model , get those points etc was tedious

3.2.2 Finding the Bouncing Point

If we observe, (assuming no noise for now) in the gathered data points of ball bouncing in multiple frames,

- x-coordinates of the data is **monotonous** i.e., constantly increasing or decreasing.
- but y-coordinates of the data is **first decreasing** till the point it hits the ground and **then increasing**.

That's where our bouncing point is. **And this is how our pitch map will look like!!!** (please follow the given code for the calculations and data)

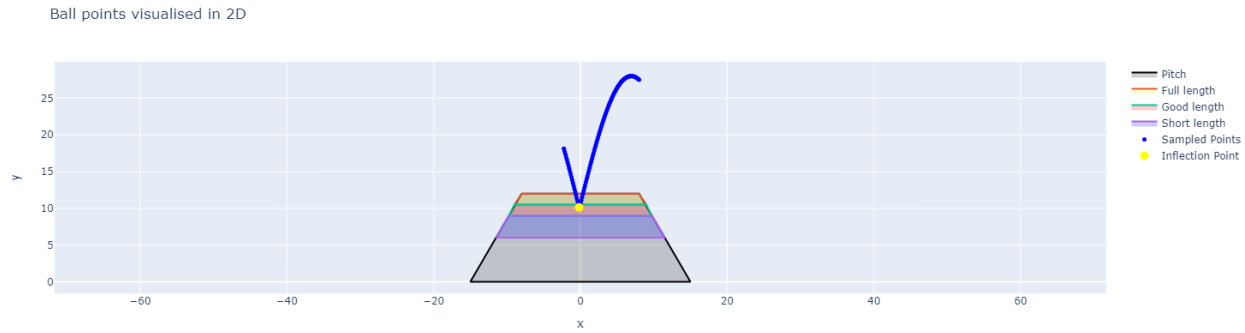

Figure 8: Pitch Map : Bouncing point detected (denoted by yellow point)

3.2.3 Getting actual point using proportions

Now this is what we have, the dimensions of pitch (as seem by camera) plotted, and the bouncing point. We could map different regions (lengths) on pitch like short, good, yorker, etc using simple geometry, since we know the dimensions of all those. And also plot the bouncing point on that. As done above. **This completes our task to get pitch map of a given ball.**

To get the upside view of this pitch map, we take a snapshot of this pitch along with the bouncing point.

And straighten it[2] (as they do for scanning documents) using algorithms like **Image Warping**, **RANSAC**, etc.

We eventually get the below figure as a final result:

[2]warp trapezium pitch into rectangular pitch, considering actual length/breadth ratio too

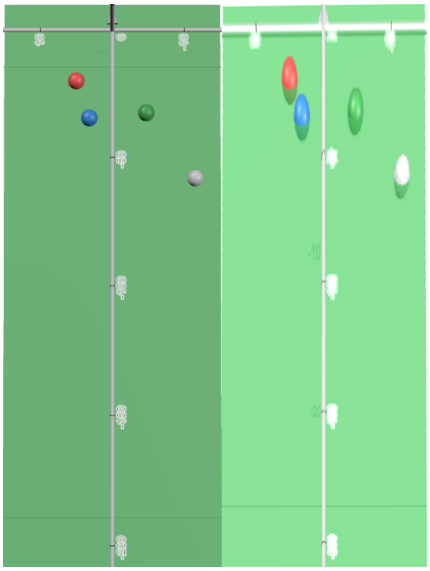

Figure 9: For top view : Actual (left) vs Warped image of pitch (right)

4 Results and Conclusion

4.0.1 Final Assessment

As can be seem in the comparison of results, the pitch map of balls obtained by our method, **closely resembles**[3] to the positions of points in the actual points in the above discussed top view.

This indicates that we've achieved our goal to obtain the pitch map in **both** perspectives (**umpire POV** and **top view**)

4.0.2 Synopsis

So, in summary, we started out with video feed. We segmented the actual pitch data (applicable when both camera is stable or slightly shaky) and ball position data for each frame.

We then find the bouncing point from the ball's position data analysis, plot all the gathered data onto 2D graphs. We warp it the pitch image to rectangle keeping the aspect ratio in mind, if top view is needed.

[3]ignoring the warp shadows/size of the balls, caused due to perspective in simulation, won't be an issue in points in 2D graphs