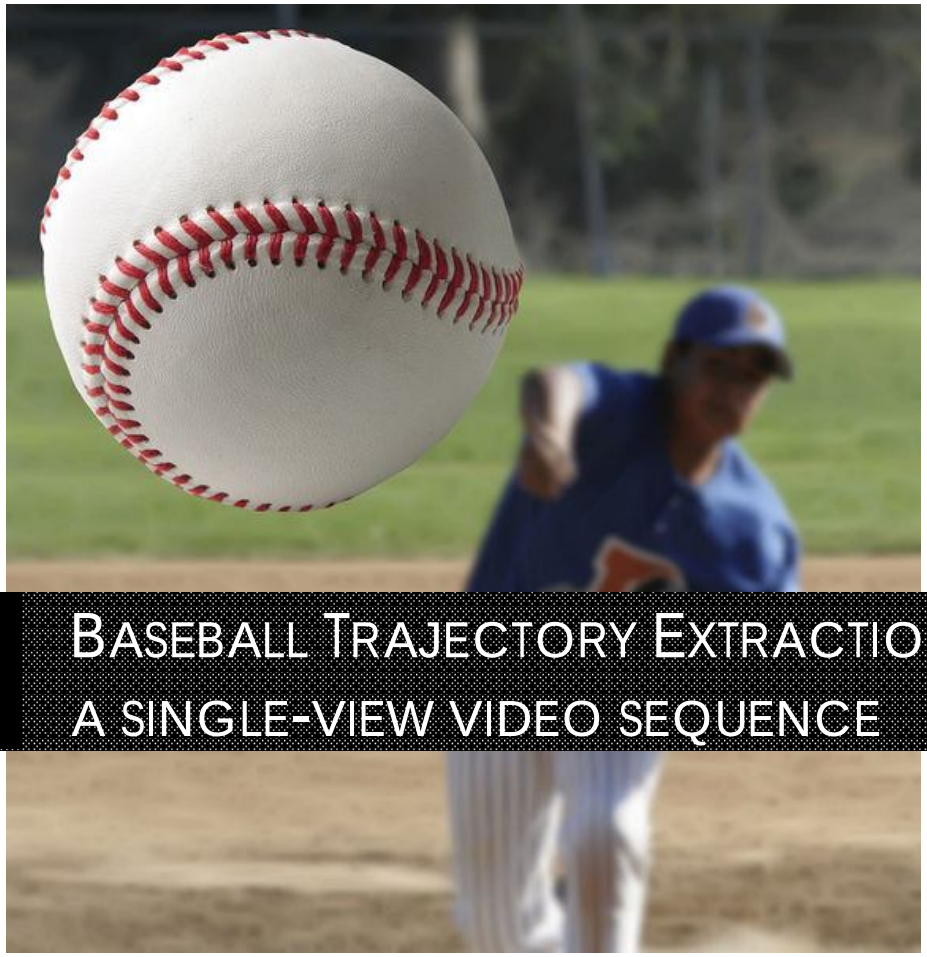
CS670 Final Project



CS4670

Baseball Trajectory Extraction from a single-view video sequence

Team members:
Ali Goheer (mag97) | Irene Liew (isl23)

Introduction

In this project we created a mobile application for Nokia N900 that can track the trajectory of a baseball in a single-view baseball video sequence. This trajectory is projected back onto the input video to further highlight ball's trajectory.

System Overview

Our program works in 4 distinct steps:

1. Ball Candidate Detection: detect all objects in the video that might be the ball
2. Track Ball Candidates: track the motion of all the ball candidates across the entire video, generating a trajectory path for each ball candidate
3. Trajectory evaluation: Pick the path that best fits the motion of a thrown ball from all the detected trajectories
4. Trajectory Projection: Extract the velocity and acceleration parameters of this "best path" and use those parameters to calculate the position of the ball for every frame

Figure 1 illustrates a high level overview of our program.

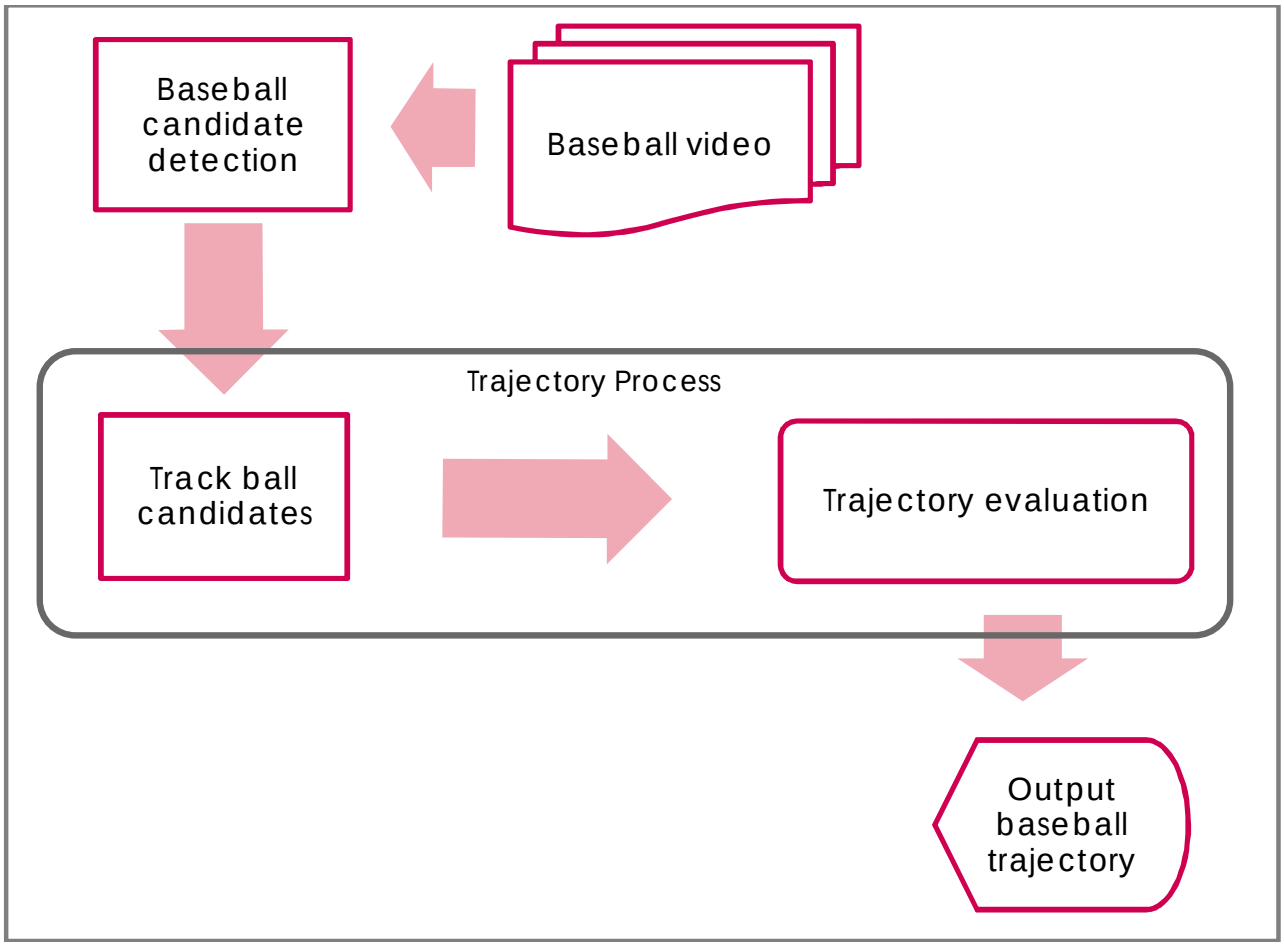


Figure 1: Program flowchart

Here are more details on these individual processing stages:

Ball Candidate Detection

We first smooth each frame using cvSmooth to remove noise and minimize the effect of light intensity variation. We convert the image into a gray-scale image and filter out the corners of the images, this minimizes the effect of any peripheral objects without affecting our baseball motion analysis as the ball will always be near the middle of the frame (we feel that this is not an unreasonable constraint to put on the input video).

We then subtract each frame from the previous frame to remove all stationary or background objects: this leaves us with a video that only has moving objects in it, which in our video will be the hitter, pitcher, umpire and the ball.

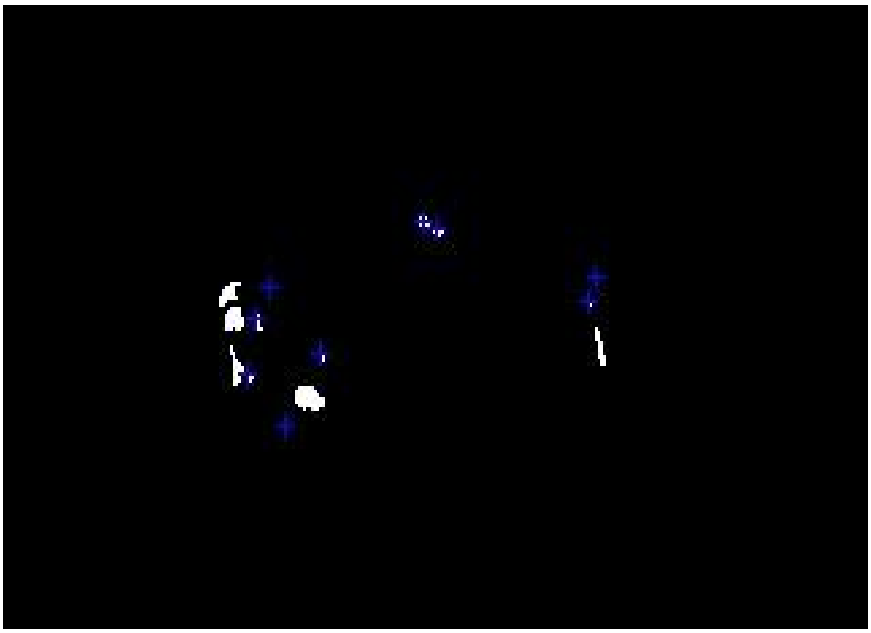
Here are a couple of frames from one of our input videos at this stage in our program:

As expected, all the background and non-moving objects have been subtracted out of the video frame. We only see the pitcher, hitter and the ball.

Here is another frame from the same input video: in this frame even the hitter has been subtracted out, probably because he was stationary at this particular instant and we only detect the pitcher and the baseball.

At this point, we apply blob detection (using openCV's cvBlob library) to chain all the connected pixels together into blobs, and we finally filter by size to remove the large blobs. Note that the size limits we use for this are very subjective and will depend upon the video. We set it to filter out all blobs having an area outside of 1 to 80 pixels.

Here are the same two video frames after blob detection; the blue crosses identify the location of the blobs that were detected:



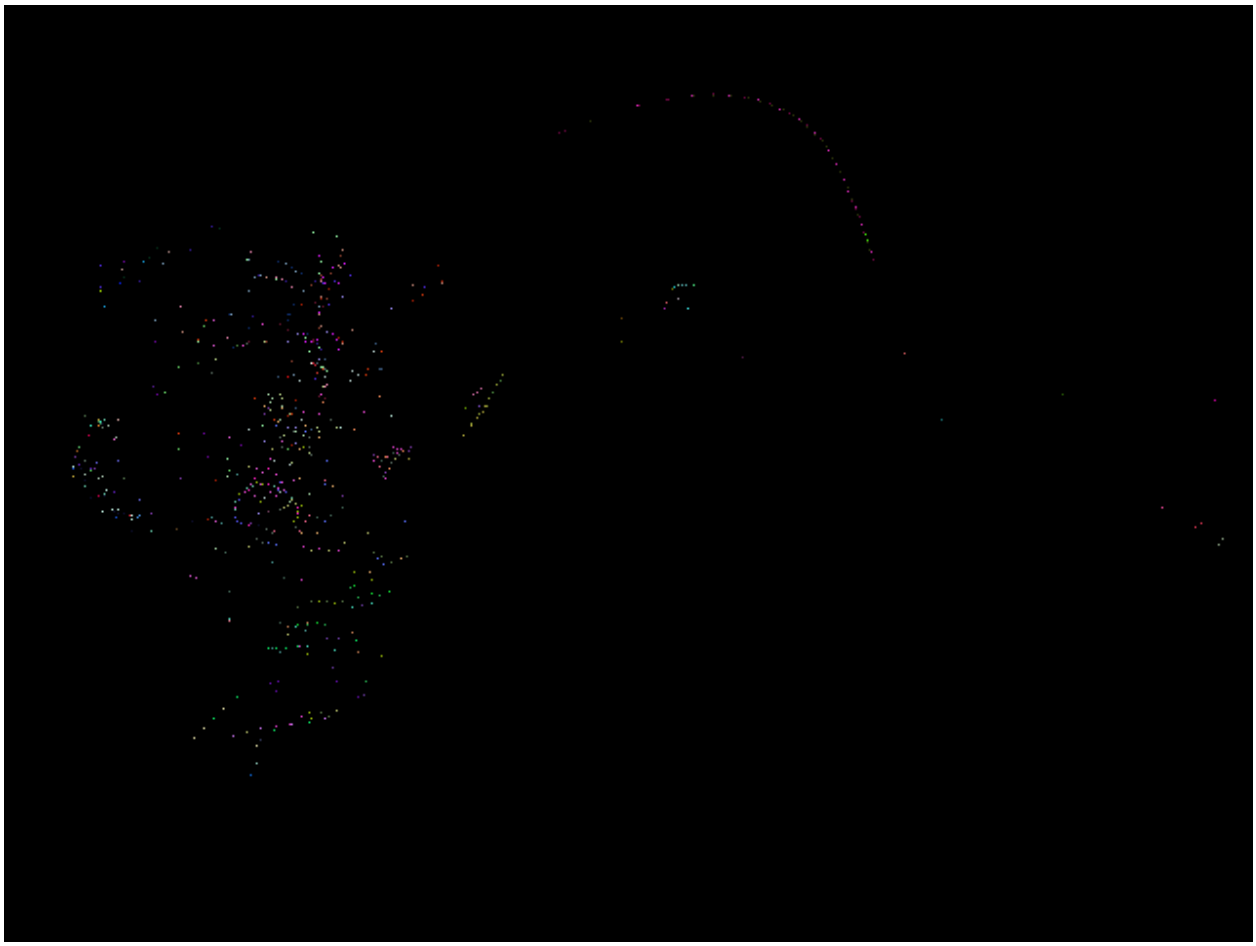
Trajectory Tracking

At this stage, we track the motion of these detected blobs through the video. We came up with a relatively simple (but eventually very effective) algorithm to do that: we go through every blob in every frame, and try to match it to the blobs in the previous frame. We do this matching both by size and the distance between the two blobs (we assigned 25% weight to the size difference between the two and 75% weight to the distance between the two). If this matching measure is within a specific threshold, we assume that it's the same blob that has moved since last frame and record it as such.

Otherwise if we found no match for the blob within our threshold, we look through the recent *'n'* frames and try to match it to the blobs in those previous frames, again within a threshold. If we find a match, we assume that it's the same blob, which for occlusion or some other reason didn't appear in the in-between frames but is detected in this current frame once again.

However, on the other hand, if we have still not found a match, we assume that this blob in the current frame is a new blob that did not appear in the previous frames, so we record it as a new frame and start tracking its motion as well.

At the end of this stage of our program, we have a data structure that contains the trajectories of all distinct blobs detected in the video. In the following image, we have plotted the motion of all the blobs in one of our videos: in this image each tracked trajectory is given a different color



Note that all the ball positions detected have the same color, i.e. the ball's motion throughout the entire video was correctly identified as the motion of one blob.

Best Trajectory Selection

Now our program has to pick the ball's trajectories from all the other trajectories that were detected. We apply 2D Kinematics to achieve that.

Note that the ball in our video will be the only object that will follow be moving through the air under an initially applied acceleration and velocity and thus will assume the physics equations that govern motion of such objects (Kinematics):

- $Sx = Vi_x * t + 0.5 * a_x * t^2$ (motion along x axis of the video)
- $Sy = Vi_y * t + 0.5 * a_y * t^2$ (motion along y axis of the video)

All the other objects (hitter's body, the bat, pitcher's body) will be moving under a constantly applied force and thus will not fit these equations. So, we fit all the detected trajectories to these equations using cvSolve and matrices, and then find the sum-square-error (SSE) of this fitting across the entire trajectory. We pick the trajectory with the lowest SSE. Note that relatively stationary objects (such as pitcher's feet) will also end up fitting these equations surprisingly well, so to correct for that, we also further impose the condition that the best-path actually includes moving objects (i.e. velocity and acceleration of that blob is greater than a threshold.

After this stage, our program correctly outputs the best trajectory, as shown in the image below

Re-projection of Trajectory onto Original Video

In this final stage, we project red-circles onto the original video's frames to identify the ball's position being detected (see attached video clips). We also use the baseball's velocity and acceleration parameters to "guess" the trajectory of the ball. We plot these positions using blue circles.

Experimental Results

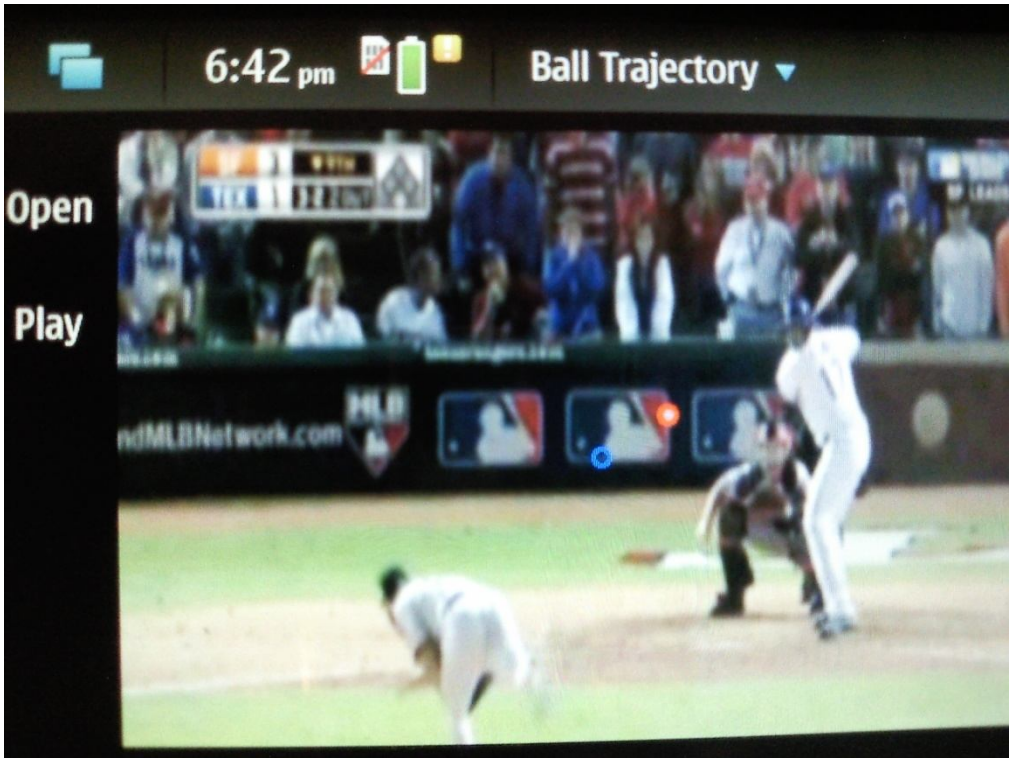
We tried our program on numerous MLB videos that we found online. We also shot numerous videos from N900 and processed them through the video. Our program successfully tracked the ball in almost every video, although sometimes it failed to pick the ball's trajectory as the best candidate because as described earlier, relatively static objects will also fit our physics equations quite well.

Also, when we use the extracted acceleration and velocity parameters to guess the trajectory of the ball (blue circles), our guessed trajectory was quite different from the originally detected trajectory. However, we feel that if we had the time to apply more accurate physics equations (involving drag coefficients etc) and if we had time to apply Kalman Filtering to remove outlier data points (see the 'future work' section below), our results would have been more accurate for the 'guessed' trajectory.

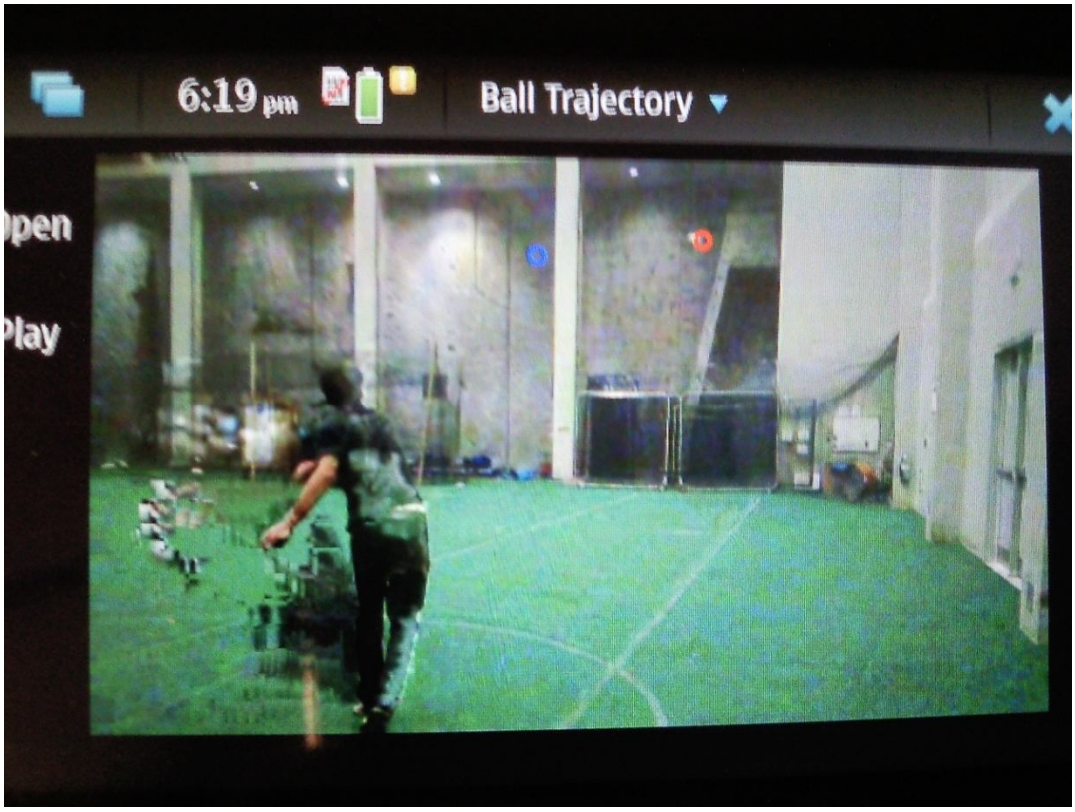
We recorded the sample output of two of our test videos, these videos have the red and blue circles drawn by our program that point out the positions of the ball detected, and the guessed positions of the ball respectively. These videos are included in the report zip file and they are:

- baseball_output1.avi - it shows our program's output after analyzing a clip from an MLB game
- baseball_output2.avi – it shows our program's output after analyzing a clip we shot ourselves

We are also including some screenshots of the program running successfully on Nokia N900:



This is a screenshot of our program analyzing MLB game clip.



This is a screenshot of the our program analyzing a clip we shot ourselves using Nokia N900

Limitations

The only time when our code actually failed was when there was significant pan and zoom in the video (e.g. the camera zooming in to capture pitcher's facial expression right after the baseball pitch). This would introduce too many new moving blobs that would overwhelm our algorithm. But we feel that it would be unreasonable to ask our program to handle such situations.

On the other hand, changes in light intensity and background noise did not affect our program's output.

Future Work

In the future, this code could be optimized so that it's fast enough to run in real time. We in fact tried to do that but ran out of time. We would also have liked to spend more time on following tasks:

- The ball isn't always correctly detected in all frames, so we have gaps in our baseball trajectory, we would have liked to fill in those gaps by applying Kalman

filter to the trajectory s

- We also would have liked to add an additional feature to our program wherein the user could throw a few controlled pitches in the very beginning to help calibrate the program more accurately

- Our algorithm has obvious applications in many other sports outside of baseball, such as cricket, tennis etc. It would have been interesting to adapt our code to detect ball trajectories in these other sports as well.

Work Breakdown by Individual Team Members

Irene Liew: filtering the input video and then applying blob detection to identify the blobs in the video; create the Maemo framework and GUI for both the desktop and Nokia N900 program;
Ali Goheer: tracking the motion of these blobs, and then applying Kinematics to identify the ball's trajectory

References

We used the following research paper as our primary source of direction

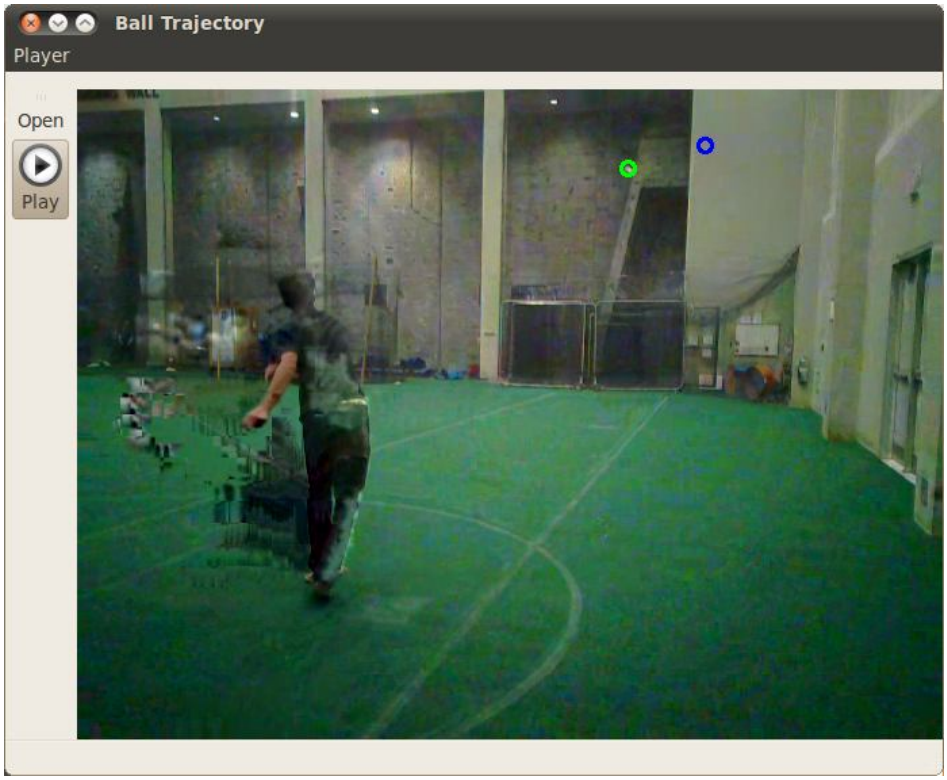
- Extraction of Baseball Trajectory and Physically-based Validation for Single-view Baseball Video Sequences
  http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4036974

And we also used the following open-source openCV library add-on for blob detection/extraction

- Blob detection library for OpenCV – cvblob
  http://code.google.com/p/cvblob/

Appendix

Here are a few sample screenshots of video s we used:



Here we used the “green” circle to replace the “red” circle which indicates the real detected trajectory. The “blue” circle is our predicted trajectory.