Semi-Automated Collection of Pitch Location and Intent in Baseball

Andrew D. Klein

April 29, 2011

Abstract

An analysis of the pitcher's intent in baseball should ideally depend upon information given by the catcher before the pitch is thrown. Expert judgment has already identified the importance of this factor (e.g. the pitcher is "missing his spots" when unsuccessful in hitting the target). We describe the underpinnings of a automated video analysis system that uses semi-supervised learning methods to identify the catcher position – specifically, the catcher's glove position. The analysis begins with video of a single pitch, supervised by a human controller; this information is then incorporated into one of a selection of learning algorithms and applied to subsequent pitches with minimal involvement from the controller. This is designed to be the first step in creating a public database of pitch intent, to be coupled with existing sources of pitch physics, so that analysts may better evaluate pitcher performance.

Contents

1 Introduction and History **7**
1.1 Catcher Spotting . . . 7
1.2 Object Recognition In Sports Entertainment . . . 9
1.2.1 FoxTrax . . . 10
1.2.2 ESPN *K-zone* . . . 10
1.2.3 Pitch-F/X . . . 11
1.3 The Introduction of Sabermetrics . . . 11
1.4 History Of Image Processing . . . 13
1.4.1 Image Processing Currently Applied To Sports . . . 13
1.4.2 General Image Processing: The Canny Edge Detector . . . 14

2 Images **17**
2.1 Quantifying the image . . . 18
2.2 Canny Edge Detection and Gradients . . . 21
2.3 Improving Canny . . . 22
2.4 Computational Problems . . . 24

3 Clustering Methods **25**
3.1 K-Means Clustering . . . 25
3.2 Other Clustering Methods . . . 26

4 Search Methods, Given Processed Images **29**
4.1 Inputs . . . 30
4.2 Up/Down, Left/Right Search . . . 31
4.3 Spiral Search . . . 32
4.4 Spray Search . . . 32
4.5 Task: Given Glove, find another glove . . . 34
4.6 Tuning Parameters . . . 34
4.7 Tracking an object over time . . . 35
4.7.1 Modified Spray Search . . . 35
4.7.2 "Stamp" Method . . . 36

5 Developments and conclusions **39**

Chapter 1

Introduction and History

1.1 Catcher Spotting

Due to an tremendous increase in the richness and quantity of available data as well as assessment methods of a player's contribution, we are living in a "Golden Age" of baseball statistics. It is now possible to compare players that played 40 years apart, or to compare the relative value of a pitcher versus the value of a position player. For example, one can see how much more (or less) Babe Ruth benefited the New York Yankees in the 1920's compared to Nolan Ryan's impact on the Houston Astros in the 1980's.

There is one area, however, that has not seen a comparable increase in data availability or assessment. Pitcher "accuracy" is one of the oldest qualitative measures, but remains unsatisfactorily quantified. Since the first days of the broadcasting of baseball games, announcers and fans alike have let their opinions be known about how "accurate" a pitcher is, under the assumption that the target his catcher sets is a faithful representation of a pitcher's intended target. A pitch is considered accurate if the catcher does not have to move his glove "much" to catch the ball. Objectively determining how "much" movement is too much contributes to the difficulty in determining pitch accuracy.

There are many practical benefits of having such a measure for pitchers in the management of a baseball team, both in the short term and long term. In the short term, a manager might look at their starting pitcher's within-game accuracy and see that it is consistently low in the early innings, and decide to make a replacement before he gets into trouble. Similarly, late in the game, a manager could use this measure to determine if it is time to take out the pitcher (if accuracy is a strong function of pitch count for example). A famous reminder of these decisions was Game 7 of the 2003 American League Championship Series, in which Red Sox manager Grady Little was forced to decide whether to leave starting pitcher Pedro Martinez in or replace him with a relief pitcher.

Having a consistent measure of pitcher accuracy would also have long-term benefits to a team. Talent scouts would have a potentially very useful measure to help determine the value of a prospect, and would be able to compare accuracies across players, past and present, in an attempt to better predict their career as a Major League Baseball player. For example, a scout could look at a prospect in 2011 and see that they have a similar accuracy to Sandy Koufax, who pitched in in the 1950's. Unlike home runs, accuracy may not be as volatile a measure; factors such as rule changes or the use of steroids would not have as strong an effect on pitching accuracy. It may also prove to be a powerful prediction of an injury; team trainers could look back at previous injuries and see that if a pitcher's accuracy suddenly decreases, or if they are consistently missing "low and away", that the pitcher is about to tear a key ligament or pull a particular muscle, leading to significant injury time.

For this analysis, I measure accuracy as defined by Andrew Thomas (2011). He defines the term "accuracy" in terms of intended target and the actual impact point of the thrown pitch (or where the ball is hit by the bat); where an "accurate" pitch is one where the distance between these two points is minimal. The distance is the Euclidean distance on the pixels between the target and the impact. This definition uses data that does not appear in previous measures, such as or any information collected by organizations such as Baseball

Prospectus, or anything released by Major League Baseball. Thomas attempts to crowd-source the solution; he has developed an applet that allows users to watch pitch sequences and manually click where the catcher set up his mitt and where it ended up. Having many users rate the same pitch allows for a higher inter-rater reliability than an automated process. Although this method would likely produce results that are closer to the truth, it would need to achieve a critical mass of people using the applet in order to tackle the massive dataset that is all pitches that have been recorded on video all time. I propose a method of tracking the movement of the catcher's glove (semi) automatically. To do so, I use a combination of Machine Learning, Statistics, and Computer Science methodologies.

I continue this chapter with some history on image processing in sports, followed by the use of statistical methods in sports. In Chapter 2, I address how the data are collected, and discuss the different approaches at representing and transforming the data pre-analysis. In Chapter 3, I describe clustering methods as an attempt to extract labels from the data and how they can be used to enhance the algorithm. In Chapter 4, I describe the algorithm created to search through an actual image to find chosen objects, such as the plate and the glove. Finally, I summarize my findings in Chapter 5, as well as discuss future work.

1.2 Object Recognition In Sports Entertainment

Here we detail several technologies that have been used commercially in the acquisition of positional data. The methods outlined below all fall under the umbrella of object recognition. The goal of object recognition is to find an "object" in a video and track its position over time.

1.2.1 FoxTrax

In 1996, the broadcast network Fox introduced a technology called *FoxTrax* into their coverage of National Hockey League games. *FoxTrax* used a special puck that had a microchip inside, which allowed for the puck to be more easily tracked with respect to its position on camera, allowing Fox to add a glow to help it stand out to the viewers. Although the stated goal of tracking an object is similar, the method Fox used is not feasible in my framework; unlike Fox, I do not have the power to mandate that all catcher's gloves include a microchip for digital tracking purposes.

1.2.2 ESPN ***K-zone***

In 2001, ESPN introduced *K-Zone*, a technology that tracks the ball from the release of the pitch to the catcher's glove (or the batter's bat). Unlike *FoxTrax*, the object being tracked, in this case the ball, did not have any embedded tracking device; the tracking is all accomplished through object recognition. Two features of the ball that make it comparatively easy to recognize by automated systems are its color and shape. Because of the frequency at which a new ball is introduced into the game, it is almost always perfectly white. Additionally, because it is a perfect sphere, when viewed in a video or image, it always appears a circle with a (almost) constant radius.

Notably, these properties are not as minimally variable with catcher's gloves, which are often of different sizes and colors for different catchers; although, these gloves may be constant within a game for a particular catcher, they are not as robust to rotation as the spherical baseball.

1.2.3 Pitch-F/X

In 2006, the company Sportvision started installing specialized cameras in baseball stadiums specifically to capture the ball as it is pitched. Similar to *K-zone*, the cameras are able to detect the ball as it leaves the pitcher's hand and track its movement until it reaches the plate. From this video, the movement, as well as rotation, of the ball can be calculated. Combining these two measures allows the type of pitch to be more easily calculated. Most stadiums have incorporated this technology such that the scoreboard shows not only the speed of every pitch, but also the type of pitch and amount of movement (e.g. "Vertical break: 3 inches"). One still might question the accuracy of these measurements as they are displayed almost immediately after the pitch is thrown.

While this approach very similar to what I hope to achieve with the glove, there are a few key differences. First and foremost, as mentioned above, having a computer "recognize" a white ball with ultra high-definition cameras is much easier than having a computer look for a glove that can be any color and change shapes between frames. Secondly, most fans who wish to pursue this type of analysis (such as myself) do not have the benefit of running their video processing algorithms on devices so powerful as to achieve near-instantaneous results.

1.3 The Introduction of Sabermetrics

Because the ultimate goal of this research is to create a new viable measure of accuracy for a pitcher, I also briefly discuss other recent advances in baseball measures. Those who analyze baseball using advanced statistical measures often refer to their work under the title of "sabermetrics", a name coined in tribute to the Society of American Baseball Research. Bill James, its most famous practitioner, describes it as the "search for objective knowledge about baseball" and "the mathematical and statistical analysis of baseball records." The latter definition shows how although useful, sabermetrics relies entirely on existing records. The advanced measures created by “sabermetricians” are derived from basic measures that have been recorded over longer periods of time. For example, “On Base Plus Slugging (OPS)” is a sabermetric that has gained popularity in the mainstream media. OPS is calculated as On Base Percentage (OPB) + Slugging (SLG), where:

$$OPB = \frac{Hits + Walks + TimesHitbyPitch}{AtBats + Walks + SacrificeFlies + TimesHitbyPitch} \tag{1.1}$$

$$SLG = \frac{TotalBases}{AtBats} \tag{1.2}$$

This measure attempts to combine “traditional” measures (such as hits and walks) to evaluate the combined ability of a hitter to hit for power and average.

Because there is no agreed-upon definition of “pitch accuracy” as an objective term, there is no single way to define it. A definition that uses the notion of distance between target and impact, while an intuitive idea, also requires a considerable amount of new data collection.

If one wanted to create a measure of accuracy using current data available, there are several viable options. For example, the number of called strikes a pitcher throws, compared to the respective number of balls, can be used to create some measure. Using the Pitch-F/X data, one could find the number of “corner” strikes (pitches that are called strikes and are located towards the edge of the strike zone. However, neither of these measures take into account any intent other than assuming the pitcher wants to throw a strike that the batter will either miss or not swing at. Using Thomas’ method, intent plays a vital role, which I believe may help better define accuracy.

1.4 History Of Image Processing

Video of sporting events has often been used both to test image processing methods and to acquire data in its own right. Sports analysts are always looking for new ways of summarizing events, creating new measures to evaluate players, and improve existing theories , all of which can benefit from image processing. Although the findings described below are a good place to start, it is important to discuss image processing in general as well.

1.4.1 Image Processing Currently Applied To Sports

Chen et al. (2010) attempts to do something very similar to what we are attempting, albeit with different goals in mind: the acquisition of the strike zone. They first find the plate, using the knowledge that home plate is a distinct white/gray horizontal stripe that is located generally in the same position on the image. They then remove pixels with low intensity that are in the middle of the screen (since they assume the plate will have a high intensity). They then find the plate based on the remaining pixels and knowing the size of the object in question.

Chen et al. (2010) also mentions the technology behind *K-zone*, and discusses its limitations: "...an operator is required to use a specific camera and PC to locate the top and bottom limits of the Szone for each batter". They then discuss their methods of using contour-based methods to help determine the strike zone. One of their methods is the Canny Edge Detection method, which is discussed in Chapter 2.

Chen et al. (2008) explores "Spatial Pattern Recognition" to identify what area of the field a particular image represents, using the dominance and position of the colors of pixels as the main feature. White pixels play a key role; they help the algorithm find foul lines, home plate, bases, and other field features. Once the area of field is determined, the methods piece together a "play by play" to summarize the game based on where the camera is looking.

The method uses color to identify features, the main distinguishing factor in my approach as well.

Chen et al. (2009) focuses on using properties of physics to help regain the information lost in the projection of three-dimensions onto the two-dimension plane of an image, namely with shooting locations in basketball. It is almost impossible to find a basketball in a single frame; with the information provided by multiple images, this task becomes easier. The physics equations used help reconstruct the movement of the ball in three dimensions. The catcher's mitt will likely not move by very much in the third dimension and so while the ideas are very interesting, they are likely not applicable here.

1.4.2 General Image Processing: The Canny Edge Detector

Canny (1987) introduces what would become known as the Canny Edge Detection algorithm (discussed in more detail in Section 2.2. The goal of edge detection is to reduce the amount of data to be processed but retain the structural information about an object's boundaries. This approach could potentially be very useful not only since it reduces the amount of information that needs to be processed, but also it naturally forms "shapes," such as (hopefully) the glove. An algorithm that is able to to do both of these sub-steps is exactly what will be needed for application on a large scale.

Worthington (2002) attempts to improve upon Canny's edge detector by "using a curvature consistency process to adjust the gradient direction estimates prior to finding the zero crossings in those directions." He points out that a weakness of Canny is its reliance on gradient estimates. To help alleviate this dependency, the author attempts to create a physically-motivated adaptive smoothing technique. He increases the consistency of the neighborhood curvature labels as well as adjusts gradient estimates to reduce differences between shape-index labels of neighboring pixels.

Ding and Goshtasby (2001) discuss faults and other possible improvements to the Canny method. Ding's enhanced Canny algorithm calculates "major edges", which are the original edges produced by the classical Canny method, as well as "minor edges", which are "pixel[s] where its gradient magnitude is larger than those adjacent to it and at opposing sides but not necessarily in the gradient direction." He then traces minor edge contours and keeps only the parts of the contours that connect the major edges. This is an improvement on the final step of the algorithm to allow for more flexibility in the definition of an edge.

Although the suggestions of Worthington and Ding are both improvements to the original Canny algorithm, they are essentially irrelevant to my interests. As I will discuss in Chapter 2, Instead of taking the final step to declare pixels as part of an edge or not, I focus on the gradient estimation steps and do not need to classify the exact edges.

Chapter 2

Images

In its most raw form, the information that needs to be collected comes from game video. For the past 30 years, almost every MLB game has been broadcast at least on regional television. Given this history, I have the potential to create this accuracy statistic for every pitcher that has played during this time period. Because of the versatility of the measure, it does not matter if a pitcher plays just a few games in his career or if they have been pitching for 15 years. Analysts can interpret the measure as they see fit. For example, the measure can be averaged over any number of games to allow for both short-term and long-term interpretations. One disadvantage of using video as the source of the data is that I will be unable to collect information on pitchers who played before this time period. Many sabermetrics are designed specifically to do this. For example, OPS+ is a modified version of OPS that takes into account the league average and the park where the player played his home games.

I will only look at video at a pitch-by-pitch level; each pitch, which is a small segment of video, will be treated as a separate dataset. If I wish to maximize the automation, the datasets will need to “communicate with each other.” In order to quantify the information that is necessary, the video is then broken down into images. Modern broadcasts are shot at

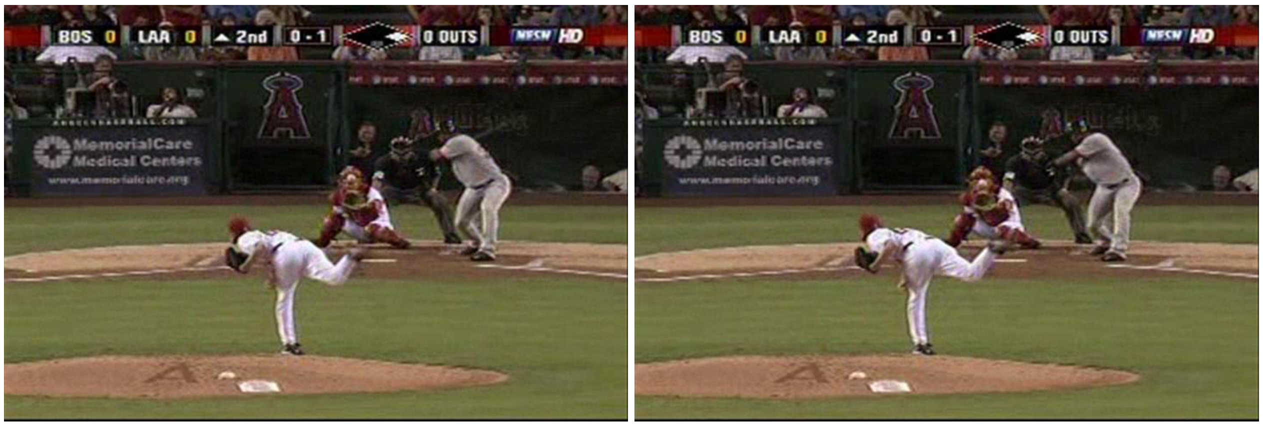


Figure 2.1: Successive Frames of a Pitch

a frame rate of 24 frames per second, resulting in about 72 frames per pitch. The amount of movement between each successive image is actually quite small. So, when trying to find an object in successive images, the best place to look first is where the image was in the previous image. As shown in Figure 2, given the speed of the pitch, the glove does not have the chance to move significantly.

2.1 Quantifying the image

In order to perform any analysis on the images, I must first transform them into a dataset. The most common way to represent images is to record the Red, Green,& Blue values (RGB) of each pixel. One can use a linear combination of these primary colors to produce almost any other color. In terms of data structure, a 64x64 pixel image would be represented as a three-dimensional matrix with the dimensions [64,64,3]. RGB values are the de-facto standard when analyzing images.

Another popular method is using a combination of the Hue, Saturation, & Value (HSV) of each pixel. These values are cylindrical-coordinate representations of RGB values. Hue is essentially what we visualize color as, Saturation is the dominance of the Hue in the color, and Value represents the brightness of the pixel. The purpose of the transformation is to create a mapping that more accurately represents the way humans see colors. Unlike RGB, HSV does not define colors in relation to primary colors, which allows them to be more easily defined by humans. I then transform the original HSV values from a cylindrical-coordinate representation to a conical-coordinate representation. The reasoning behind this transformation is to give more weight to the Value of the image and to allow for different values of the Value to affect the weight of the Hue and Saturation. The transformation is done through the following equation:

$$sv * cos(2\pi h), sv * sin(2\pi h), v \tag{2.1}$$

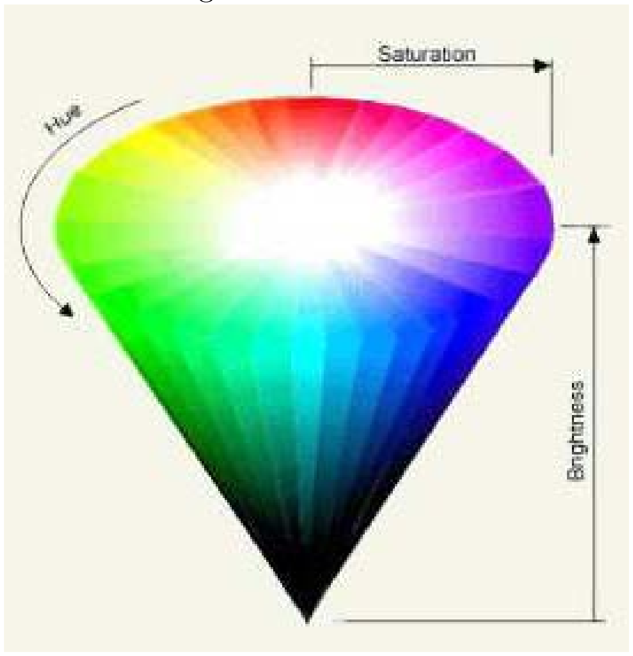
Value is given the most weight because when thinking about how humans see color, when a room is dark (Value is low), they cannot easily distinguish between colors. The resulting spectrum is shown in Figure 2.2. As you can see, when the Value is low, the Hue and Saturation do not matter very much, however when you increase the Value, the Hue and Saturation carry more weight. If you look at the bottom of the cone, the entire region is very dark, regardless of the Hue or Saturation values, showing how the Value outweighs their measurements and is dominant.

Using the intensity of each pixel, or the grayscale value, is also a common representation, although it contains less information than the two previously mentioned. Since I am starting with color images, if I want the greyscale values, which are needed for the Canny Edge Detection algorithm mentioned in Section 2.2, I just take the following specific combination of the RGB values.

$$Gr = .30 * R + .59 * G + .11 * B \tag{2.2}$$

Pixels in an image are not arranged in a random order. It is important to include the

Figure 2.2: HSV Cone



x and y coordinates of each pixel in my analysis. When dealing with object recognition, knowing information in adjacent pixels is critical. Because the x and y values range from 1 to the dimensions of the image, it is important to scale the values to also range between [0,1]. For all three representations, I also multiply a weight (w) to the x and y values to allow me to tune their importance relative to the other variables. The choosing of this weight will be discussed further in Section 4.6.

Each of these representations of the image has its own advantages: RGB values are the easiest to obtain from the image, HSV values are the most "true" representation of the color, and Greyscale values are the simplest way to represent pixels. I will use a combination of these representations throughout the analysis, each being used where appropriate.

2.2 Canny Edge Detection and Gradients

As mentioned in Chapter 1, one option to transform the image & reduce the amount of data was using the Canny algorithm. Because it finds edges, if tuned correctly, it would find the edge of the glove itself, a key step in object recognition. Canny algorithm has four steps: Noise Reduction, Finding the Strength of the Edges, Non-maximum suppression, and Hysteresis.

The first step in the algorithm is to filter out any noise in the image using a Gaussian filter. The result is a blurry version of the original image where pixels that would be considered outliers are toned down and have less weight.

The next step in the algorithm is to find the strength of the edges, which is done by taking the gradient of the image. Using an edge detection operator, the first derivative in the x and y directions are found. The gradient and direction are then calculated. The strength of the gradient is defined as:

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.3}$$

where G_x and G_y are the gradient in the x and y directions, respectively. The direction of the edge is then calculated as:

$$\theta = arctan(\frac{G_y}{G_x}) \tag{2.4}$$

The third step in the algorithm is known as non-maximum suppression. This step performs a search throughout the image to determine the edges. The search looks at the gradient angle and compares its intensity to the neighboring orthogonal points. For example, if $\theta = 90^\circ$, the pixel is considered an "edge" if its intensity is greater than the intensities in the perpendicular directions: in this case, north and south.

The final step, hysteresis, is used to eliminate streaking (when the edge "breaks" due to going outside of threshold). Instead of using a single threshold value, to determine if a pixel is part of the edge, it creates a range; anything above the upper bound of the range is accepted as an edge, anything below the lower bound is considered not part of the edge, and anything that falls within the range is considered an edge if the pixel is adjacent to one that is already considered part of the edge.

Figure 2.4 shows the results of the Canny algorithm on a picture of a pitch.

2.3 Improving Canny

Recall that the process takes a video as input, not a single image, and so I have more information available for the algorithm than in the standard case. Because all of my analysis is on video that has already been collected (as opposed to live television) I am able to use information in the future to help better understand the present. The added dimension, time, significantly improves the algorithm. In order to implement this added functionality, instead of deciding on whether or not each pixel is part of an edge, I merely take the gradient of the pixels and use that information. Tracking the gradient over time naturally allows for moving objects to "appear" in the image. Static objects, even if their edges are clearly defined, are far less prominent because the gradient over time goes to zero.

The improved algorithm was implemented in C++ with a wrapper for the R language (in which it was previously not written for public consumption), which allows the user to adjust the gradient in each direction as well as how much to smooth the image on each axis. Smoothing allows the user to change the amount of noise reduction done on the original image. Figures 2.2-2.10 summarize the new algorithm, which is extremely flexible and can provide users with various different results, and is being processed for posting to the Comprehensive R Archive Network (CRAN).

Although the glove will likely be moving throughout the sequence of frames, there is still

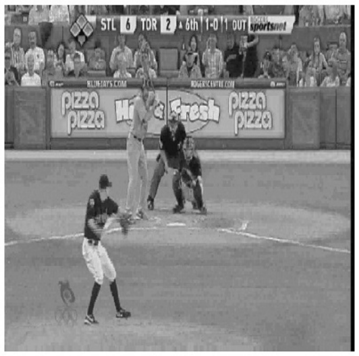


Figure 2.3: Grayscale Image 1

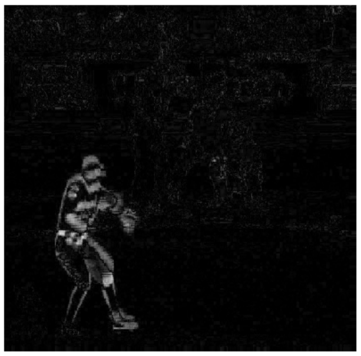


Figure 2.4: Difference of Image 1 and Image 2

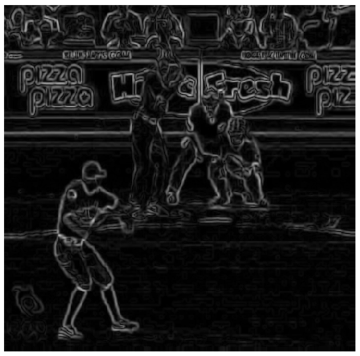


Figure 2.5: No Smoothing & Time Not Considered

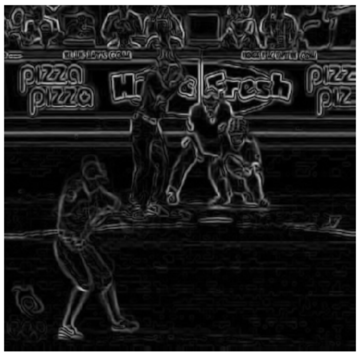


Figure 2.6: No Smoothing & Time Considered

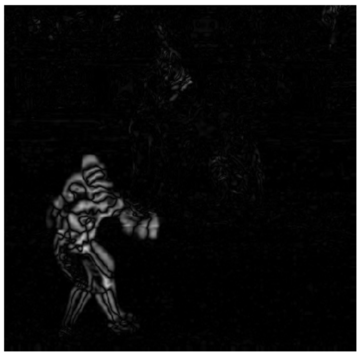


Figure 2.7: Smoothing = 5 & Time Not Considered

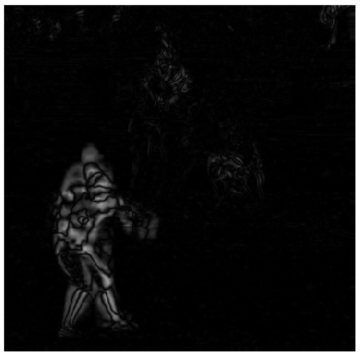


Figure 2.8: Smoothing = 5 & Time Considered

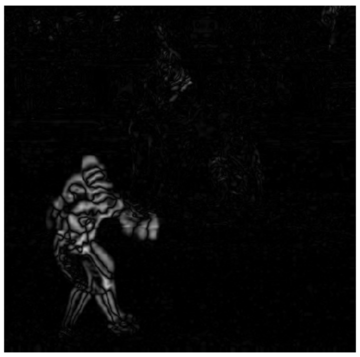


Figure 2.9: Smoothing = 10 & Time Not Considered

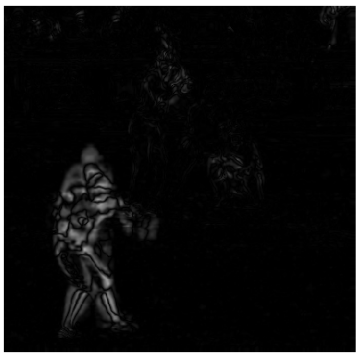


Figure 2.10: Smoothing = 10 & Time Considered

the possibility that it does not move at all. Because of this, I cannot make the assumption that the glove will always be highlighted after running the improved Canny algorithm on an image sequence.

2.4 Computational Problems

The computational complexity of this analysis is largely dictated by the size of the image. The most common images analyzed had an average of 348,480 pixels. Each of these pixels have associated information such as x and y location as well as RGB / HSV / Grayscale values to describe their color. Due to the large size, I design my algorithms attempting to control the computational complexity.

When needing to identify a spatially cohesive set of observations (an "object") in a large dataset, it is helpful to think about how one would identify an object by eye looking at an image. An "object" is a group of pixels in close proximity that have similar features. Thinking statistically, it is a group of pixels where there is low within-group variation, and high between-group variation (i.e. different than other objects. A popular method of detecting objects is clustering.

Chapter 3

Clustering Methods

Cluster analysis is a statistical tool which assigns observations into subsets of the data such that observations in a cluster are more similar to each other than to observations in other clusters. There are several different approaches to cluster analysis, all having unique advantages and disadvantages and relying on different size and shape assumptions. After weighing the pros and cons of several methods, I decided that the best clustering method for my analysis would an implementation of the K-Means algorithm due its speed and simplicity.

3.1 K-Means Clustering

K-means[1] is a fast and easy-to-implement algorithm. The K-means algorithm tends toward compact spheres. The primary objective is to partition the observations into K clusters such that the within-cluster sum of squares for clusters $C_1, C_2, ..C_K$ is minimized:

$$\sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} (\mathbf{x_i} - \bar{\mathbf{x}}_\mathbf{k})^2$$

The algorithm begins with K initial random cluster centers. It then alternates between

[1]Need citations

the two following steps until convergence: *Assignment*: Assign each observation to the closest cluster center. *Update*: Calculate the new center of each cluster by finding the centroid of its assigned observations Hartigan and Wong (1979).

Because the algorithm partitions into compact spheres, it may be beneficial when trying to look for something like a glove; however, it could cause problems on non-spherical objects, such as grass. It is also important to note that the solution depends on the initial set of starting centers, which the user needs to pick in advance. When thinking about an image of a baseball game, it may be impossible to determine a true number of clusters. In order to get a reasonable estimate for K, the most practical method was trial and error. To get a rough estimate I chose 3 values for K: 5, 10, & 20. I then ran K-means on both the RGB & HSV datasets for each K. The results are shown in Figures 3.1 - 3.6. As you can see, the two datsets produce similar results, with some notable differences. Notably, the HSV results seem to put more emphasis on the x and y values. Because the pixels corresponding to the glove will all have very similar locations, I believe HSV will be a better representation of the data for this analysis. I will use this analysis as part of the semi-supervised algorithm that I will discuss in Chapter 4.

3.2 Other Clustering Methods

As mentioned above, there are many different methods of clustering besides K-Means. However, because of the massive amounts of data used, all other clustering methods tried, such as Model-Based Clustering and Hierarchical Clustering, were too inefficient to implement at any scale larger than a few images.

Model-Based Clustering attempts to fit a mixture of Gaussian distributions to the data. The algorithm fits different ellipses by varying their volume, orientation, and shape. The algorithm then picks the “best” as the one that minimizes the BIC criterion. This method

Figure 3.1: K-Means Clustering

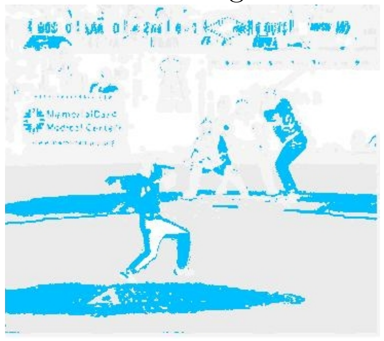


Figure 3.2: RGB, K=5

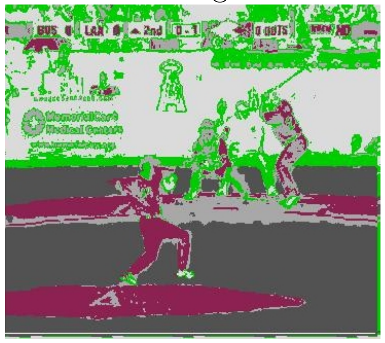


Figure 3.3: HSV, K=5

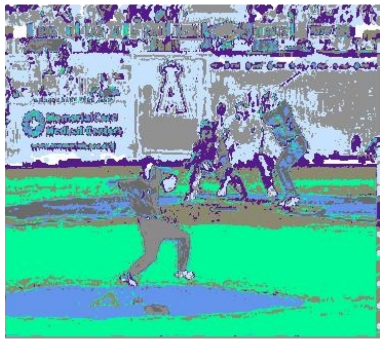


Figure 3.4: RGB, K=10

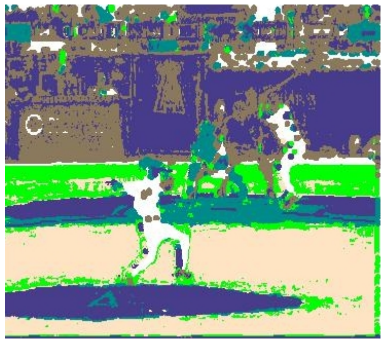


Figure 3.5: HSV, K=10

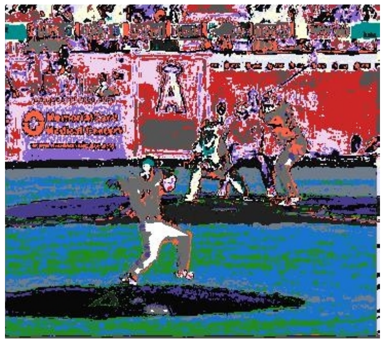


Figure 3.6: RGB, K=20

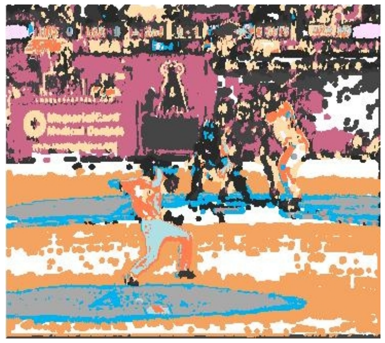


Figure 3.7: HSV, K=20

allows users of the algorithm to input a range of number of clusters instead of having to decide on the number a priori. However, because it has to fit every combination of number of clusters and shape of cluster, it is very time consuming (Fraley and Raftery, 2002).

Hierarchical Clustering builds a dendrogram "tree" of possible groupings of the data. The algorithm uses distances in the covariates to determine the structure of the tree. The most common method of hierarchical clustering is known as agglomerative, where each observation starts in its own cluster and clusters are then merged int order of closeness until every observation is in the same cluster. A major advantage of this method is that you do not need to pick the number of clusters a priori; you can run the algorithm, then look to see what the best number of clusters is for your given data, however picking the "correct" number of clusters can be a very subjective choice; the "best" place to cut the dendrogram may not be obvious (Mardia et al., 1980).

Although these other methods are deterministic (compared to the non-deterministic K-Means) and more flexible (in both the assumptions needed to be made beforehand as well as the the results produced), the computational power required to run these algorithms on a large dataset is far beyond the scope of this project.

Chapter 4

Search Methods, Given Processed Images

Once the image has been converted to a dataset, I then turn to object recognition. In order to do so, I developed different search algorithms. The methods outlined below start with finding the plate in one image. They then move to find the glove in an image. Finally, I present an algorithm that finds the glove in an image, and tracks the glove through multiple images. Although the ideal search method requires no input from humans, my algorithms include some interaction in order to maximize the effectiveness as well as minimize the amount of time required for the algorithm to run. This interaction happens at the beginning of the algorithm & then intermittently to keep the algorithm "on track." For example,I ask the user to click on the object of interest in the first frame to allow the algorithm to reduce the search area from the entire image to the local pixels around the click. This interaction is the semi-supervised part of the algorithm.

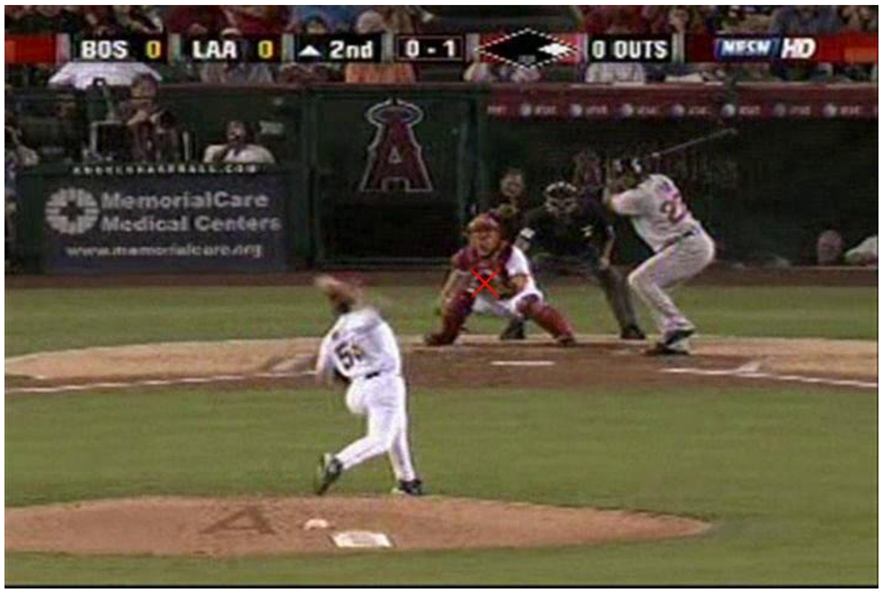


Figure 4.1: The location where the User Clicks is marked with the Red X

4.1 Inputs

As mentioned above, there are many ways to represent the image, such as RGB values or Cluster assignments from K-means, all of which could be inputs in the search methods developed. For example, one might just have grayscale values to input into the search algorithm. The same algorithm also is able to take in Red, Green, and Blue values of the image, as well as cluster assignments, and different weights for each variable.

My algorithms initiate with the "human input" mentioned above. The first frame of the sequence is presented to the user, and they click on the object they with to identify (See Figure 4.1). The pixel location is then recorded and is fed into the algorithm.

For each search method below, I will show the results of the algorithm on both the plate as well as the glove to show how different shapes cause difficulties for the different methods.

4.2 Up/Down, Left/Right Search

The first and most rudimentary search method created was essentially an exercise to help get a better understanding of searching images. The basic algorithm is written below:

1. User clicks on the object in the picture
2. Mark pixel as starting pixel
3. User selects threshold for comparison
4. "Go right"
   A. "compare" starting pixel to the pixel immediately to the right
   B. If the difference is less than the threshold, mark pixel "in" object and mark it as the current pixel
   C. "Compare" pixel immediately to the right of new current pixel
   D. Continue until comparison breaks threshold
5. "Go left"
6. "Go up"
7. "Go down"

Compare, in this case, means take the difference between each variable for said pixel. Once a difference is obtained, I need to compere it to a threshold to determine if the difference is acceptable or not. Picking a threshold is subjective and can be difficult to find one that produces desirable results. I go into more detail about picking thresholds in Section **??**

The goal of this search method was essentially to find the edges of the object and to learn how to utilize tuning parameters. As shown in Figures 4.2, the algorithm returns the

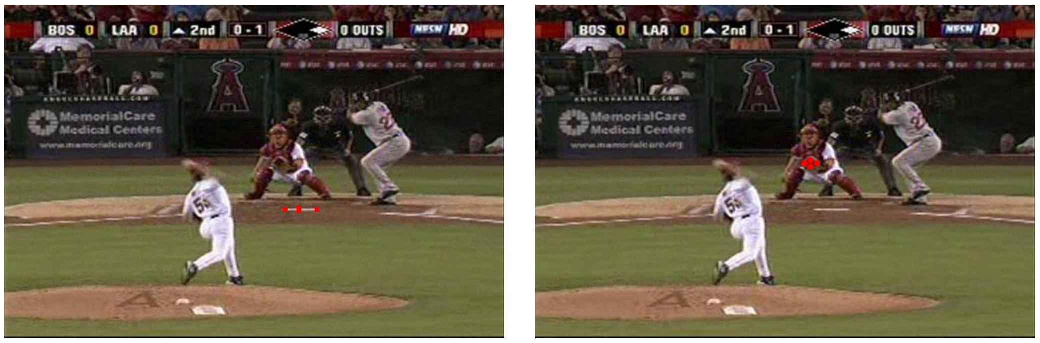


Figure 4.2: Up Down Search on the Glove and Plate

furthest points due north, south, east, & west of where the user clicks. This algorithm is very quick and returns a rough representation of the image. However, it does not find all points in the object, just those that lie on the same x or y plane as the initial click.

4.3 Spiral Search

The next incarnation was the first attempt to find the entire object, not just the boundaries. The spiral search is similar to the first search, however it takes a different path. Starting with the pixel clicked on, it searches up 1 pixel, then right 1 pixel, followed by down 2 pixels, then left 2 pixels, and so on until the comparison results a difference less than the threshold. The result is that the object takes a square-like shape as shown in Figure 4.3. If I want to find the glove, this approach would probably suffice, however for an object like the plate, where one dimension is much larger than the other, it causes severe problems.

4.4 Spray Search

The final algorithm, which consistently is able to find an object in an image follows the steps below:

1. User clicks on the object in the picture

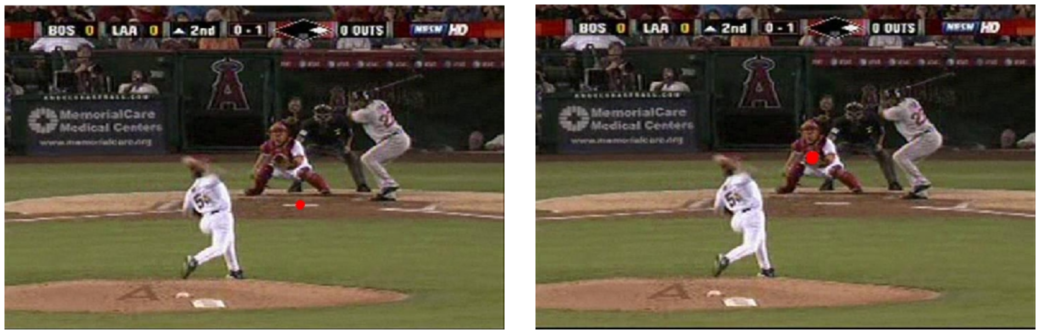


Figure 4.3: Spiral Search on Plate and Glove

```
2. Mark pixel as "comparison pixel"

3. Add pixel to "to be checked" queue:

4. While there are pixels in queue

    A. Check to see if pixel is "similar" to comparison pixel

    B. If true

        i. Mark pixel as "part of object"

        ii. Add all adjacent pixels that haven't been checked to "to be checked" queue

    C. Else

        i Mark pixel as not part of "object"
```

Using this algorithm allowed objects to take any shape, as long as it was contiguous, allowing for non-traditional shapes. This step is a huge improvement compared to the previous algorithms. The assumption of a contiguous object is reasonable given the application. The biggest disadvantage of this algorithm is the complexity; it takes the longest to run.

4.5 Task: Given Glove, find another glove

Once I know the location of the glove in one image, I can utilize similar methods used in finding the plate to find the glove in the next image. Because there are 24 frames per second, it is impossible for the glove to not be at least partially in the same position as it was in the previous frame, there is always overlap. I am then able to optimize my search over just the area close to the glove in the initial image.

4.6 Tuning Parameters

Each search method mentioned above as well as the Canny and K-means algorithms require tuning of some sort. Because catcher's mitts can be different colors, shapes, and sizes, these tuning parameters will need to be updated for each dataset. To help find the optimal tuning parameters for each method, I developed an applet to hone in on a range near the optimal parameter values. The applet first runs the Spray Search on an image. It then takes the center of the object that the user clicks on, and uses it to reduce the image to just the mitt and close proximity (30 pixels in every direction) around the catcher. After the image is reduced, it runs through the various algorithms and displays the results to the user, who is then asked a series of questions, such as "How did the spray search do?" The user can tell the applet to either increase or decrease the tuning parameter for each method. The algorithm then updates the tuning parameters accordingly and re-runs each method. The process is repeated until the user is satisfied with the results. Figure 4.6 shows an example of what the user would see after they run Spray Search the first time. The user can then decide, for example, to increase the number of clusters in K-means on the RGB values, decrease the weight of the x and y variables in K-means of HSV values, or to leave the Spray Search tuning para mater alone.

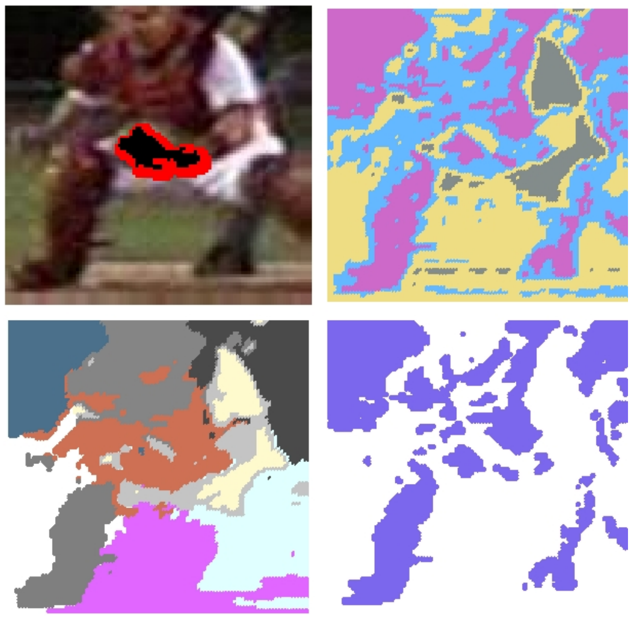


Figure 4.4: Tuning Applet displaying Spray Search and K-Means on RGB, HSV, & Grayscale Values

4.7 Tracking an object over time

The ultimate goal of this project is to be able to track the glove over time, where the user does not need to be involved at every step. Until now, all of the methods described attempt to find the glove in a single image. Although mentioned in Section 4.5, I have yet to implement tracking. Below I discuss my work on tracking the object through multiple frames.

4.7.1 Modified Spray Search

The Spray Search algorithm mentioned above is very versatile. In order to use the algorithm over multiple frames, I modified the code to allow for the coordinates of the "center" of the object as an argument instead of having the user clicking the object. Once this functionality was implemented, I was able to chain multiple iterations of this search on successive images, to track the mitt over time. The steps of the algorithm are:

1. Run original Spray Search on Image 1
2. Find centroid of the returned object
3. Pass in centroid as new "click" to Spray Search on Image 2
4. Repeat for entire pitch sequence.

The pitch sequence and resulting objects found can be seen in Figure **??**. As you can see, this method actually performs extremely well. Although the entire shape of the glove is not perfect, I would still be able to extract the center of the glove, which is ultimately what I am after. One of the biggest problems with this method was the run time. For a 6 pitch sequence, with images with dimensions 720*480, the algorithm took roughly 45 seconds. As previously stated, the Spray Search algorithm can be very computationally intensive. This new algorithm calls the Spray algorithm n times, where n is the number of frames in a sequence.

4.7.2 "Stamp" Method

In order to speed up the process, a new algorithm is introduced. The "Stamp" method still relies on Spray Search, however just for the first image. The basic algorithm is reproduced below: [1]

1. Run original Spray Search on Image 1
2. Store Glove as object
3. Create list of pixels that are in object (Pixel List)
4. In Image 2, for every pixel in Pixel List, "Stamp" the Glove

[1]This needs to be cleaned up

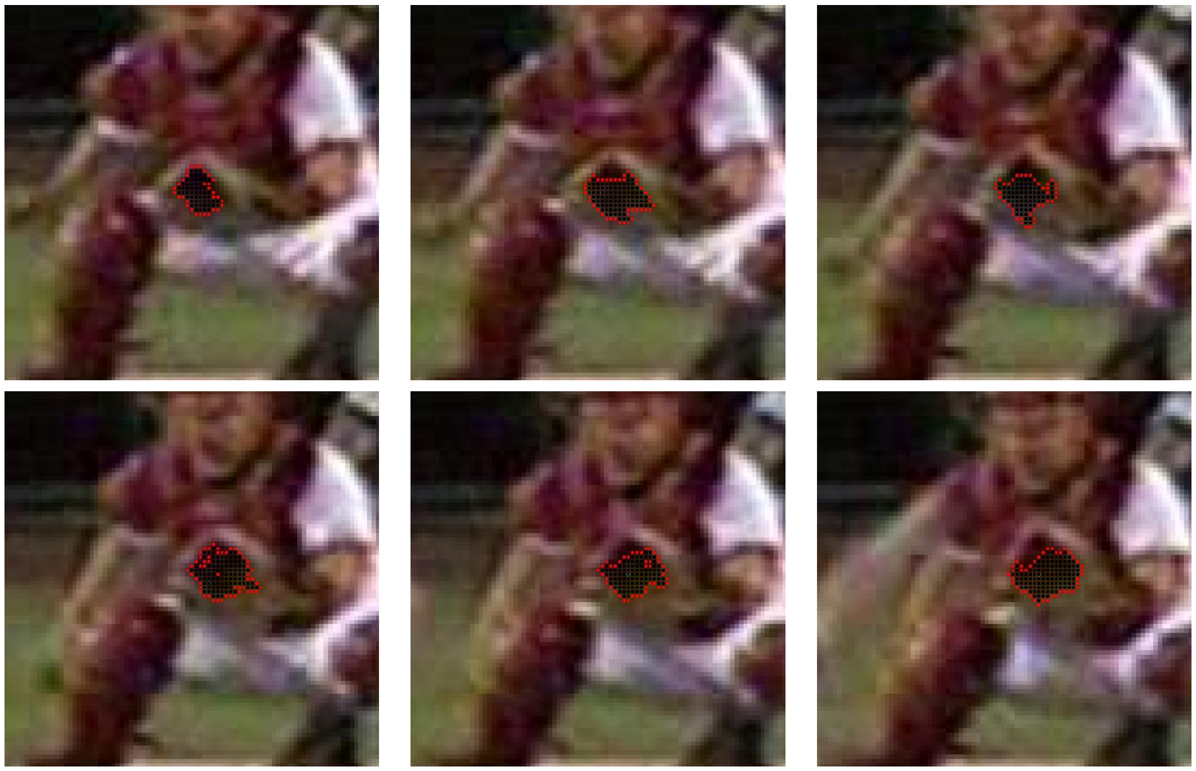


Figure 4.5: Modified Spray Search Across 6 Frames

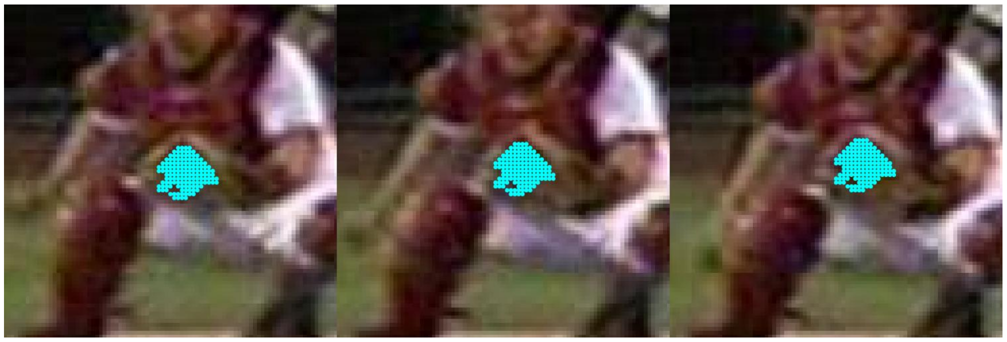


Figure 4.6: Stamp Method Run on 3 Consecutive Frames

A Compare all variables (i.e. RGB values) for each pixel

B Find the pixel in which the summed difference is minimized

5. Mark pixel as new center of glove

6. Shift Glove object to new center

Essentially, the algorithm attempts to "stamp" the original glove on a series of pixels surrounding the original glove. The "stamp" that produces the closest match becomes the new object. This is repeated on each successive image. Figure 4.7.2 shows the algorithm on 3 consecutive frames. Although much faster than the Modified Canny Search, this method also has its disadvantages. Primarily, the algorithm assumes that the object's shape will remain constant throughout the sequence. As discussed earlier, a catcher's glove will likely change shapes slightly from frame to frame as he adjusts it for the incoming pitch. Additionally, a search region is needed to be specified before running the algorithm, which can cause problems if the object moves too far too quickly.

Chapter 5

Developments and conclusions

I have made significant progress in the development of an algorithm to create a measure of "accuracy" for a pitcher. The various search methods created have been able to accurately find the plate, with minimal human input. Each method described in this paper serves a specific purpose and provides valuable information. Combining these methods (as shown in Section 4.7) allows for the mitt to be tracked within a pitch sequence. From this, the distance moved can easily be calculated.

Improving the Canny Edge Detection Algorithm to factor in time can make it significantly easier to track objects that move over time. In its current state, it is not ready to be directly implemented into these algorithms; given its obvious potential as a motion detector, it is the obvious next step that needs to be taken. Similarly, implementing the Cluster solutions provided by K-means on the different datasets would also likely yield improved results.

Along with adding the functionality of Canny, there are several steps that still need to be taken to reach the goal of creating an "accuracy" measure. Although I have successfully created an algorithm that tracks the glove over the course of a pitch, if I wanted to create a statistic to use for MLB players, I would need to be able to track the glove throughout the entire game. This would require knowing when the glove is at the "target" location and when the “impact” occurs.

The ideal solution to this problem would likely involve tracking the mitt in real time, similar to the ball in *K-Zone*, so that decision makers such as managers can have information that is as current as possible; even decisions based on single pitches are known to be enough to influence championships.

Bibliography

Canny, J. (1987). A computational approach to edge detection. *Readings in computer vision: issues, problems, principles, and paradigms*, **184**.

Chen, H., Hsiao, M., Chen, H., Tsai, W. and Lee, S. (2008). A baseball exploration system using spatial pattern recognition. In *Proc. IEEE International Symposium on Circuits and Systems*.

Chen, H., Tien, M., Chen, Y., Tsai, W. and Lee, S. (2009). Physics-based ball tracking and 3D trajectory reconstruction with applications to shooting location estimation in basketball video. *Journal of Visual Communication and Image Representation*, **20** 204–216.

Chen, H., Tsai, W. and Lee, S. (2010). Contour-based strike zone shaping and visualization in broadcast baseball video: providing reference for pitch location positioning and strike/ball judgment. *Multimedia Tools and Applications*, **47** 239–255.

Ding, L. and Goshtasby, A. (2001). On the Canny edge detector. *Pattern Recognition*, **34** 721–725.

Fraley, C. and Raftery, A. (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, **97** 611–631.

Hartigan, J. and Wong, M. (1979). A k-means clustering algorithm. *Journal of the Royal Statistical Society C*, **28** 100–108.

Mardia, K., Kent, J. and Bibby, J. (1980). Multivariate analysis.

Thomas, A. C. (2011). Pitcher Accuracy Through Catcher Spotting: Assessing Rater Reliability . *Journal of Quantitative Analysis in Sports*, **7**.

Worthington, P. (2002). Enhanced Canny edge detection using curvature consistency. *Pattern recognition*, **1** 10596.