242 Term Project: The Electronic Umpire

Adina Elisheva Smudski Rubinoff, Gregory Charles Batchelder Wilbur

May 4, 2010

Abstract

This paper describes an algorithm developed to determine whether a video clip of a baseball pitch depicts a swing, a called strike, or a ball. It details the machine learning and vision techniques used to distinguish the different pitches, and describes the results obtained from initial testing.

1 Introduction

Automatic segmentation and annotation of baseball games is a well-explored area. Anyone who has watched a baseball game on TV is familiar with computer-generated annotations on the games, showing information such as the tracks of pitches, whether or not the ball went in the strike zone, etc. However, these computers work with multiple cameras, as well as extra metadata provided by humans about the current state of the game [1]. In this paper we present a program that can identify pitches from far less information. Given an unlabeled video of a baseball pitch from a single angle, our program can identify whether the pitch is a swing, a called strike, or a ball.

The remainder of this paper proceeds as follows. Section 2 briefly describes a library used to aid in video processing and discusses our choice of programming language. Section 3 details the machine learning techniques used in this program, and Section 4 similarly describes the machine vision techniques used. Section 5 mentions some other techniques we considered in the course of the project, but ultimately abandoned. Section 6 describes the testing we did on our project and its results, and Section 7 sums up the project and discusses its uses.

2 Resources

To facilitate the machine vision techniques used in our program, we made use of the OpenCV (Open source Computer Vision) library. This library is made for the manipulation and display of images and video. It has support for C, C++, and Python; we chose to use C++, since the C++ interface has better type safety and makes good use of C++'s operator overloading and exception handling features.

3 Learning

Our program uses machine learning for two purposes: to learn to distinguish pitches where the batter swings from pitches where he doesn't, and to learn to distinguish balls from called strikes.

In both of those cases the k-nearest-neighbor algorithm is used, which works as follows. First the program is provided with a training set of labeled data points. Then, for each data point in the test set, the program identifies the k data points in the training set closest to the test point. The label of the test point is determined by which label is most common among those k training points.

In order to distinguish between swings and non-swings, the program looks at the amount of motion in the video; a swing will have more motion in it than a non-swing. If a pitch is determined to be a non-swing, the program then must decide if it's a ball or a strike. To do this it looks at the location of the ball relative to the plate. (The vision techniques used to find motion and ball location are detailed in Section 4.) So in the first case the data points are doubles representing amount of motion, and in the second case they are Point objects.

4 Vision

The goal of the vision part of this project is two-fold. Given a video of a pitch, the program must first determine whether the batter has swung or not. This is done by looking at the motion in the video. If the batter has not swung, the program must then determine whether the pitch was a ball or a strike. This involves finding the location of the ball when it passes over the plate and seeing if it is in the strike zone. As mentioned in Section 3, once the location of the ball has been found, the program simply uses learning to determine if it's a ball or a strike. However finding the ball's location is all about machine vision.

Given a video, the first thing the program does is split it into individual frames. This is made simple by the OpenCV library (described in more detail in Section 2), which has support for reading in videos one frame at a time. OpenCV also provides methods for finding motion between two frames of a video, and this is used to achieve the program's first goal. The motion of a set of sample points is totaled, and the k-nearest-neighbor algorithm (described in Section 3) is used to determine whether the video is a swing or non-swing.

If the video is a non-swing, the next goal is to find the location of the ball, which is again done by looking at motion. Here the program focuses in on a predefined region around the batter and plate where the ball is expected to go, which we call the motion window. Within this window it focuses on a sample of points; not every point is needed to identify the ball, and using every single point would make the program prohibitively slow. Specifically, it looks at every fifth pixel in every fifth row. These are the dark blue pixels in Figure 1. Using the Lucas-Kanade method for finding optical flow built in to OpenCV, it determines the motion of these pixels between every pair of consecutive frames and divides the moving points into clusters. The centers of these clusters are potential locations for the ball in that frame. This is seen in the center of Figure 1 - the teal dots are moving points, and the teal circle indicates a cluster of them with the center marked with a pink dot.

Once the program has identified these motion points, it connects them together into paths (shown as white lines in Figure 1). This is done on a frame by frame basis. At each frame, it looks at all the paths that have been formed up until now, and adds each motion point to the path whose endpoint is closest to it. Any leftover motion points start new paths. This way the program traces the path of the motion points through time.

The next step is to find the path which is most likely to be the path of a baseball being pitched. This is done by finding functions to approximate all the paths. Each path is approximated by a quadratic function, which is found using regression with the Gauss-Newton algorithm. These functions are then given a score based on a weighted sum of their length, their curvature (flatter

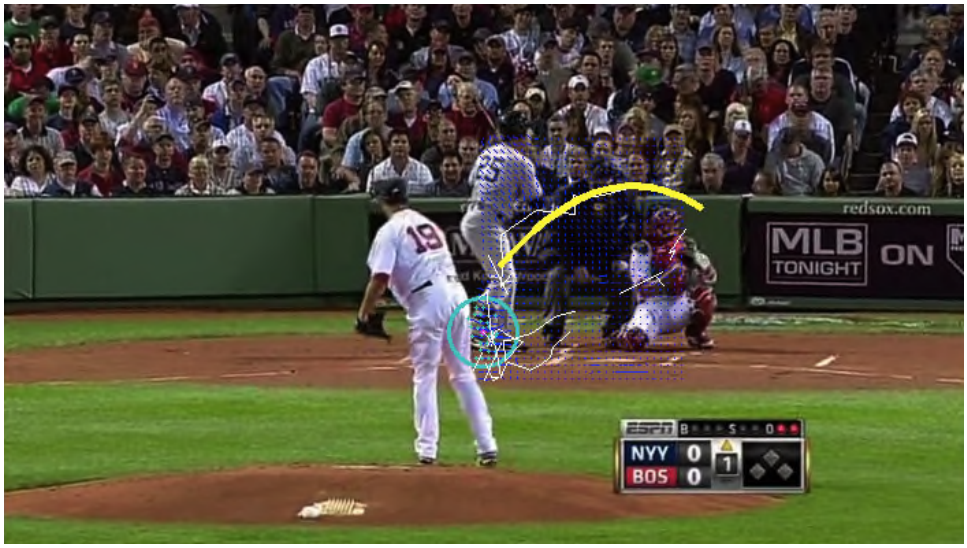

Figure 1: The final frame of a video. Moving points are in teal, clusters marked with a teal circles. Motion paths are drawn in white, and the path of the ball is identified in yellow.

paths are better) and their error (closeness to the original path). The path with the best score is determined to be the path of the ball. In Figure 1 the path identified to be the ball is marked with a yellow line. All of these values can vary between 0 and infinity, except length, which can be at most the length of the diagonal of the video. However, in practice the values were very different: the length was usually between 0 and 500, the curvature was between 0 and 1 (usually smaller than 0.01), and the error was anywhere from 0 to 10^6. From this we chose to set both the curvature and length weights to 1, while the error weight is set to 10^{-6}.

Now that the ball's path has been found, the program can identify where the ball was when it went over the plate. This is found by looking at the next-to-last point on the ball's path. The last point on the ball's path is where it is right before it disappears into the catcher's mitt. Given the speed the ball travels at, the next-to-last point on the path is usually when the ball is over the plate.

In order to return a value for the ball's location, however, the program must normalize the found location to the location of the plate, since it is possible the camera angle will be different for different pitches. To do this, the plate itself must be located. This is done by looking in a predetermined window for the longest horizontal line of white pixels. Since the plate is the biggest white object in its immediate surroundings, this method does a very good job of finding it. To normalize the ball location to the plate location, the leftmost point on the plate is treated as the origin, and the width of the plate is treated as one unit. So if the ball is found at location (x, y) and the plate is d points wide with its leftmost point at (a, b), the normalized ball location is returned as $(\frac{x-a}{d}, \frac{y-b}{d})$.

Figure 2: The root mean square of two consecutive frames.

5 Other Methods Considered

In the course of developing the methods detailed above, we tried many other techniques that were ultimately abandoned for one reason or another. This section briefly discusses these methods and explains the reasons we had for deciding against them.

Initially we tried using a Hough transform to identify circles in the image. We hoped that one of these circles would be the ball, and we could identify it based on its size and location. This proved impossible, however; the video was too messy and the ball was traveling too fast to be picked up by the transform. We also tried looking for clusters of white or light grey points, but this was also foiled by the video's messiness.

From there we turned to look at motion, which proved to be a much better route. One idea we tried was considering only the motion moving in the most common direction; since most of the motion was from the ball, and the ball was moving strongly in one direction, this did a fairly good job of picking the ball out of the video. This ran into problems when the pitcher's foot crossed into the motion window, however, so we abandoned it when we began looking for multiple clusters of motion.

Finally, at one point we considered using the root mean square of the differences between two frames in each color channel (red, green and blue) as an indicator of motion. This did a very good job of identifying the motion in the frame, and shading each pixel relative to how much it moved (as shown in Figure 2) showed very clearly what in the picture was moving . However, it proved hard to find the ball even in this simplified picture, and we eventually abandoned it in favor of the motion-finding techniques provided by the OpenCV library.

Actual Pitch	Program's Guess		
	strike	ball	swing
strike	1	5	0
ball	5	13	0
swing	0	6	14

Table 1: Breakdown of the results into types of pitches and the corresponding program responses.

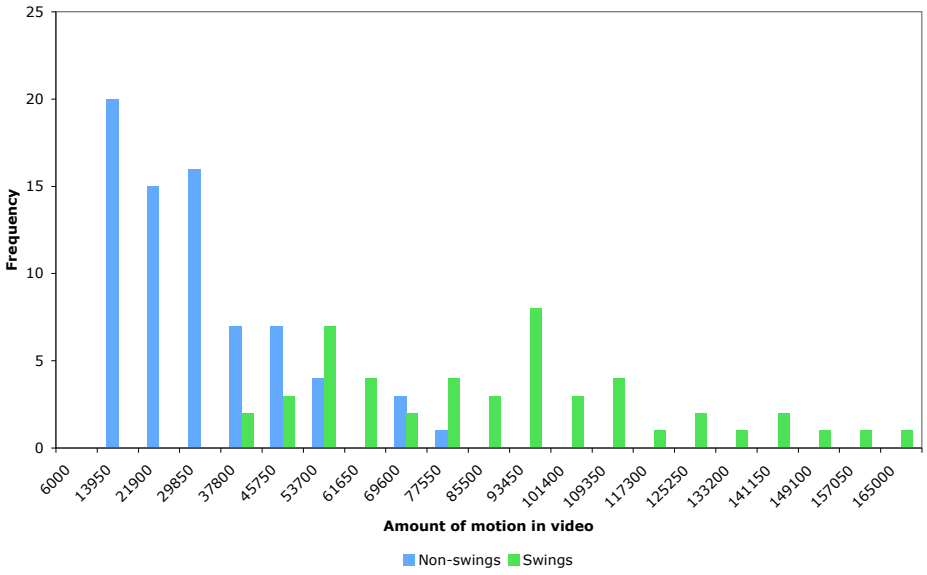

Figure 3: Training data for non-swings vs. swings.

6 Results

We trained our program on a training set of 122 videos and tested it on a test set of 44 videos. This gave us a success rate of 63.6%. A breakdown of the results is shown in Table 1.

The program is very good at distinguishing swings from non-swings; in fact, it never misidentified a non-swing as a swing. It runs into a bit more trouble distinguishing strikes from balls; it correctly identified a majority of the balls, but did think some were strikes. It also misidentified almost all the strikes as balls. The reason for this is pretty clear if we look at graphs of the training data. As Figure 3 shows, the data for swings vs. non-swings is pretty well differentiated. Most of the non-swings are on the left side of the graph, corresponding to less motion, while most of the swings are on the right side where there's more motion.

The data for balls vs. strikes shown in Figure 4, however, is a different story. The data points for balls and strikes are completely interspersed, meaning that the program would have no real way to determine which class an unidentified pitch should be in. The only reason the program was able to correctly identify so many balls is because the training data contains many more balls than

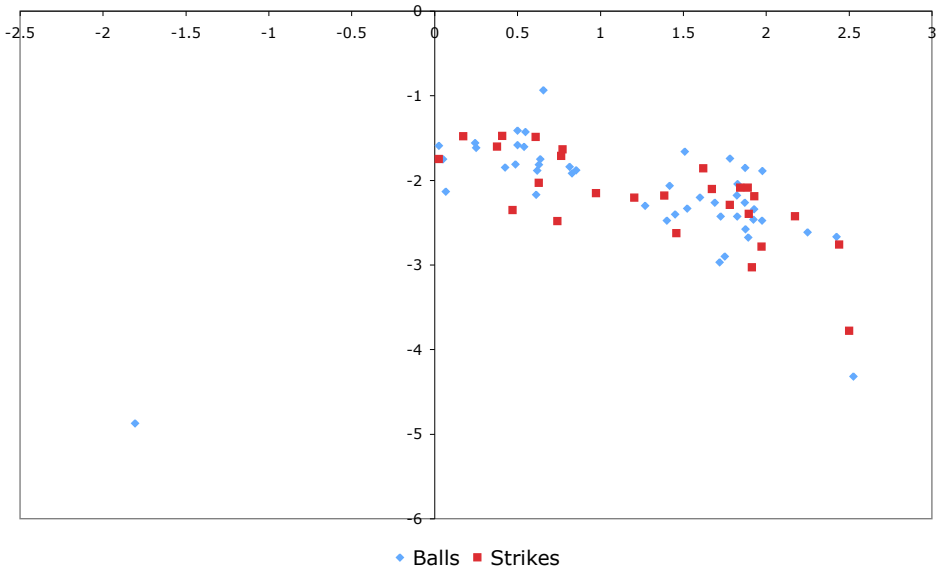

Figure 4: Training data for balls vs. strikes.

strikes. This indicates that our methods for determining whether a pitch is a ball or a strike are flawed. Most likely the program is having trouble identifying the location of the ball. In the future, it would do to examine these methods and improve them.

7 Conclusion

In this paper we present a system for identifying whether a given video of a baseball pitch shows a swing, a called strike or a ball. This is done using both vision and machine learning techniques. Vision is used to identify the location of the plate and ball and to determine the amount of motion in the video. Learning is used to determine how much motion indicates a swing, and whether the location of the ball relative to the plate indicates a ball or a strike. Ultimately, the program achieves a success rate of 63.6% in our tests. We think this is a satisfactory rate given the difficulty of machine vision and this problem in particular. We also think this project can be of much use in annotating baseball footage, since it requires much fewer resources than standard computer-annotation programs - only one camera angle is needed, and no human metadata needs to be provided.

References

[1] Akafuji Tomohisa et al. Annotation method for video content based on application of semantic graph. *Journal of the Institute of Image Information and Television Engineers*, 59(11):1677–1686, 2005.