18-551 Project Final Report
Spring 2002

The Virtual Zone

Group #11
Robert DiMaggio (rsd)
Louis Trebaol (lpt)
Shao Chiang (schiang)

TABLE OF CONTENTS

1. THE PROBLEM

Over the past 100 years, Major League Baseball has remained virtually unchanged. Set in a stadium battleground, it is an epic struggle between foes. Most of the action rests in the hands of the pitcher and the batter. The pitcher attempts to throw the baseball into the strike zone and outwit the batter. The batter is trying to do everything in his power to hit the ball and place it into play if the ball enters the strike zone. The omnipotent ruler, the umpire, regulates this altercation by defining what is considered a strike and what is considered a ball. He is usually an elderly man dressed in blue that stands behind the catcher and interprets every play. The lack of accuracy in this method is largely attributed to human error, for example blocked sight of ball entry or vision degradation on the part of the individual. Many spectators become angry when the umpire calls a play contradictory to what they collectively saw. Both audiences in the stadium and at home would demand to know if the umpire had erred. We propose overlaying a virtual strike zone on the pitcher's throw in the actual game footage so that every viewer on broadcast will be able to see if the baseball was truly in the strike zone when the ball crossed the plate. This will ultimately strengthen the fairness of the game and not have significant rulings be decided on by the sole discretion of one individual.

2. THE SOLUTION

We propose a digital image processing solution that will be able to use as input a video clip of a baseball game and superimpose an outline of each batter's strike zone. This project was made possible using many different methods of edge detection, motion detection, segmentation, and filtering to allow for this robust system to deliver a rich viewing experience for broadcast audiences.

3. THE PROCESS

Once the pitch begins, our system will determine where the strike zone actually is. We are also assuming only right-handed batters in our system design due to possible complexity issues that may arise. In order to achieve our goal of producing an accurate depiction of the correct size and location of the batter's strike zone, we need to extract information from the video clip. In the Major League Baseball rulebook, it states that the parameters of a batter's strike zone is from the batter's knee to his elbow that falls over the width of the plate while the batter is in his ready position. So, we must find the batter's knee and elbow along with home plate to determine where the strike zone must be placed.

Also from viewing many clips of batter's waiting for a pitch, we observed that the batter and the camera do not have much movement from when the batter gets into his crouched position, ready position, until he begins to swing at the pitch. Therefore we only have to perform the knee and elbow calculation once. The lack of movement also makes creating the zone much easier because the batter will be in an almost constant position for most of the duration of the pitch. The lack of movement of the camera and constant positioning for each pitch allows us to narrow down the positioning of the batter and plate much easier.

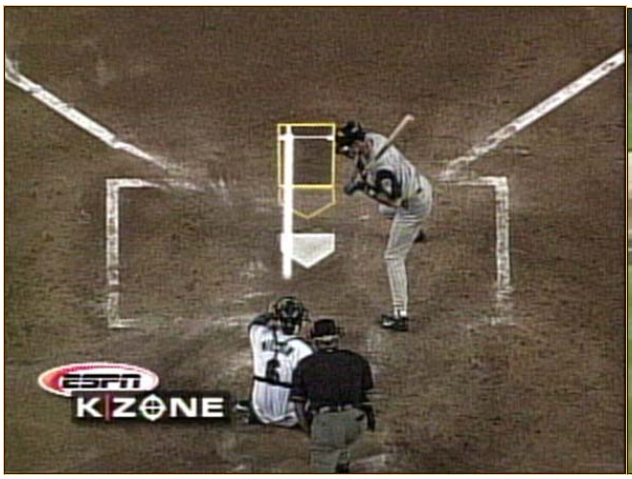
The first thought is to segment the batter and focus on his lower extremities. Once the lower half of the body is noticed, feature detection can be done to find where the knee joint is oriented in the image. Using an edge detector we can determine where the edge of the leg is and then search to find where the edge forms an acute angle. This intersection is the knee. After obtaining the edge map we can use the knee height as the lower line of the box.

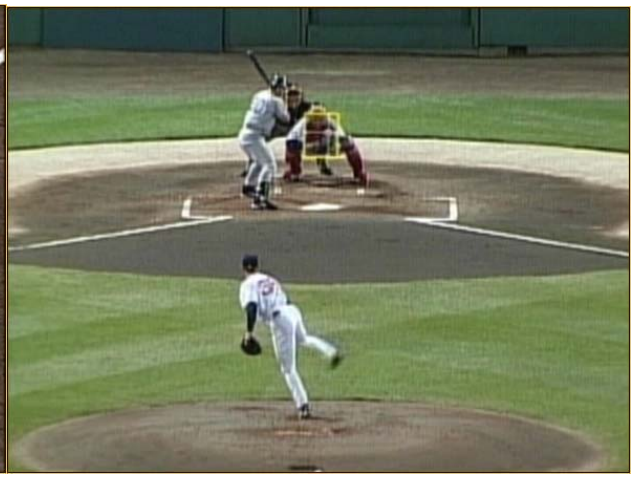
The upper limit of the strike zone is the next problem that must be addressed. While creating the algorithm to find these two points we ran into a bit of trouble. In most of video clips we viewed, the batter was either wearing a dark sleeved uniform or a dark long-sleeved shirt under his uniform. This causes great trouble in finding the elbow because both the umpire and the catcher are wearing dark clothing. The batter's elbow blends into this clutter and is indistinguishable. To solve this problem, we decided to use what we knew about human body characteristics. We know that the distance from a person's knee to hip is about the same length as from the hip to the shoulders. Therefore when a person is in a crouched position, the distance between the knee and the butt is about the same distance as from the butt to the elbow. Instead of finding the elbow, we will focus on finding the butt and calculating where the elbow will be. Having these two heights gives us the top and bottom of the strike zone.

With the ceiling and floor known, we still need the width for completion. This is a comparatively easier task than figuring out the location of the elbow or knee. We will be using edge detection to determine where home plate is located. This should not be a difficult task seeing that the plate is a white rectangle in our video surrounded by brown dirt. We know the plate is located relatively close to the batter on his left side and somewhere below the height of his knee. So segmentation of the plate is not much of a problem. Once the edges of the plate are determined, we can use the side edges as the vertical lines representing the left and right side of the strike zone box. Using the heights from before and our new width measurements we can now place an accurate strike zone on the video clip. This box will remain in the same position above the plate until after the batter swings. In order to insure this we must calculate the edges of the plate for each frame to ensure accuracy encase the camera pans or zooms.

4. OTHER WORK

In early April, a company named SporTVision created a virtual strike zone system entitled the “K-Zone.” This system is a highly accurate pitch calling system that uses three cameras mounted around the baseball stadium, along with a number of sensors, and a truck full of computers. The system has the power to track the path of a baseball up to four-tenths of an inch. Using three video feeds and sensor inputs allow the system to determine a batter’s strike zone along with where the ball was located when it crossed the plate. This system also is able to keep track of the location of all the pitches thrown by a pitcher in a game. It can take this information and map out the path of all of these pitches onto one image showing patterns of the pitcher’s throwing habits. Below are a few pictures from some video clips of what the system output looks like.





Of course the attaining information from this company about their new product was impossible. This system appears to have overcome some of the problems we incurred by adding sensors, using multiple cameras, and no offset camera view of the pitch. These alterations make it easier or even possible to track the ball, determine the outcome of a pitch, and determine the exact location of the strike zone.

5. VIDEO MANIPULATION

Video to bitmap

Our video source was a DVD containing the 2002 playoffs and World Series highlights. We used a DVD ripper called Smart Ripper which was found on this website: www.riphelp.com/articles/dvdripping_1.html. This ripper converted the DVD video into a vob format. Once we had the data in this format we needed to transform it into avi format in order to parse it into bitmaps. Xmpeg software was used to perform this transfer. This software was found at www.mp3guest.com/default.asp. This software was very hard to come across. A small amount of manipulation is needed to acquire this software in the U.S. Then using Adobe Premier 6.0 we were able to make small usable avi clips from the entire DVD movie. This software is available at www.adobe.com. Finally, the last software package used on our video clip was MediaConverter. This software converts avi files into parsed bitmaps and bitmaps back into avi files. Its location is www.mystikmedia.com/mediaconvert.asp.

Bitmap to Raw

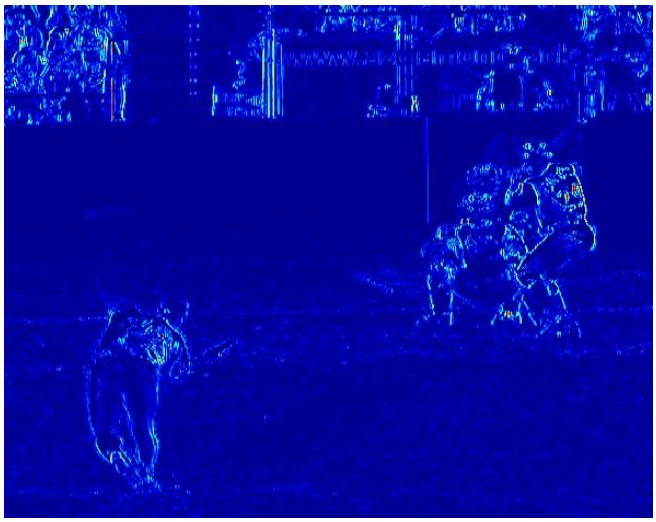
Once we finished with the long process above to give us semi-useful data we still could use the bitmaps on the board. We had to write in C an algorithm that would take a bitmap and turn it into raw data and then transform it back. This turned out to be a greater task then we expected. We spent a large amount of time working on deciphering the header of the bitmap with little success. The problem with the bitmap format there are multiple formats which contain different header sizes and data arrangements. After days of struggling through development of the code, we found that our headers were 54 bytes long. Instead of interpreting the header, we would cut it from the frame data and paste it back on the front

of the frame when we finished with it. Our next problem was that we did not realize that the rows of the bitmap are stored in reverse order. When the bitmap is loaded, stored, and then written we did not discover this formatting issue because we read and wrote in the same manner. Once dealing with our manipulations of the data when it was on the board we realized something was wrong. Once this was solved we discovered a problem we had with color. Instead of the colors being stored as red, green, then blue, they were in reverse order. Finally, after getting through all of this we had an easy way to take data from video down to raw data back to video.

6. PROJECT HISTORY

Our project required the use of baseball video which became fairly difficult to find in the baseball off-season. We found a DVD that contained baseball footage of the pitching camera angle that we needed for our project. After capturing some of the images from the DVD, we began our work of processing the images.

Initial Research



We knew that our project required some form of edge detection to find the batter's knee and elbow. We started our initial work using Matlab and the built in functions included in it. In order to run these operations, the images were converted to grayscale initially. We also tried running edge detection algorithms on each separate color matrix of the RGB values. The results from these tests yielded no helpful results so we had to start looking for some more unconventional methods of edge detection

The Path We Walked

The initial edge detection algorithms that we tried didn't work for two reasons:

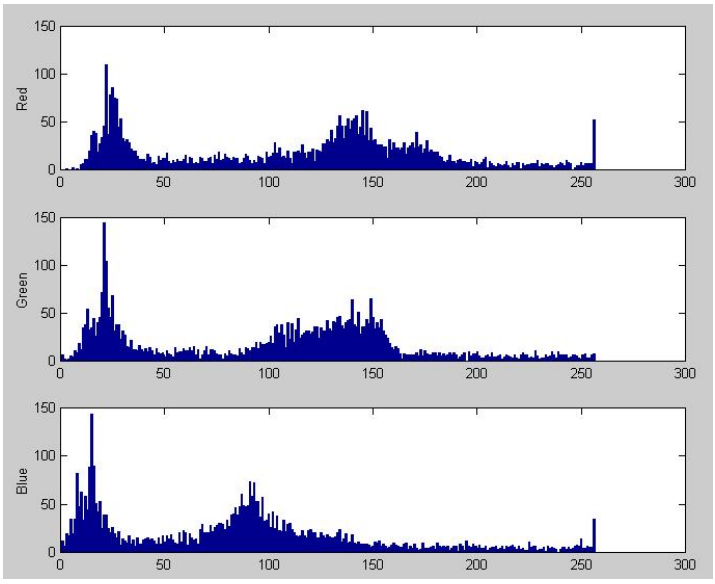
1. The images were had too much noise, especially near the batter
2. The algorithms were not restrained enough for the images that we had

We began looking at the pictures trying to find some similarities in all of them trying to develop a better method for finding the knee. We noticed that in all of the baseball footage, the teams either wore gray or white uniforms. We used this point as the center point for our project. We first tried to use a simple threshold method, but this was not enough. We started to combine different processes in the search for the ever-hiding knee. The next problem was finding the elbow for the upper bound of the strike zone. After a couple failed attempts, we began to realize that there was no way of successfully finding the batters elbow in any image. We began to look into ways of finding an approximation of the batters elbow such as the height of the batter and the height of the batters lower leg. We settled on an approximation that was fairly easy to extract the data after the knee detection ran. We found the line in the image of the batter's butt's furthest out point. The last bit of data that we need to extract was the position of the plate. This proved to be the easiest part of the image processing.

7. THE FINAL ALGORITHM

Knee Detection

The first part of the algorithm separates looks for the batters knee. All of the other data for the algorithm is derived from the position of the batters knee in the first image. The area around the batter is looked at statistically to find the RGB values of the batters pants. This data is then used filter the image. The filtered image is then swept from the bottom right, to the top left looking for the furthest out point of the knee. The function finds the leftmost point on a line and then looks at the next 15 lines to see if another point lays further left than it. It continues this process until there are no points further left than the one it is on. It stores these values as *lineKnee* and *colKnee* so that we know the exact position of knee for the future functions that are going to be called.



The pixel statisitics for the batter 1

Height of the Strike Zone

The next step uses the point of the batter's knee to sweep up from left to right of the filtered batter's image to find the batter's butt. We observed early on that the strike on that the distance from the knee to the batter's elbow is approximately twice the distance from the batter's knee to his butt. By finding the line of the batter's butt we were able to make a fairly accurate estimation of where the batter's elbow would be and therefore where the top of the strike zone should be.

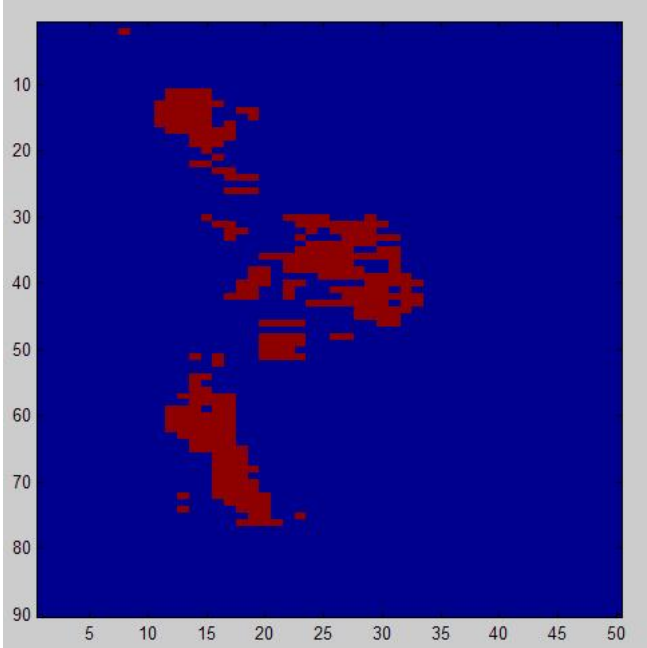


Figure 1 This is the filtered batter area that is used to find the batter's knee and butt

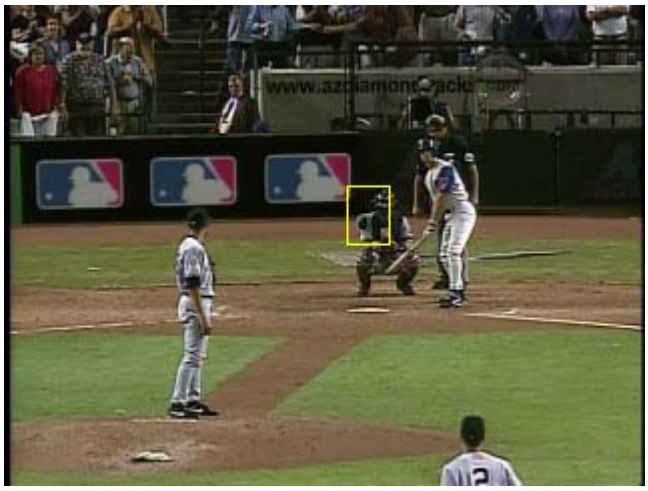
Plate Detection

The final data needed to find the strike zone is the left and right edges of the plate. The position of the knee is used as a reference point to capture the area where the plate should be located in the image. This area is then filtered to find the near-white pixels that should be the pixels of the plate. The plate detection function uses morphological image processing to find the plate in this roughly filtered image. As can be seen in the picture bellow, the plate stands out very well after the filtering. A 1x7 pixel construct is used to find the left side of the plate. After the left side of the plate is found successfully, the right side of the plate is found by looking for the right-most edge on the same line as the left-most edge of the plate. The plate detection operates very successfully from picture to picture.

Processing the Video

There process of taking these functions and running them from 1 picture to running them on a series of pictures took a good deal of additional functions. The noise in the images caused the strike zone to jitter substantially from picture to picture, causing fairly poor results in the constructed videos. This forced us to make the decision of using only the first image to find the position of the knee and calculate the height of the strike zone. Due to the general stability of plate edges and the fact that it is stationary object from

picture-to-picture, the remaining image's strike zones are keyed off the position of the plate. There were issues of the plate edges changing slightly from picture-to-picture. To solve this problem the width of the plate was computed and run through a low-pass filter. The amazing part of our project is that we were able to run our much faster than real time. This is a significant advantage over our competitors in this quest for the perfect strike zone.



8. USE OF THE EVM

What Does the EVM Do?

The EVM is responsible for performing all of our projects calculations. All edge detection, filtering, and frame altering are all done on the EVM. Our C code loads a bitmap image onto the board one at a time just like in lab 3. Once the image is on the board all necessary calculations and augmentations are performed on the frame. Once this is complete, the frame is sent off the board back into computer memory and the next frame is read onto the board. This cycle continues until all frames from the video clip are manipulated.

Memory Allocation

On-board Memory:

Name	Description	Type	Dimensions	Size (Bytes)
Frame	Input frame and augmented output frame	Unsigned char	320x240x3	307,200
batterFilter	Segmentation containing the batter	Unsigned char	45x90	16,200
Plate.vect	Segmentation containing the plate	Unsigned char	80x40	12,800
Map.colorMap	Three vectors for grey detection	int	256x1x3	1,536
Misc.	Variables for strike zone computation	Int	1x19	38
TOTAL				**337,774**

EVM Timing

(For a 10 frame trial)

Program Portion	Cycles	Seconds
Load and Write Frame	17,753,380	.135
Color Detection	1,766,290	.013
Knee Detection	239,203	.00179
Butt Detection	3,159,596	.0237
Plate Detection	9,657,325	.0726
Drawing Strike Zone	3,479,402	.0261
Total	**36,055,196**	**.27**

*It appears that our system works in real-time.

What Doesn't the EVM Do?

The Pc is responsible for converting .avi files into bitmap frames or vice versa. The reason the EVM does not do this in our project is because we could not find C code that could perform this task and it would have been too time consuming to create code that could. So we downloaded a software package that could perform these tasks for us. It is named… and can be found at…

9. FUTURE IMPROVEMENTS

All Batters

Presently our system only works for right handed batters. Making the system work for both right and left handed batters would make the system more robust and is a necessity if the system were to be implemented. A mechanism to determine what side of the plate the batter is positioned needs to be created. But beyond that it seems that the color and edge detection algorithms could be easily altered to look for the body parts of a left handed batter.

Queue Off of the Pitcher

At this time, the strike zone is created from the first bitmap the EVM receives. This means that all video clips which are fed to have the strike zone created must be queued so that the batter is in his crouched position and the pitcher is about to throw. Developing an algorithm to determine when the pitcher begins his windup would be very beneficiary. The video clips would not have to be cropped to meet such strict specifications and it would probably add to the viewer's experience.

Video Clip Quality

The resolution of the video clips must be produced if this system were actually implemented. A resolution of 320 x 240 is not high enough for broadcast television. The reason we used this low resolution, especially since our video source was a DVD, was because of the limited memory and the speed of the EVM. We wanted to avoid memory storage problems on the board to keep the speed down. Also when doing all of our calculations and manipulations we wanted to make the process as quick as possible so we also only used 15 frames per second. We believed processing frames of a

high resolution at 30 frames per second would make our system very slow and hinder is perception.

Speed

Optimizing our code would improve the speed of our system. While optimizing we would have to look at saving more time in transferring the frames on and off the processor. This will increase our speed greatly and make the system work closer to real-time, which is our ultimate goal.

10. REFERENCES

Bitmap Information

http://www.javaworld.com/javaworld/javatips/jw-javatip43.html

http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/gdi/bitmaps_87eb.asp

DVD ripper: Smart-Ripper

www.riphelp.com/articles/dvdripping_1.html

.avi->bitmap: MediaConvert

www.mystikmedia.com/mediaconvert.asp

vob->avi: xmpeg

www.mp3guest.com/default.asp

avi->avi clips: Adobe Premiere 6.0

www.adobe.com

10. APPENDICES