Report Number: WUCSE-2011-99

2011

Real Time Baseball Augmented Reality

Authors: Adam Kraft

As computer hardware becomes increasingly powerful, there is an ongoing trend towards integrating complex, legacy real-time systems using fewer hosts through virtualization. Especially in embedded systems domains such as avionics and automotive engineerin

2011-99

Real Time Baseball Augmented Reality

Authors: Adam Kraft

Abstract: As cellular phones grow faster and become equipped with better sensors, consumers are seeing a rise in augmented reality applications available on their mobile devices. Sports augmented reality is one area that is projected to grow over the next couple of years. We aim to build an application intended for use at a live baseball game. Our application optimizes a best-fit homography to locate the structure of the playing field. From there, we are able to visualize information and statistics directly onto the user's mobile device. This is accomplished in real time, requiring minimal input from the user.

Type of Report: MS Project Report

Real Time Baseball Augmented Reality

Adam Kraft
awk1@cec.wustl.edu

1. Introduction

As cellular phones become faster and equipped with better sensors, consumers are seeing a rise in augmented reality applications available on their mobile devices. These applications span a multitude of topics, including city navigation, astronomy, and gaming. Our project aims to bring a more interactive viewing experience to people at a live baseball game. Nowadays, television broadcasts of baseball games display several statistics to viewers at home. However, these statistics cannot be viewed while attending a live game. With augmented reality, we hope to bring a similar viewing experience to users watching the game from the stadium.

Our application detects the structure of the baseball field and overlays statistics and other information on the user's mobile device. To do this, we calibrate a homography mapping from a pre-processed baseball field image to the real world baseball field observed through a mobile device. To maintain this mapping, we track several key points and find the direction of motion within a scene. After establishing the layout of the field and ability to track it over time, we are able to display visually exciting statistics onto the real world view using the mobile device.

We also design our application as a framework for future developers to use. Although we choose to display statistics, other developers could build on our framework for social applications or gaming applications that use aspects of the baseball field. Or, someone could simply display a larger variety of statistics than our current application. Regardless of the end goal of future applications, our framework provides an important step in recognizing the baseball field for any baseball augmented reality application.

1.1 Previous Work

Even before the invention of modern smart phones, augmented reality applications were in use. The NFL was an early adapter of augmented reality, showing a virtual yellow line to mark the first down on televised games. To accomplish this, the television cameras are equipped with sensors to read the camera's parameters, such as roll, pitch and zoom. This allows the coordinates of football field to be know as the cameras move around the field, letting a yellow line to be drawn at each frame.

Recently, mobile applications have been created, due to the development of phones with faster processors, better cameras, and smarter sensors. Many phone applications, for example AR Tower Defense, require the use of pre-learned patterns that the application can easily detect and track (Figure 1.1). These applications are mostly intended for use on a smaller scale, such as on the surface of a table. Other applications target larger, outdoor scales, including cities and parks. These applications, such as Wiktitude and Layar, rely on GPS and compass sensors to locate shops, landmarks, and other interesting areas (Figure 1.2).

Figure 1.1- AR Tower Defense uses pre-learned calibration patterns

Figure 1.2 – Layar relies on GPS and compass senors.

2. Motivation

Today, sports enthusiasts are accustomed to vast amounts of statistics available at the touch of a button. Due to fantasy sports leagues, fans are constantly checking sports websites to get the latest scores and statistics. Television broadcasts are adapting to the new-age viewers by showing more statistics in innovative ways. In particular, baseball broadcasts now show exciting visuals, like the exact trajectory of a baseball as it travels from the pitchers hand towards home plate. Additionally, they show the exact location of the baseball as it crosses home plate, allowing viewers to review the pitch in comparison to the umpire's call (Figure 2). We believe that a baseball fan watching a live game at a stadium should not miss out on the opportunity to see visually exciting statistics. Furthermore, we hope that our application will help aid in viewers' understanding of and interest in the game of baseball.

Figure 2 - Sportvision's Pitch F/X / K-zone, showing the trajectory of a pitch and where it falls in the strikezone.

3 System Design

3.1 Phone Specifications

For this project, we programmed a Motorola Droid, which was released in October, 2009. This phone is equipped with an Arm Cortex A8 processor, which runs at 600 MHz. For memory, the phone has 256 MB of mobile DDR SRAM and 512 MB of flash memory. The phone is supplied with a 5 MP camera, an accelerometer, a compass, and a GPS sensor. The phone's screen spans 3.7 inches, which includes a 854 x 480 pixel display.

3.2 Methods

The application has two main steps that allow it to find and track the playing field. First, with the help of the user, we find the location of the four bases from the user's perspective. This process, described in Section 4, allows us to find a homography mapping between a preprocessed playing field and the real world view of the playing field. After we have the homography mapping and can find the planar structure of the playing field, we maintain that mapping by tracking key points. By locating and keeping track of the four base points, we are able to keep a structure of the playing field. We can then display statistics directly to the user's view.

4. Calibration

The calibration process consists of three main steps:

1. Preprocessing
2. Finding Candidate Base Points
3. Non-Linear Optimization

After these processes, we have an initial location for the four bases, which we then track. The four bases provide a good set of points because they have standard layout dimensions across all baseball fields and are easy for people to see. The results of the calibration process are visually displayed so that the user can confirm that they are accurate.

4.1 Preprocessing

The first step for calibration involves creating a binary image, where we map each pixel to a *true* if its hue is closer to a predefined grass hue or *false* if its hue is closer to a predefined dirt hue. More precisely:

1. Map Red-Green-Blue image to Hue-Saturation-Value space.
2. For each pixel p:
   p = 1 if (p.hue is closer to grass hue than to dirt hue)
   else p = 0
3. Apply a median filter to resulting image, removing noise

Note that the hue scale has values from 0 to 1, but the scale is radial, meaning that 0 and 1 are next to each other in the scale.

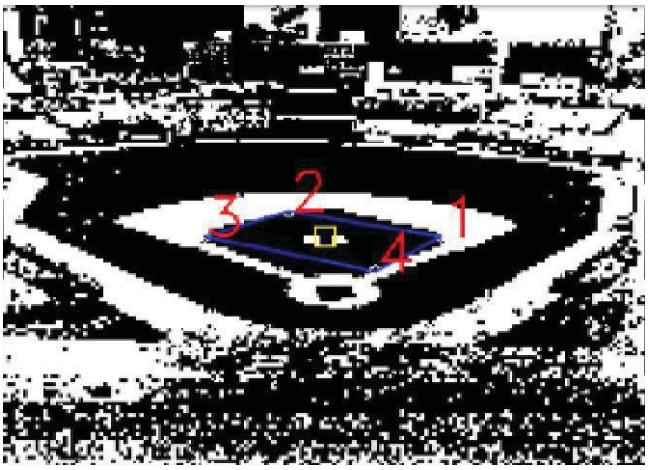

Figure 3 - Binary Image before applying median filter.

The values of the grass hue and dirt hue are hard coded. We collected several dirt and grass hues from sample images and tested binarization from the different values. We found that the grass hues range from around .23-.27 while the dirt hues range lower from .07-.09.

This step produces a binary image where every pixel receives a 1 or a 0, including those not on the playing field. However, we are mostly concerned with the playing field, hoping that the real grass pixels get a *true* value, and the real dirt pixels get a *false* value. A good binary image is necessary for the following calibration steps.

4.2 Finding Candidate Base Points

Next, we want to find four points that are close candidates to the four locations of the real bases. This provides a good starting condition for the optimization in the following calibration step. During this step, we draw a yellow box at the center of the image, and ask the user to point this box at the pitcher's mound. We then extend several lines out from the center of the pitcher's mound to the inside of the infield, where the dirt meets the grass. We collect the end points from the longest line and the second longest non-neighboring line (four points total) as our four candidate points (Figure 4).

The four bases are arranged in a square in the infield, therefore, the longest lines would represent the diagonals of the square from an aerial view. In reality, the dirt pattern of the infield is not always square from the viewer's warped perspective, so that the longest lines do not always contain good candidate points for all four bases. In practice, this metric is usually successful for collecting excellent initial candidates for two of the bases, preparing the program for the optimization step to locate the remaining two bases.

After we have captured the four points, we display the points to the user. Since the user could be sitting in any area of the baseball stadium, the ordering of the bases must be determined by the user. We provide a button that allows the user to rotate the order of the bases until the four candidate points are positioned closest to their respective bases. The four bases will always have the same counter-clockwise ordering.

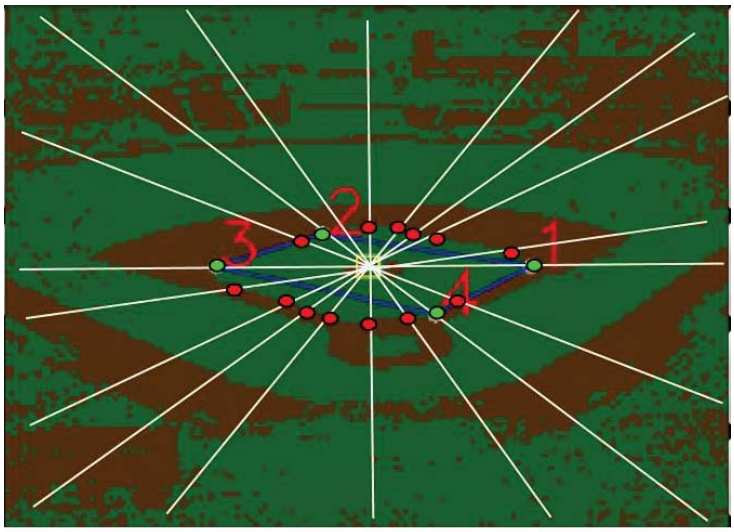

Figure 4 - Display of candidate lines and endpoints. The green points are returned as the 4 candidate points.

4.3 Non-Linear Optimization

Once we have our four candidate base points, we want to find the locations of the four bases more precisely. Having four corresponding points allows us to solve for a homography mapping between a overhead image of a baseball field and the real world view. We still request the user to point the center of the image at the pitcher's mound to begin the optimization.

To solve for the optimal point locations, we could simply optimize over the four X-locations and four Y-locations of the bases (Figure 5.1). In practice, this works relatively well, although the optimization can get stuck in some unfavorable local-minima. Instead, we utilize the fact that lines connecting opposite bases run through the pitcher's mound. First base lies on a line with the pitchers mound and third base, while the same is true for second base and home plate. With this fact, we now only have to solve for six parameters instead of eight, making the optimization faster and more accurate (Figure 5.2):

1. The angle between the horizontal and the first base point
2. The angle between the horizontal and the second base point
3. The distance from the pitcher's mound to first base
4. The distance from the pitcher's mound to second base
5. The distance from the pitcher's mound to third base
6. The distance from the pitcher's mound to home plate

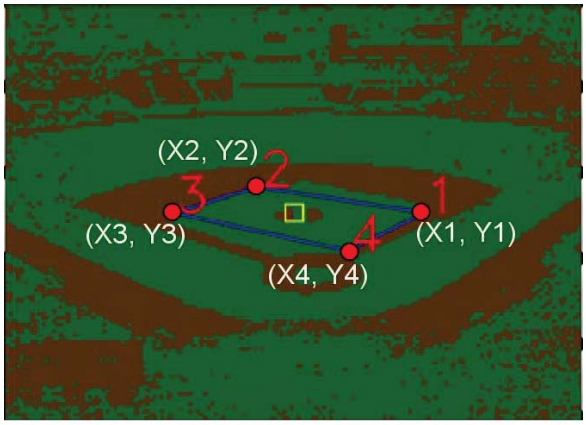

Figure 5.1 - Optimizing over 8 values.

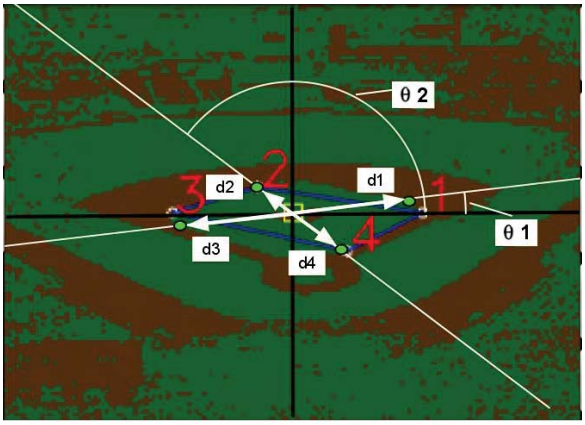

Figure 5.2 - Optimizing over 6 values.

We then run Nelder-Mead optimization over the six initial values, providing a score function described in Section 4.3.1. We continuously rerun the optimization for 35 steps and show the updated results to the user. The user is asked to stop the optimization process when he or she can see that the optimized points are close enough to the actual base points. Once the user signals that the points are close enough to the actual locations of the bases, the calibration process is complete.

4.3.1 Score function

Our score function, given the 6 optimization values, is:

1. Find the 4 resulting points (x and y values) from the 6 input values.
2. Solve for the homography between the 4 resulting points and the 4 locations of the bases from the pre-processed baseball field view.
3. Map the real world binary image onto the pre-processed binary view.
4. Return the number of mismatched pixels for a pre-selected region around the infield.

Optimizing this function results in a successful line up of the shapes of the two infields. We are not actually concerned with finding a global minimum of this function. Instead, we are only concerned with finding a homography mapping that is visually appealing to the user.

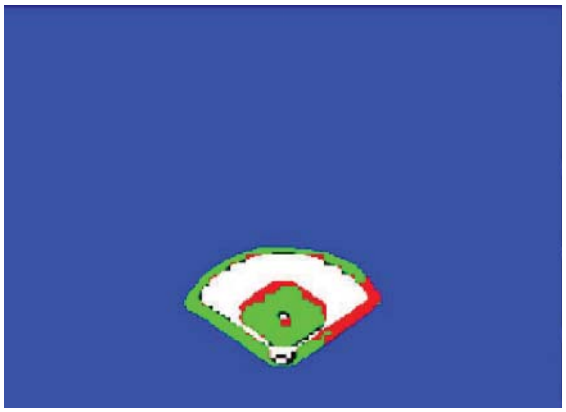

Figure 6 - Sample score image. Dirt matching pixels in white, grass matching pixels in green, dirt mismatching pixels in black, and grass mismatching pixels in red. Blue pixels are ignored.

4. Tracking

Once we have successfully located the four bases, we want to track those locations, so that the user can move his or her phone around, while maintaining the augmented reality view of the field. To do this, we use Shi-Tomasi corner detection to locate interest points in conjunction with Pyramidal Lucas-Kanade to track those interest points. We assume that the base points move in the same direction. This comes from the following assumptions:

1. The user is sufficiently far from the baseball field. The field of view is not affected by zoom from the user extending their arms forwards or backwards.
2. The user does not perform extreme rotations of the mobile device. We are currently only accounting for translational movement in the X and Y direction of the camera's view. Large rotations will cause the tracking to be incorrect.
3. The majority of the motion from the camera's perspective comes from the physical movement of the mobile device (as opposed to motion from the player's on the field, surrounding baseball fans, etc.).

4.1 Determining Optical Flow

After completing calibration, we save one frame as our *Base Frame*. For each frame afterwards, we always attempt to track 50 key points collected from the *Base Frame*. However, if we are unable to find a minimum of 5 key points from the *Base Frame*, we then attempt to track key points from the previous frame that was captured.

Here is the algorithm for calculating the *Mode Optical Flow:*

1. Track location of key points from the *Base Frame* using Pyramidal Lucas-Kanade.
2. Collect a histogram of the counts of vectors for each point that is found. The vector values are rounded to integers for binning.
3. If the vector with the highest count has a count of at least 5, return the vector as the *Mode Optical Flow* (Figure 7.1).
4. If the highest count is less than 5, track the location of key points from the previous frame (Figure 7.2).
5. Return the vector with the highest count as the *Mode Optical Flow.*

Using the *Mode Optical Flow*, we move the location of the base points accordingly and update the augmented view on the mobile device.

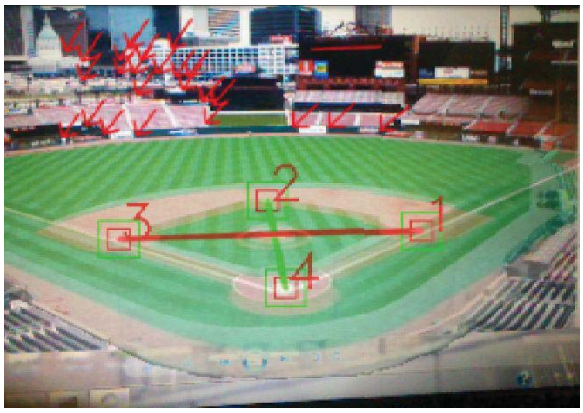

Figure 7.1 - Tracking from Base Frame.

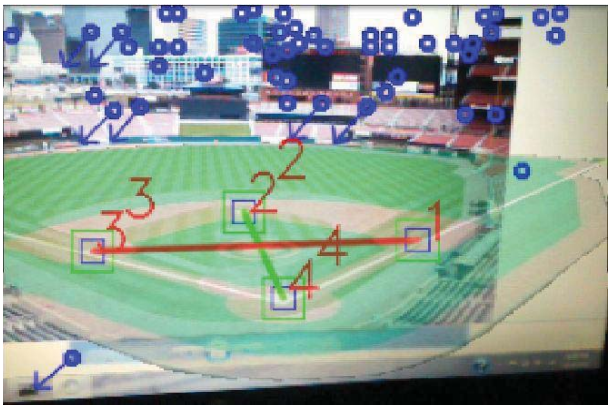

Figure 7.2 - Tracking from previous frame.

5. Statistical Visualization

Once we are able to locate and track the four bases, we overlay the real world view with visual baseball statistics. We continuously overlay an overhead baseball image onto the real world view, while giving the user the option to display other statistics. Our application displays three different statistics, currently containing hard coded data:

1. The spray chart of the current batter
2. The roster of the defending team on the field
3. The trajectory of a home run

Although we currently only display hard coded data, it would be possible to collect live game data from sites such as MLB.com or ESPN.com.

5.1 Spray Chart

The spray chart shows the positions on a field where a batter has hit successfully or gotten an out in recent games. This allows fans to assess where batters should be aiming and where fielders should be positioned. As our example data, we exhibit Albert Pujols' spray chart data from the 2011 playoffs at

Busch Stadium. Using the homography from the calibration, we draw green circles onto the real world view to signify a successful hit, while red circles mark outs.

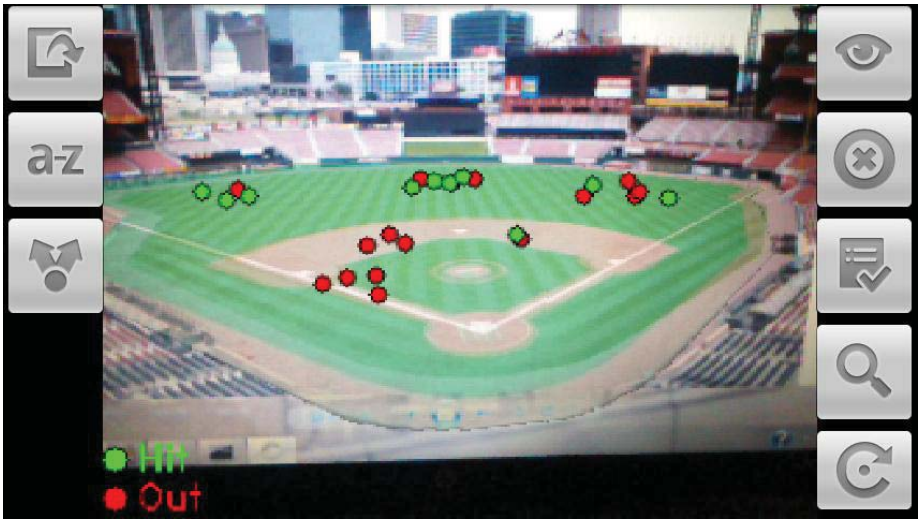

Figure 8 - Spray chart showing hits (green circles) and outs (red circles) for previous batter's at-bats.

5.2 Team Roster

At live baseball games it is often difficult to determine the identity of the players on the field, due to changes throughout the game and distance from the field. Our application provides a way to easily view the names of the players currently playing on the field. We select the locations of the nine positions in our baseball field overview image and map those locations to the real world view. We then write the names of the players at their respective locations, allowing the user to instantly know the roster of the team currently positioned on the field.

Figure 9 - Roster being displayed on real world view of baseball field.

5.3 Home Run Trajectory

We also show the trajectory of a hypothetical home run hit during a game. In our example, we select an endpoint, marking where a home run would land. Using the phones pitch and roll sensors, we map a parabolic path from home plate to the end point of the home run. The parameters of the parabola are predetermined. In actuality, we would not be able to solve for the exact parameters of the parabola.

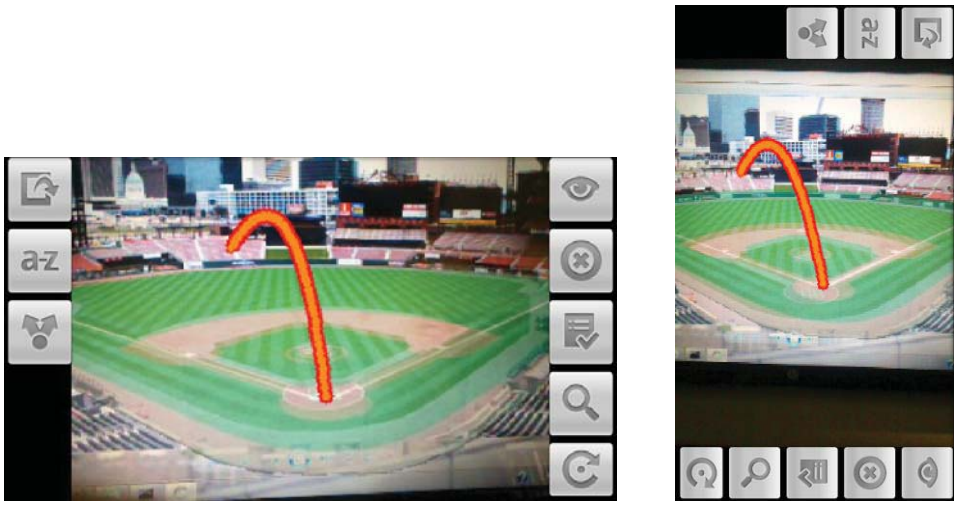

Figure 10 - Showing a home run trajectory in both landscape (left) and portrait mode (right).

6. Real-Time Application / Discussion

Programming such a computationally expensive mobile application proves to be challenging. Many of the algorithms in our application require looping over every pixel for every frame. Scaling down the images before running certain parts of the algorithm provides a noticeably significant increase in speed, while sacrificing only small amounts of accuracy in the homography calibration and tracking. Also, when the application runs in real time, it helps the user position the camera for the calibration process, providing a better final homography mapping. When the application takes several seconds to register each frame, it is difficult to steady the camera at all, which in turn affects the accuracy of the calibration.

During the calibration optimization process, we only run 35 steps of the Nelder-Mead minimization at a time. This helps show updates of the optimization to the user, letting them know the application has not stalled. This also allows the user to easily choose when the optimization is done, or realize that the process should be restarted if it is getting stuck in an unfavorable local minima.

Conclusion

We have shown that it is possible to create a real time augmented reality application for baseball using computer vision. Our application models a small subset of the statistics that are possible to display. Future developers can integrate new statistics or entirely new applications by using our homography optimization and tracking system.