5-2017

Developing computer vision technology to automate pitch analysis in baseball.

Mahdi Moalla
University of Louisville

Recommended Citation

Moalla, Mahdi, "Developing computer vision technology to automate pitch analysis in baseball." (2017). *Electronic Theses and Dissertations.* Paper 2686.
https://doi.org/10.18297/etd/2686

DEVELOPING COMPUTER VISION TECHNOLOGY TO AUTOMATE PITCH ANALYSIS IN BASEBALL

By

Mahdi Moalla
B.E., Telecommunications Engineering, Higher School of Communications of Tunis, 2015

A Thesis
Submitted to the Faculty of the
J.B. Speed School of Engineering
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science in Computer Science

Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky

May 2017

DEVELOPING COMPUTER VISION TECHNOLOGY TO AUTOMATE PITCH ANALYSIS IN BASEBALL

By

Mahdi Moalla
B.E., Telecommunications Engineering, Higher School of Communications of Tunis, 2015

A Thesis Approved On

4/25/2017
Date

By the following Thesis Committee:

ACKNOWLEDGEMENTS

I would like to thank Dr. Frigui for his continuous support during my master thesis and my family for encouraging me during the whole master program. I would also like to thank Steve Stemle from Lokator Inc. for funding this research project.

ABSTRACT

DEVELOPING COMPUTER VISION TECHNOLOGY TO AUTOMATE PITCH ANALYSIS IN BASEBALL

Mahdi Moalla

April 25, 2017

Lokator is a baseball training system designed to document pitch location while teaching pitch command, selection and sequencing. It is composed of a pitching target and a smartphone app. The target is divided into a set of zones to identify the pitch location. The main limitation of the current system is its reliance on the user's feedback. After each throw, the pitcher or the coach needs to identify and report the target's zone that was hit by the ball by just relying on the naked eye. The purpose of this thesis is to investigate the possibility of using computer vision technology to automate the pitch analysis in baseball and improve the usability and accuracy of the Lokator system. Towards this goal, we have developed, implemented and tested a computer vision-based software system that adds the following contributions to the Lokator system:

1. Automated and accurate reading of the pitch location on the target.
2. Automated and accurate estimation of the velocity of the ball.
3. Provide contextual information about the pitch, such as vertical movement of the ball which can indicate late breaking.
4. Replace the target by a catcher and estimate the pitch location using a virtual target.

We have tested the software on a large set of recording. Those recordings are from indoor and outdoor environments with various illumination conditions and different backgrounds. The software was also tested on videos with softball pitches. To estimate the accuracy of the software, the sponsor gave us a set of 15 videos that include a total of 144 pitches along with the hit location of each pitch. Another set of 8 videos were provided to measure the accuracy of our software in terms of speed calculation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

Humans' brains extract visual knowledge from the surrounding world that their eyes perceive at an apparent ease. Visual knowledge includes discerning shapes, identifying objects and humans and predicting the emotions from facial appearances. The human visual system has been studied for decades to understand its inner-workings [1].

Computer vision is a subfield of computer science that imitates the brain visual role to gain high-level information and patterns from visual data such as images and videos [2]. It encompasses several subjects including object segmentation and recognition, 3D model reconstruction from 2D images and motion estimation. It aims to automate the tasks that are usually performed by the human visual system. Computer vision has been considered as a difficult field. Essentially, we have a reverse problem where we want to recover unknown information given insufficient input. For example, in the 3D model reconstruction problem, 2D input images lack the depth information on each point. Researchers have been incorporating physical and probabilistic models to generate valid solutions.

Over the past few years, computer vision methods has been developed for various problems. Sample tasks include:

- Optical character recognition (OCR): reading numbers and letters from receipts or number plates. The characters can be handwritten or printed [3].
- 3D model reconstruction: automated building of 3D models from images to describe monuments and objects. The reconstructed models can be used by robots to obtain a better overview of the surrounding world [4].
- Faces and objects recognition: These algorithms could be used to improve camera focus, improve face search relevancy in an image database or detect cars in self-driving vehicles [5,6].
- Visual authentication: This application involves using visual biometrics to login or verify the identity of people entering secured buildings or using a computer.

- Motion capture: use of cameras to capture the movement of objects or humans. This task can be used for different purposes such as monitoring areas for surveillance against intruders or crimes or explore road traffic.

Computer vision methods have been used in a variety of domains including defense, security, environment and sports. For instance, ball-based sports exhibit fast motion frequently [7]. This motion is hard for coaches and trainers to analyze and extract information from it. It is also difficult for the audience to observe the ball movement. Computer vision can be very useful for these cases.

Coaches and trainers have used computer vision methods to to analyze data to improve the performance of the athletes. For example, in the single-player sports, different techniques have been developed to monitor the movement of the athlete and commercial motion-capture systems are becoming popular. Typically, these systems involve optical markers fixed on the athlete's clothes and multiple calibrated cameras. Figure 1.1 shows an example of a motion capture system used for indoor ski. Markers are attached to the player clothes.

For some other sports, non-intrusive vision-based methods are required. For example, in tennis, the trajectory of the ball is important. However, it is not feasible to modify the ball. In this case, a simple camera can be used to track the ball.

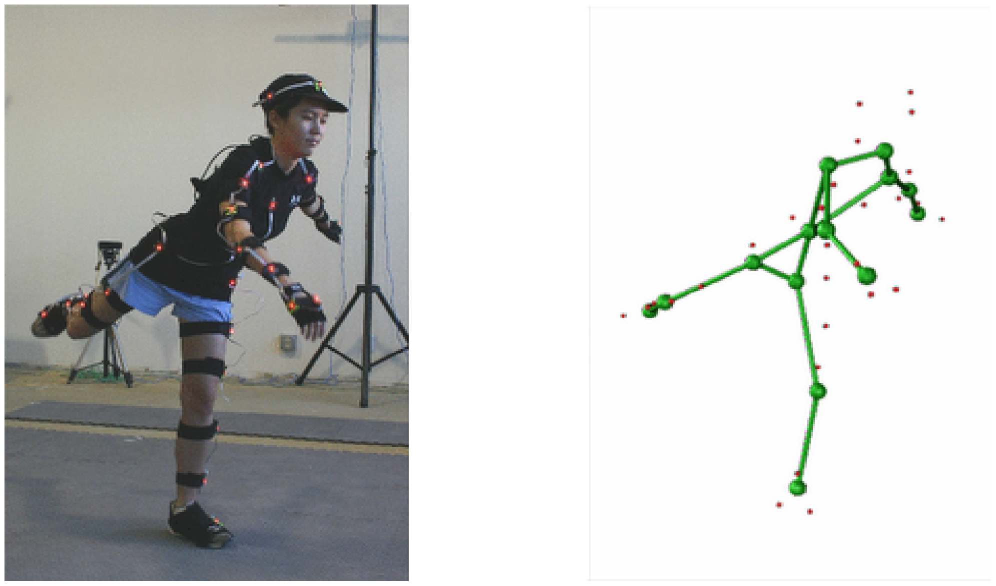

(a) Athlete with markers (b) Motion detected by markers

Figure 1.1: Motion capture system with optical markers and multiple cameras

Computer vision methods have also been used for team sports to improve the team's performance. First, multi-camera systems are used to record the movement of the team players. Then, a computer vision system can be used to extract and analyze information from the videos such as the position of the players and the formation of the team.

Computer vision has also been applied to analyze sports events in real-time. Solutions in this category help referee and audience to keep track of the players and provide a real-time coverage of the game events. Sports analysts can also use it to explain events on TV to the viewers. Football is a popular case where computer vision is used for the analysis of the players' movement and the events during the match and after its end. Figure 1.2 shows an example of football event analysis.

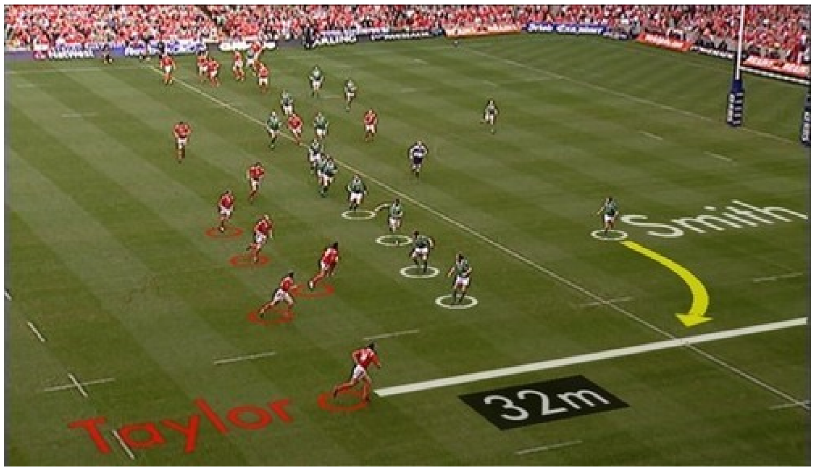


Figure 1.2: Computer vision used for the analysis of a match event in football.

In this thesis, we focus on using computer vision methods for analyzing ball motion in baseball and softball. Baseball involves high-speed motion of the ball that can be hard for a coach to analyze. The player that throws the ball is called the pitcher. Several computer vision solutions have been particularly developed for training in Baseball. For instance, Rapsodo [8] is a technology that gives a full description of a ball pitch metrics such as velocity, spin rate using high-speed cameras augmented with computer vision technology. Figure 1.3 shows the Rapsodo tracking device.

Figure 1.3: Rapsodo tracking device.

Rapsodo incorporates an additional software that displays a pitch trajectory and metrics on PC or iPad. However, the Rapsodo system is very expensive and can not be bought by every client. Another system is Athla [9]. It is a pure vision-based technology that estimates the speed of a baseball pitch. An iPhone is placed at a fixed location from the pitcher as illustrated in figure 1.4. The device starts capturing external motion using the camera. Once a baseball passes in front of it, the Athla software estimates its velocity. This application is cheaper than the previous system. However, it fails when the lighting condition is low. Moreover, it doesn't give more information about a pitch such as the trajectory.

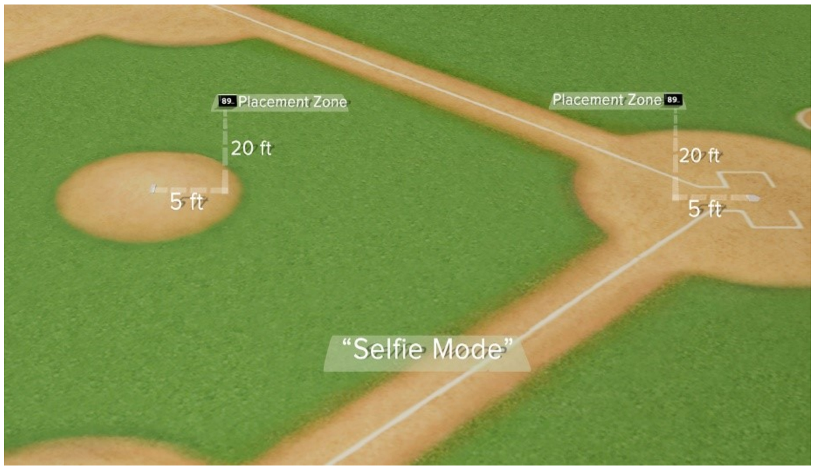
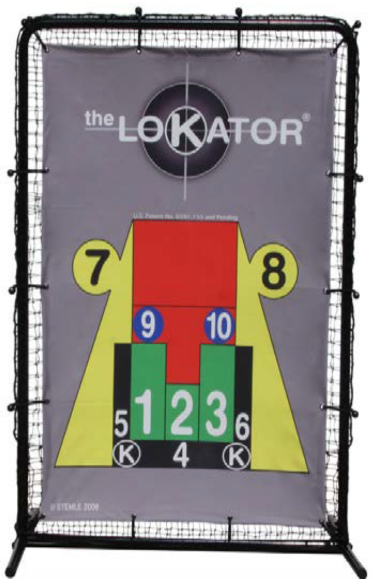


Figure 1.4: Athla iPhone placement example.

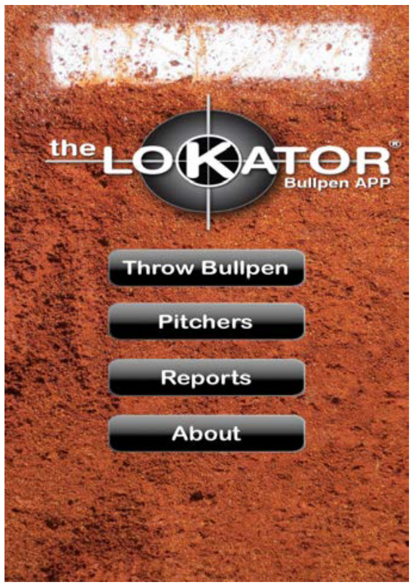
In this thesis, we describe the computer vision software that we have developed to extend the functionalities of the Lokator System. Lokator is a baseball and softball training system designed to document pitch location data while teaching pitch command, selection, and sequencing. The Lokator system is composed of a pitching target and a smartphone App and is illustrated in Figure 1.5.

As illustrated in Figure 1.5 (a), the target has 10 marked zones displayed with different colors. These zones are used to quantify pitch related data, such as strikes and other game related locations. During game training, the pitcher sets the target and uses the smartphone app to get instructions about which location on the target he needs to aim at. After the pitch, he documents the pitch location manually (see Figure 1.6). The collected data can then be used to compute a pitching score and recommend training sequences (bullpen).

The main limitation of the current Lokator system is its reliance on the judgment and manual input of pitchers. In fact, after throwing each bullpen, the pitcher (or coach) needs to assess and report which of the 10 zones within the target was hit by the ball just relying on the naked



(a) Pitching Target



(b) The Lokator Bullpen App

Figure 1.5: Pitching target and current App of the Lokator System

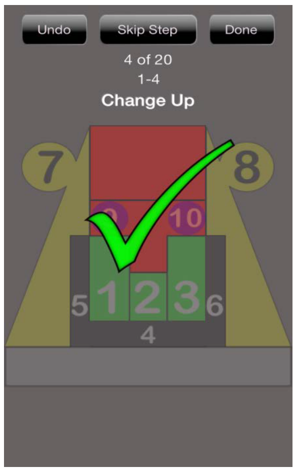


Figure 1.6: Pitch location documentation using the Lokator Bullpen app

eye. Such assessment can have low accuracy, given that a professional baseball pitcher can throw the ball at velocities that can exceed 90 mph. Thus, the current Lokator system could not be used effectively and efficiently to document pitch data.

Under this project, instead of relying on the naked eye to decide and manually enter pitch location, we developed and adapted computer vision methods to automate this task and improve its accuracy. The main contribution of this thesis is the development of computer vision technology for the Lokator system to achieve the following goals:

1. Accurate reading of the pitch location on the target.

2. Accurate estimation of the velocity of the ball.

3. Provide contextual information about the pitch, such as vertical movement of the ball which can indicate late breaking.

4. Explore the possibility of the replacing the target by a catcher while keeping the validity of the developed algorithms.

The remaining of the thesis is organized as follows. Chapter 2 gives an overview of the different techniques used in our software system. Chapter 3 details the developed software algorithms. Chapter 4 describes the experimental results. Finally, chapter 5 provides conclusions and future work.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we present background material that is relevant to the algorithms used in this research project.

2.1 Background subtraction

Background subtraction is a common task in computer vision. It is used to capture moving objects in videos. Several methods have been developed to perform background subtraction. In the following subsections, we outline three of the most commonly used approaches.

2.1.1 Background subtraction based on frame differencing

Frame differencing is one of the simplest methods for background subtraction. It considers the first frame as the background (B). Then, for all subsequent frames, an image pixel $I(x, y)$ is considered to be part of the foreground if

$$|I(x, y) - B(x, y)| > T, \tag{2.1}$$

where T is a given threshold. Figure 2.1 shows an example of running background subtraction based on frame differencing

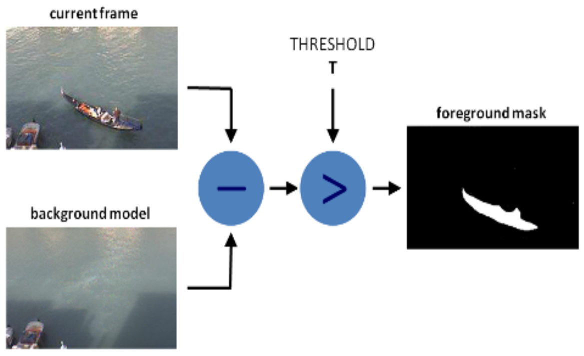


Figure 2.1: Background subtraction based on frame differencing

2.1.2 Background subtraction based on mixture of gaussians

In this method, the probability of each pixel's value is modeled as a weighted sum of adaptive Gaussian components. The Gaussian components are estimated from the pixel's values from previous frames. If a pixel is generated from a single surface using a static lighting, it can be modeled using a single static Gaussian. If we have dynamic lighting, each pixel can be modeled by a single adaptive Gaussian. On the other hand, under a more realistic scenario of dynamic lighting with different surfaces, multiple adaptive Gaussian components are needed [10].

Using a mixture of K gaussian components, the probability of a pixel's value can be modeled using:

$$P(X_t) = \sum_{i=1}^{K} w_{i,t}\eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{2.2}$$

The value of K depends on the available memory and computing resources. In (2.2), η is the Gaussian probability density function:

$$\eta(X, \mu, \Sigma) = \frac{1}{((2\pi)^{n/2}|\Sigma|^{1/2})} e^{-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)} \tag{2.3}$$

To avoid costly matrix inversion, the Gaussian covariance matrices are assumed to be diagonal, i.e.,

$$\Sigma_i = \sigma^2 I, \text{ for } i = 1..K. \tag{2.4}$$

The i^{th} Gaussian is considered to be matching the current pixel's value if the distance between them is within 2.5 standard deviation, i.e.,

$$|X_t - \mu_i| <= 2.5\sigma_i \tag{2.5}$$

After processing each frame, every pixel's Gaussians are updated using the following two steps:

- Step 1: Update the Gaussian's weight using:

$$\omega_t = (1-\alpha)\omega_{t-1} + \alpha M_{k,t} \tag{2.6}$$

  where α is the learning rate and $M_{k,t}$ is 1 for the matching Gaussian and 0 for the remaining components. In (2.6), α is a constant. Higher values of α implies that the weight of the matching Gaussian will be dominant in the mixture after few frames. Lower values of α keeps considerable weights for the non-matching Gaussians after processing a higher number of frames.

- Step 2: Update the pixel' matching Gaussian component using:

$$\mu_t = (1-\rho)\mu_{t-1} + \rho X_t \tag{2.7}$$

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t) \tag{2.8}$$

  where $\rho = \alpha\eta(X_t, \mu_{t-1}, \sigma_{t-1})$.

The K Gaussian components of each pixel are divided into foreground and background distributions. A pixel is considered as foreground if its matching Gaussian is foreground. To decide between the types of the Gaussians, we use the following steps:

1. Order the K Gaussians in a descending order of w/σ
2. The first B Gaussians are considered as background where:

$$B = arg\min_b(\sum_{k=1}^{b} w_k > T) \tag{2.9}$$

   In (2.9), T is a constant threshold.

The pseudo-code for background subtraction based on mixture of Gaussians is outlined below:

Algorithm 2.1 Background subtraction based on mixture of Gaussians

Inputs: F: video frame.
Outputs: Fg: Resulting foreground.

```
for each pixel $p$ in $F$ do
    Check for a matching Gaussian using (2.5)
    if a match is found then
        Update the matching Gaussian using (2.7) and (2.8)
    else
        remove the least probable Gaussian and insert a new one with the current pixel's value as
        a mean, with a high variance and a low weight. The new Gaussian will be the matching
        component for $p$.
    end if
    Update weights using (2.6)
    Classify $p$'s Gaussians as background or foreground using (2.9)
    Classify $p$ as either background or foreground
    if $p$ is classified as a foreground pixel then
        $Fg(p) = 1$ [Mark the pixel having the same location as $p$ in the resulting foreground]
    else
        $Fg(p) = 0$
    end if
end for
```

2.1.3 Background subtraction based on kernel density estimation

The Kernel Density Estimation (KDE) is based on a non-parametric estimation [11]. In this case, the probability of a pixel's value is defined as:

$$P(X_t) = \frac{1}{F}\sum_{i=1}^{F} K(X_t - X_{t-i}) \tag{2.10}$$

where $\{X_{t-i}, i = 1..F\}$ are the set of pixels' values from the previous F frames that are in the same location as X_t. In (2.10), K is a kernel function (usually Gaussian).

To account for small motion of the background, we maximize the probability of each pixel's value over a small neighborhood $N(X)$. Thus, instead of using (2.10) to estimate the probability of each pixel, we use

$$P_N(X) = \max_{y \in N(x)} P(X|D_y) \tag{2.11}$$

where $P(X|D_y)$ is calculated in the same way as in (2.10) except that it is using the pixels' values that are in the same location as y. A pixel is considered as background pixel if:

$$P_N(X) > T_1 \tag{2.12}$$

where T_1 is a constant threshold.

2.2 Kalman filter

Kalman filter is a common algorithm that has been used in various domains including control systems and signal processing. Kalman filter estimates the state of a system in case of missing or inaccurate measurements [12]. It can also predict the next state of a system. A system is modeled as a discrete-time process with the following stochastic difference equation:

$$x_k = Ax_{k-1} + Bu_k + w_k \tag{2.13}$$

where A is the state transition matrix, B is the control matrix, u_k is a control vector and w_k is the process noise. We also have the measurements:

$$z_k = Hx_k + v_k \tag{2.14}$$

where H is a matrix that transforms the state space to the measurement space and v_k is the measurement noise. Both w_k and v_k are assumed to have a Gaussian distribution with a zero mean, i.e.,

$$w \approx \eta(0, Q_k) \tag{2.15}$$

$$v \approx \eta(0, R_k) \tag{2.16}$$

Q_k is the covariance matrix of the process noise and R_k is the covariance matrix of the measurements noise. Kalman filter estimates the state space using two steps:

1. Prediction Operation: Predicts the current state of the system using the previous state.

   $$\widehat{x} = A\widehat{x}_{k-1} + Bu_k \tag{2.17}$$

   $$\widehat{P}_k = A\widehat{P}_{k-1}A^T + Q_k \tag{2.18}$$

   where P_k is the covariance matrix of the estimation error.

2. Correction operation: Uses the current measurement to correct the current predicted state

   $$G_k = P_k H^T (HP_k H^T + R_k)^{-1} \tag{2.19}$$

   $$\widehat{x}_k = \widehat{x}_k + G_k(z_k - H_k x_k) \tag{2.20}$$

   $$\widehat{P}_k = (I - G_k H_k)\widehat{P}_k \tag{2.21}$$

   The matrix G_k is called the Kalman gain.

2.3 Expectation-Maximization algorithm

Expectation-Maximization (EM) is an unsupervised learning algorithm. It estimates the data distribution as a mixture of Gaussians [13, 14]. The number of components K is an input parameter to the algorithm. Given a set of d-dimensional feature vectors $(x_1, ..., x_N)$ that are sampled from a mixture of Gaussians distribution, Their probability is expressed as:

$$P(x) = \sum_{i=1}^{K} w_i g_i(x), \tag{2.22}$$

$$g_i(x) = \eta(x, \mu_i, \Sigma_i) = \frac{1}{((2\pi)^{n/2}|\Sigma_i|^{1/2})} e^{-\frac{1}{2}(X-\mu_i)^T \Sigma_i^{-1}(X-\mu_i)} \tag{2.23}$$

where η is a Gaussian probability distribution with mean μ_i and covariance matrix Σ_i. In (2.22), w_i is the weight of component i and satisfies the constraint:

$$\sum_{i=1}^{K} w_i = 1 \tag{2.24}$$

The EM algorithm estimates the distributions' parameters iteratively. An iteration is composed of two interleaved steps:

- Expectation step (E): The likelihood that the k_{th} data sample is generated from the i_{th} mixture component is calculated using:

$$\alpha_{k,i} = \frac{w_i g_i(x_k)}{\sum_{j=1}^{K} w_j g_j(x_k)} \tag{2.25}$$

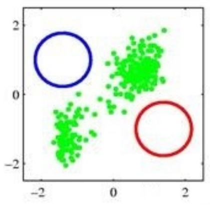
- Maximization step (M): The new distributions' parameters are calculated using the new likelihood coefficients using:

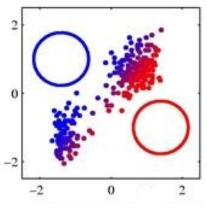
$$w_i = \frac{1}{N}\sum_{j=1}^{N}\alpha_{j,i} \tag{2.26}$$

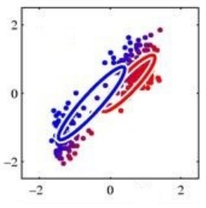
$$\mu_i = \frac{\sum_{j=1}^{N}\alpha_{j,i}x_j}{\sum_{j=1}^{N}\alpha_{j,i}} \tag{2.27}$$

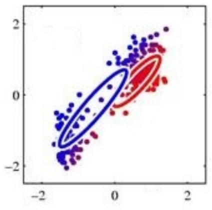
$$\Sigma_i = \frac{\sum_{j=1}^{N}\alpha_{j,i}(x_j-\mu_i)(x_j-\mu_i)^T}{\sum_{j=1}^{N}\alpha_{j,i}} \tag{2.28}$$

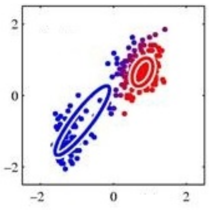
Figure 2.2 illustrates the result of executing EM on a sample 2-dimensional data. The green dots in figure 2.2(a) correspond to the data samples. The two initial components are shown in red and in blue. Figure 2.2(b) shows the result of running the E step for the first time. Each point has a probability of being generated from one of the two components which is depicted by its color. Figure 2.2(c) shows the results after running the M step where the two components are updated to match the new likelihood coefficients. Figure 2.2(d) and 2.2(e) show the results of E and M steps after the next iteration. Finally, figure 2.2(f) shows the results after convergence as it can be seen. The two components fit the input data.


(a)


(b)


(c)


(d)


(e)

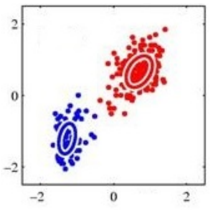

(f)

Figure 2.2: Illustration of the EM algorithm. (a) Original data and initial components. (b) Results after the first E step. (c) Results after the first M step. (d) Results after the second E step. (d) Results after the second M step. (f) final results after convergence.

2.4 Template matching

Template matching is a general purpose technique that is used to find a template image T within a source image I. The template image and the sub-image in the target location do not need to be pixel-wise equally. Template matching works by sliding the template image over the source image and calculating a score value for each location. The location that has the highest value will be selected. Figure 2.3 shows an example of template matching.

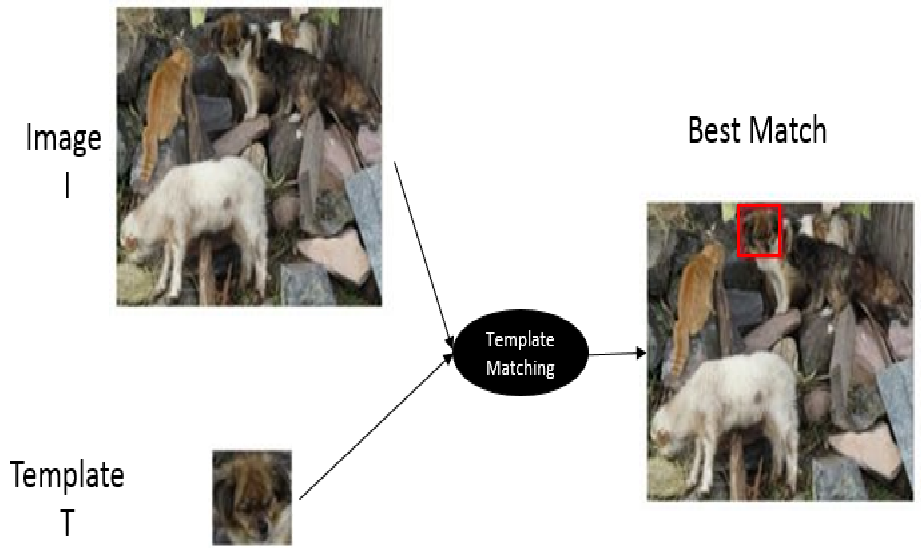


Figure 2.3: Template matching example

A simple score function is:

$$S(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2 \tag{2.29}$$

Another score function uses normalized cross-correlation coefficients [15]:

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - \overline{T})(I(x+x',y+y') - \overline{I})}{\sigma_T \sigma_I} \tag{2.30}$$

where $\overline{I}$ and $\overline{T}$ are the averages of the source and template images and σ_I and σ_T are their standard deviation.

CHAPTER 3

DEVELOPING COMPUTER VISION TECHNOLOGY TO AUTOMATE PITCH ANALYSIS IN BASEBALL

In this chapter, we will present the developed software. We will start with the overall structure of the software. After that, we will describe the target detection and zones recognition method. The next part deals with detecting and tracking the ball. The final part details the speed estimation method.

3.1 Structure of the developed software

The developed software is composed of 4 main parts. Figure 3.1 highlights the structure of the software. The input data are video files that contain recording of successive pitches. The first frame is used to locate the target using template matching and recognize the zones. After that, the software enters in a loop that has three parts. The first part detects the first appearance of the ball in the scene. The second part tracks the ball until it reaches the target. It also deals with the case when the ball bounces off the ground. The final part continues tracking the ball around the target. This component also takes into account the different possible scenarios. Once the iteration of the loop is completed, we clear its context and return to the first part. The software ends its executions once all the video frames are processed. To develop the software, we used C++ with the OpenCV library [16]. OpenCV is a software library. It is used to accelerate the development of computer vision and image processing applications while targeting production systems. It also includes machine learning primitives.

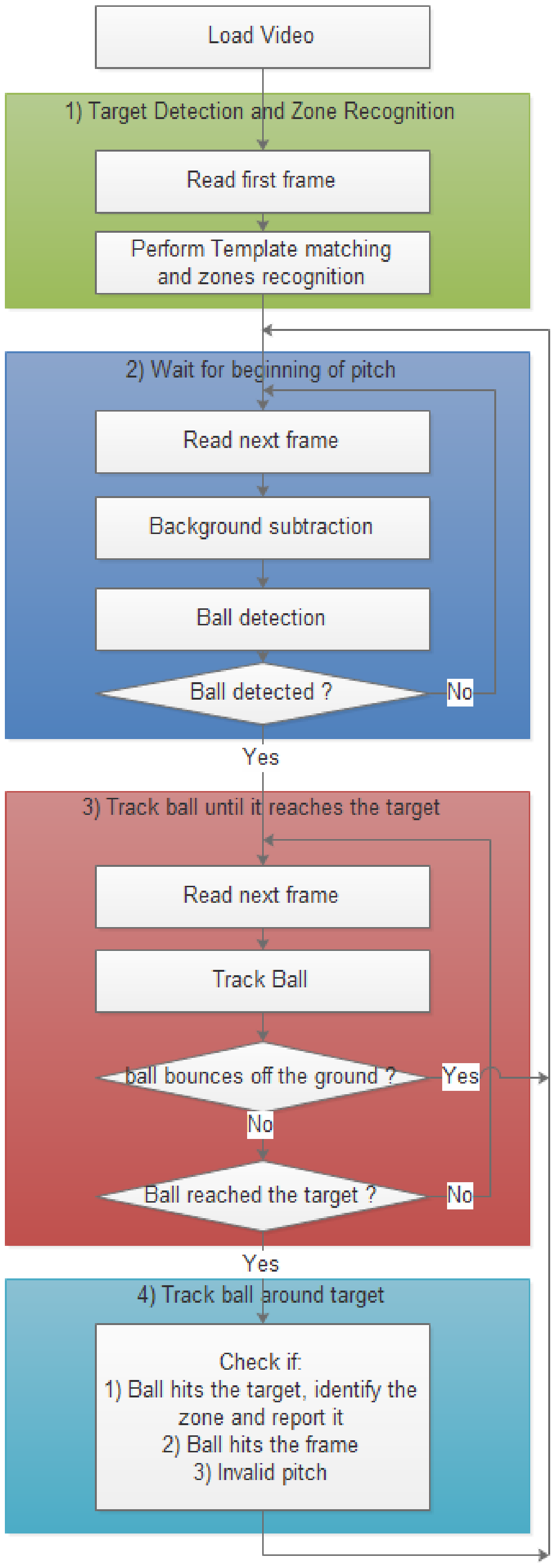

Load Video
1) Target Detection and Zone Recognition
Read first frame
Perform Template matching and zones recognition
2) Wait for beginning of pitch
Read next frame
Background subtraction
Ball detection
Ball detected ?
No
Yes
3) Track ball until it reaches the target
Read next frame
Track Ball
ball bounces off the ground ?
Yes
No
Ball reached the target ?
No
Yes
4) Track ball around target
Check if:
1) Ball hits the target, identify the zone and report it
2) Ball hits the frame
3) Invalid pitch

Figure 3.1: Flowchart of the developed software

3.2 Target detection

To locate the target, we use the template matching technique described in section 2.4. Using one of the video recordings, we manually locate the target and extract its image. Figure 3.2 shows the target template that we extracted and used for all for our experiments.

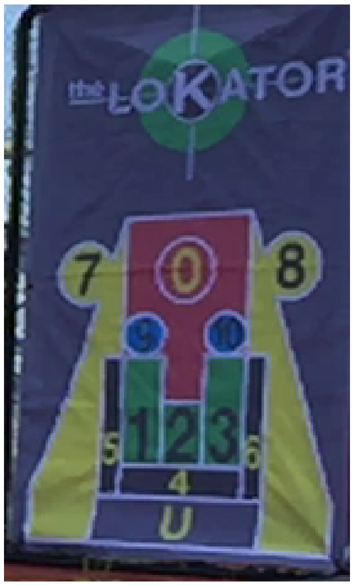
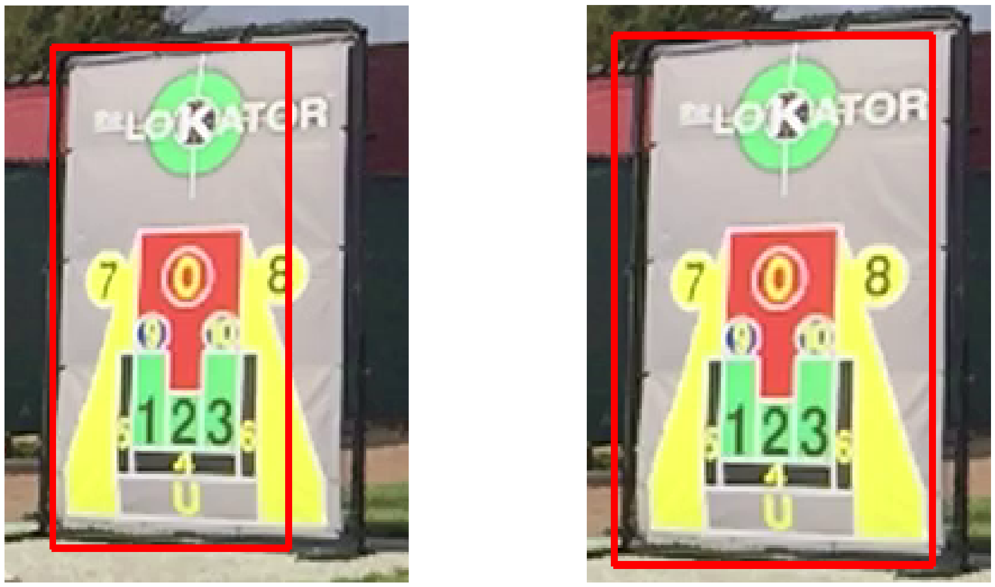


Figure 3.2: Image of the target template that we extracted from one od the video recordings

OpenCV provides several score functions for template matching including the normalized cross-correlation based function. We have compared few methods. We found that the normalized cross-correlation approach is the most effective and has the best results. However, the template matching technique has a limitation. If the distance from the camera to the target changes, the apparent target size will change and the algorithm may fail to detect the target. To overcome this limitation and make the matching scale invariant, we perform a multi-scale search for the best region that matches the template. Specifically, we scale independently the height and the width of the target by a factor ranging from 50% to 200% by an increment of 5% and we repeat the search for the best matching region. Figure 3.3 illustrates an example where single-scale template matching failed to correctly identify the target whereas multi-scale template matching succeeded in detecting it correctly.



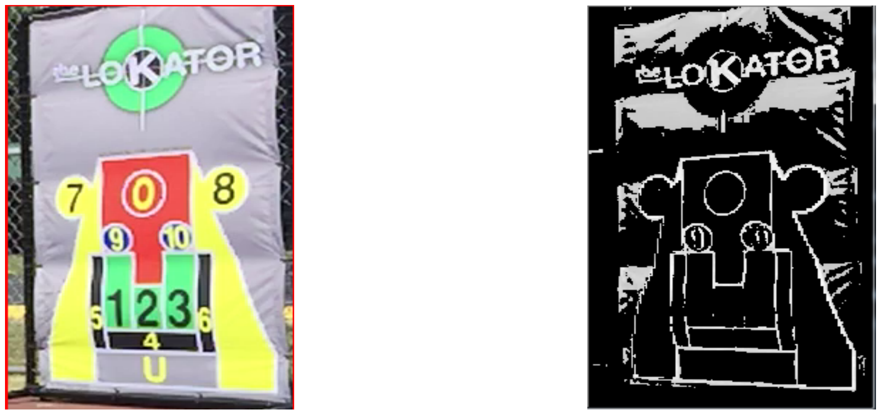
(a) (b)

Figure 3.3: Target detection example where multi-scale template matching solved the problem of height/width ratio mismatch. (a) Results of single-scale template matching. In this case, half of zone 8 will be missed. (b) Results of multi-scale template matching

3.3 Zones recognition

Zones recognition proved to be a hard task. Our approach has evolved as we encountered several challenges. Our initial approach attempted to explore the fact that zone boundaries are the brightest pixels (white) in the image. Thus, we simply threshold the image of the target to identify the brightest pixels, then identifying line segments in the thresholded image. Even though this approach can tolerate a small amount of noise and few disconnected edges, it can fail when the noise level increases drastically due to significant variation in illumination or target distortion. Target distortion can be caused by wrinkles or when the ball, traveling at high speed, gets colse to the target. Sample scenarios where our approach has failed to separate and label the different zones are illustrated in the following figures.

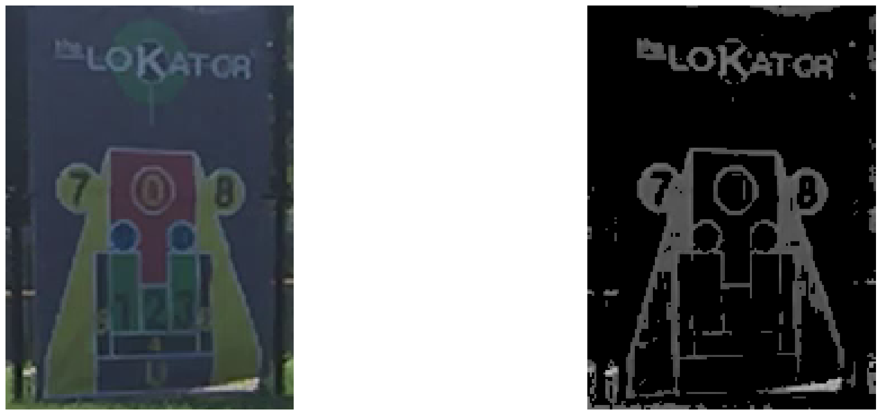
In Figure 3.4(a), we show a sample image where the pitching target appears to be too bright. The thresholded image is displayed in Figure 3.4(b). As it can be seen, even though too many bright pixels where selected, some edge pixels (e.g. separating zones 1 and 2 and 3) are missing. Decreasing the threshold will cause more non-edge pixels to be selected. On the other hand, increasing the threshold will cause more edge pixels to be missed.



(a) (b)

Figure 3.4: (a) Sample image of a target that appears to be too bright. (b) Resulting image after thresholding the one in (a).

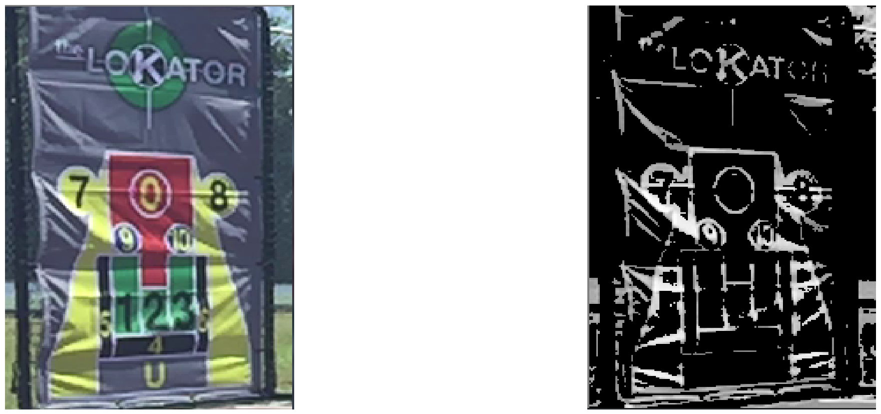
In Figure 3.5(a), we show a sample image from the other extreme condition where the pitching target appears to be too dark. The thresholded image is displayed in Figure 3.5(b). As it can be seen, even with a relatively low threshold (that generates too many non-edge pixels), many zone edges are missing. Consequently, the different zones cannot be identified for this sample.



(a) (b)

Figure 3.5: (a) Sample image of a target that appears to be too dark. (b) Resulting image after thresholding the one in (a).

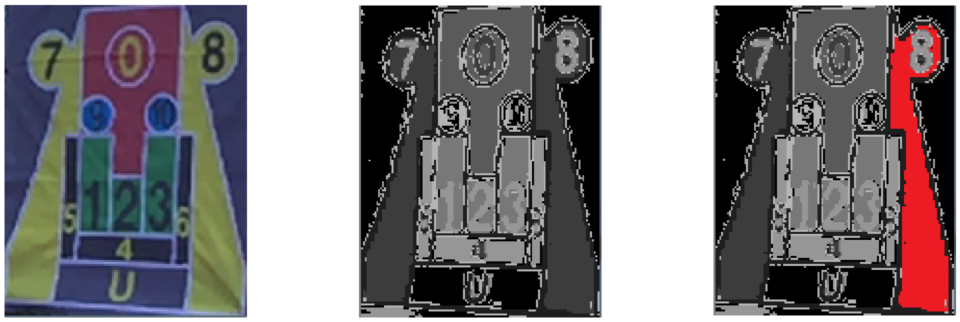
Another factor that seems to affect the reliability of the zone detection is the way the target is attached to the frame. If the target is not flat (i.e. has wrinkles), its image will be distorted and detecting the zones' boundaries becomes more challenging. Figure 3.6(a) displays a sample image of a wrinkled target. Figure 3.6(b) displays the thresholded image. As it can be seen, distortion in the target has created many strong edges making it impossible to isolate the zones' boundaries based on edges only.



(a) (b)

Figure 3.6: (a) Sample image of a target that is not attached properly to the frame. (b) Resulting image after thresholding the one in (a).

To address all of the above issues, we have designed, developed, and tested an alternative algorithm to detect and identify the different pitching target zones. Instead of thresholding the target image then identifying edges (i.e. bright edges), we use an unsupervised learning approach to cluster the different pixels of the target according to their color distributions. In particular, we use the Expectation-Maximization (EM) algorithm to cluster all pixels of the target based on their 3-dimensional colors (Red, Green, and Blue). Each cluster, characterized by an average color and a covariance matrix, will correspond to pixels that share similar colors. Since different zones can have the same color (e.g., both zones 7 and 8 are yellow), we need an additional step to split clusters that combine multiple zones into sub-clusters. We use the fact that pixels from different zones cannot belong to the same connected component since a white edge separates all zones and there are no white zones. Thus, after clustering, each cluster is analyzed to identify its connected components. Finally, we consider each connected component of neighboring pixels that belong to the same cluster to be part of the same target zone. Figure 3.7 uses a sample target image to illustrate the steps of our new and improved approach to detect the target zones.



(a) (b) (c)

Figure 3.7: Illustration of the EM-based approach to detect target zones. (a) Image of original target. (b) clusters obtained after applying the EM algorithm (we used 7 clusters). Notice that zones 7 and 8 were assigned to the same cluster. (c) Zone 8 (shown in red) was detected as one connected component.

So far, our approach ensures that regions from different zones cannot be combined into the same connected component. However, it is likely that some zone are partitioned into multiple connected components. Thus, a post-processing step is needed to merge all adjacent connected components that have similar colors. First, we create a model target that assigns a unique color and label to each region. This needs to be done only once and will change only if the target design or size changes (it may need to be adapted if the location of the camera with respect to the target changes significantly). Figure 3.8 displays the model target that we have created based on the current video collection.

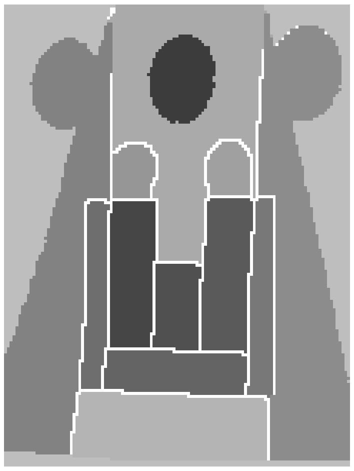


Figure 3.8: Model of the target zones

Each pixel from a connected component will be part of a zone in the model target. To label each connected component, we calculate the membership percent of its pixels to each region and we select the region that has the highest percentage. Figure 3.9 illustrates the labeling process where the blue color outlines the connected component and the red color outlines a model target region. In this case, the connected component will be mapped to the region 1.

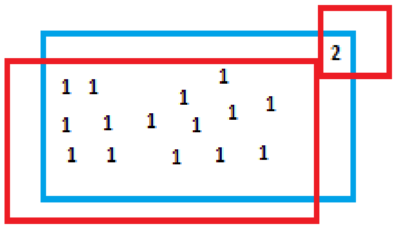
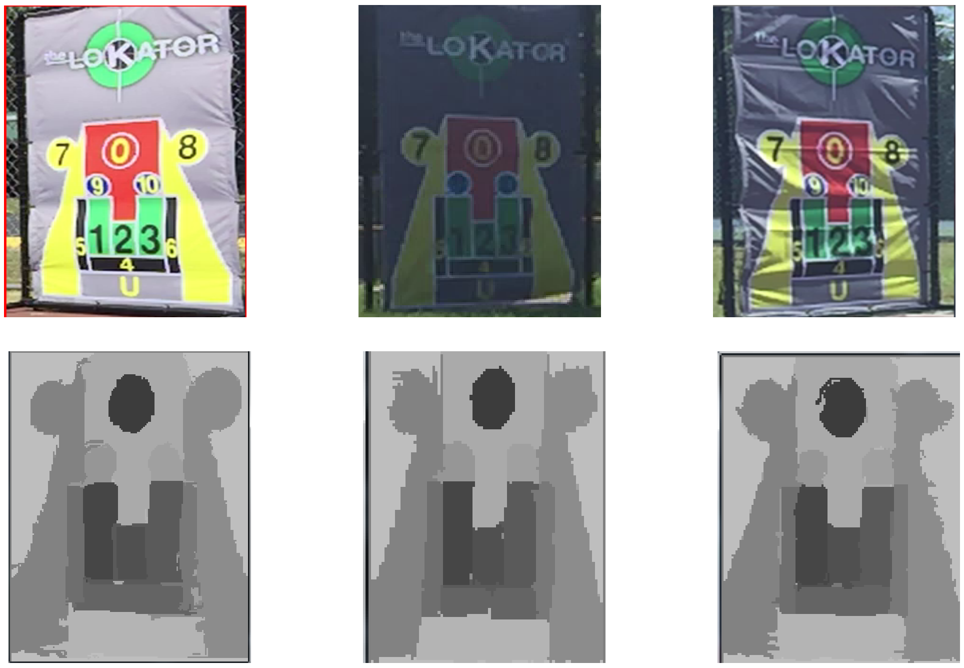


Figure 3.9: Illustration of our labeling process of the connected components. Blue outlines a connected component and red outlines model target regions. In this case, the connected component will be labeled as zone 1.

After extensive testing to optimize the parameters of the proposed zone detection algorithm, we found that our algorithm is more robust when a large number of clusters are used for the EM clustering component (currently we use 30 clusters). This will make the EM more sensitive to color variations. Thus, the identified clusters can distinguish between the different colors even when the illumination varies significantly. Moreover, the large number of clusters will cause some clusters to be dedicated to the white edges separating the zones. Thus, connected components are not likely to mix pixels from different zone. The downside of using a large number of clusters is that some regions may get partitioned into a large number of connected component. However, our mapping using the target model will assign the same zone number to all connected components. Figure 3.10 illustrates the detected zones for the 3 cases (shown in Figures 3.4-3.6) where the old approach has failed.



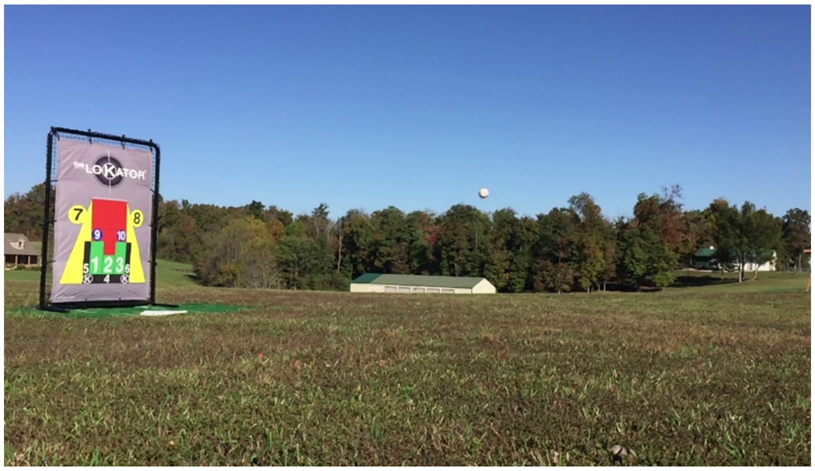
(a) (b) (c)

Figure 3.10: Robustness of the new zone detection approach to: (a) over illumination, (b) under illumination, and (c) target deformation. These are the 3 cases shown in figures 3.4-3.6 where the old approach fails.

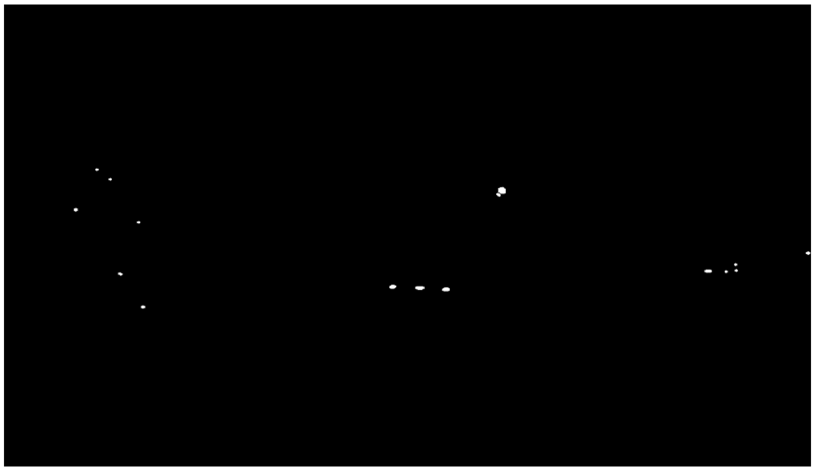
3.4 Ball detection

3.4.1 Background subtraction methods experiments

We have tested the different background subtraction methods that we have discussed in the second chapter. First, We have implemented and experimented with the frame differencing approach. We found out that this approach is very sensitive to small random motions (e.g. wind) and other discontinuities within the image frame. Figure 3.11 displays such an example. As it can be seen in Figure 3.11 (b), using a small threshold (T=32), results in many false positive examples (i.e. non-moving objects identified as foreground). On the other hand, as illustrated in Figure 3.11(c), using a large threshold (T=64), the moving ball was missed as no foreground pixels were identified. We have concluded that the frame differencing method is not robust and cannot be used in this application.



(a)

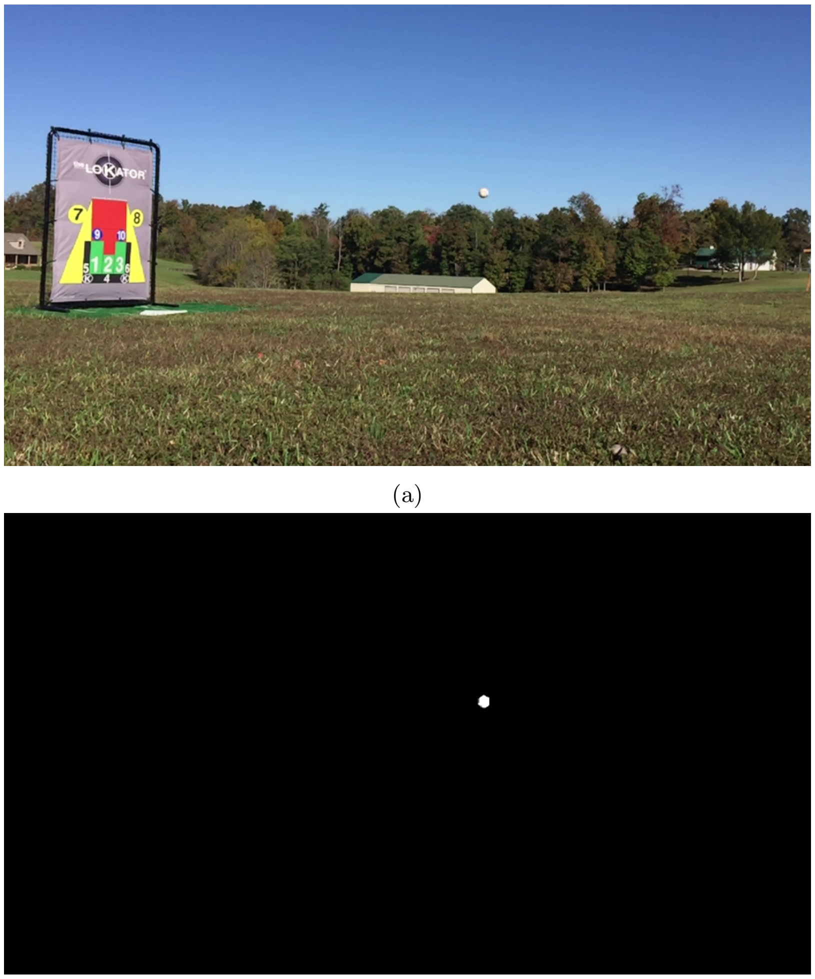


(b)

(c)

Figure 3.11: Sample results using frame differencing for ball detection. (a) sample frame from one of the video collection. (b) foreground objects detected using a small threshold ($T = 32$), and (c) no foreground pixels detected when $T = 64$

The method of the background subtraction based on Mixture of Gaussians (MoG) approach is implemented in OpenCV. We found out that this approach outperforms the frame differencing approach. It is reliable when the background is static. However, for dynamic background (e.g., moving camera, tree motion due to wind, etc.) the mixture of Gaussian approach results in many false positive. Figure 3.12 displays typical results of the background subtraction based on MoG for video with static background. As it can be seen in Figure 3.12(b), in this case, this approach can detect the moving ball correctly with no false positives.

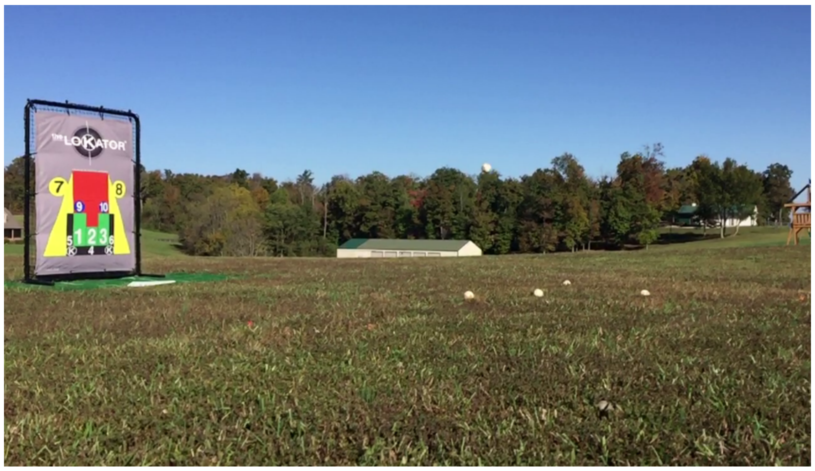


(a)

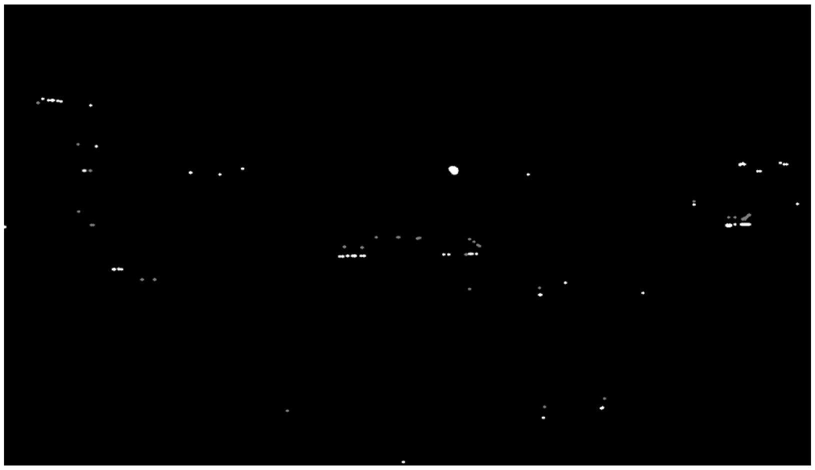
(b)

Figure 3.12: Sample results using Mixture of Gaussians for ball detection. (a) sample frame from one of the video collection with static background. (b) Detected foreground pixels correspond to the moving ball

Figure 3.13 displays typical results of the background subtraction based on MoG for video with dynamic background (moving camera). As it can be seen in Figure 3.13(b), in this case, this approach can detect the moving ball correctly but it has also identified several false positives.



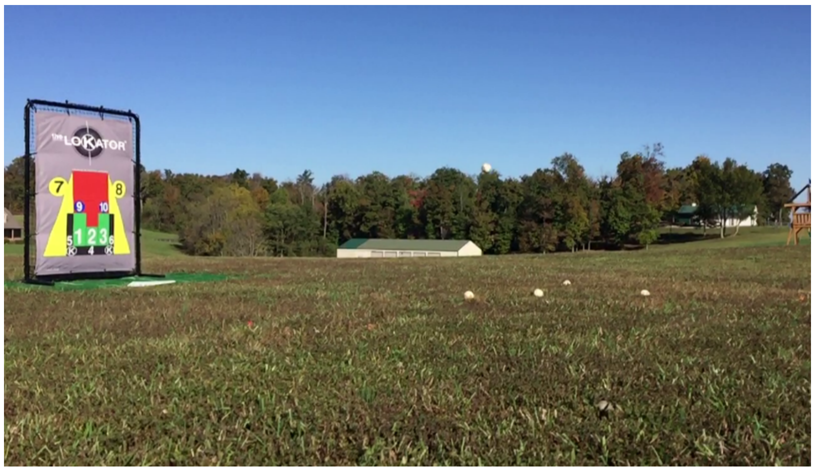
(a)



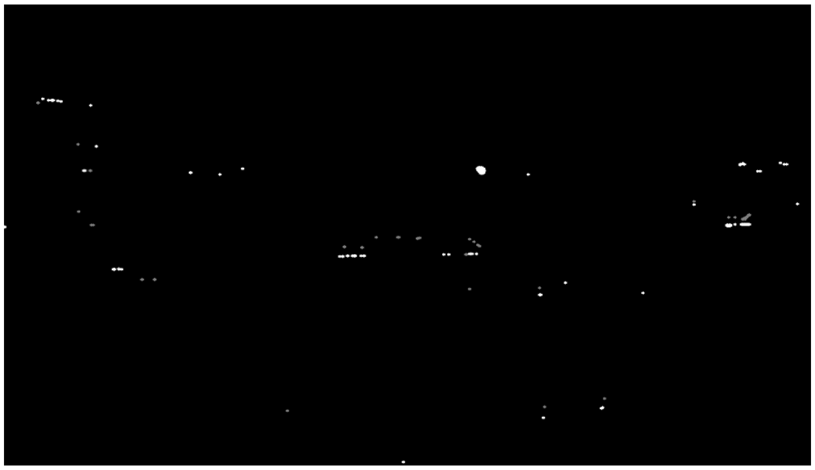
(b)

Figure 3.13: Sample results using Mixture of Gaussians for ball detection. (a) sample frame from one of the video collection with dynamic background (moving camera). (b) Detected foreground pixels correspond to the moving ball and many other false positives

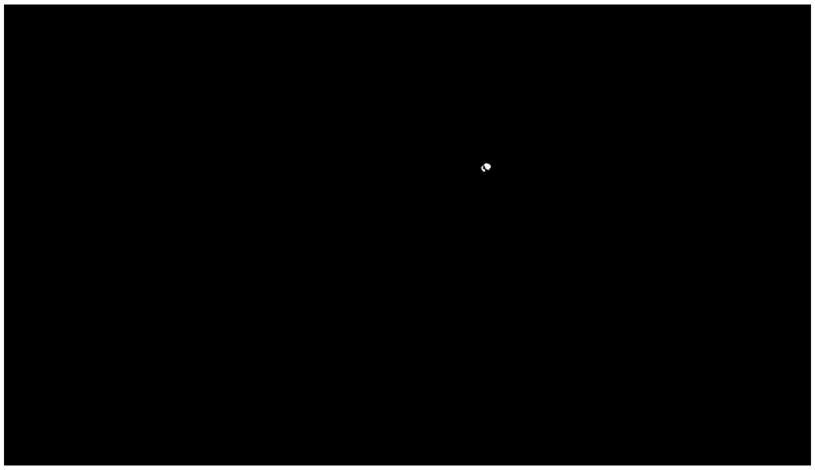
We have also implemented and tested the method of background subtraction that is based on kernel density estimation. In Figure 3.14, we compare the performance of the MoG and KDE approaches for the case of dynamic background. As it can be seen, KDE is more robust and outperforms the MoG approach. However, KDE approach is computationally expensive. It uses a considerable amount of time to process each frame. In our experiments, KDE took $400ms$ to process one frame where MoG takes only $3ms$.



(a)



(b)



(c)

Figure 3.14: Comparison of the MoG and KDE background subtraction approaches for video with dynamic background. (a) sample frame from one of the video collection with dynamic background (moving camera). (b) Detected foreground pixels using MoG correspond to the moving ball and many other false positives. (c) Detected foreground pixels using KDE correspond to the moving ball with no false alarms.

3.4.2 Foreground subtraction

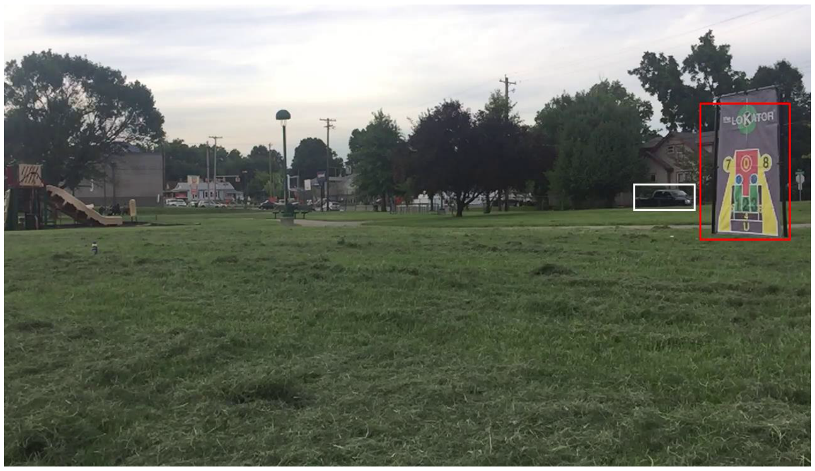
The problem of motion noise is present in different videos and it is caused by different causes including dynamic background (e.g., moving camera, tree motion due to wind) or moving entities such as cars or birds. We developed a new method that can handle the motion noise while keeping the frame processing time low.

Since we are using a high frame rate recording, objects moving at a speed that is much slower than the ball will appear to be static between two consecutive frames. This is in contrast to the faster moving ball that will exhibit a small position shift between consecutive frames. Thus, after our background subtraction step, we subtract the foreground pixels between 2 consecutive frames and use the following simple rules:

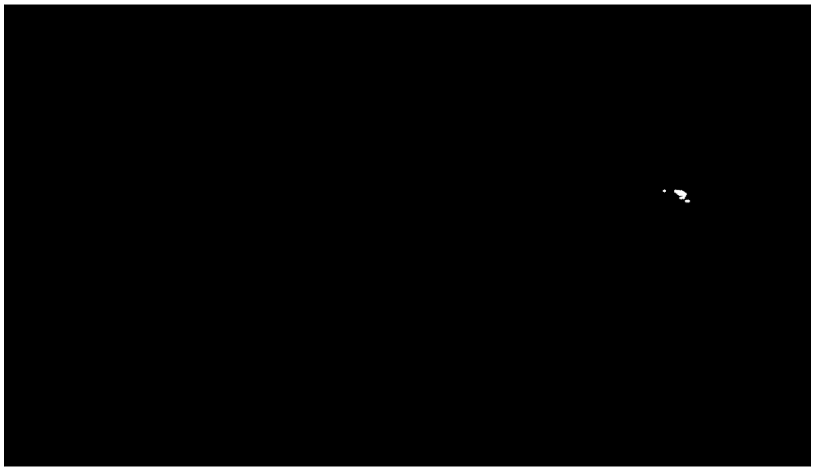
1. If a moving object (detected as foreground after background detection step) is not detected in at least 3 continuous frames (after subtracting consecutive frames), it is removed from the foreground.

2. If the moving object has not moved more than a predefined number of pixels, it is removed from the foreground.

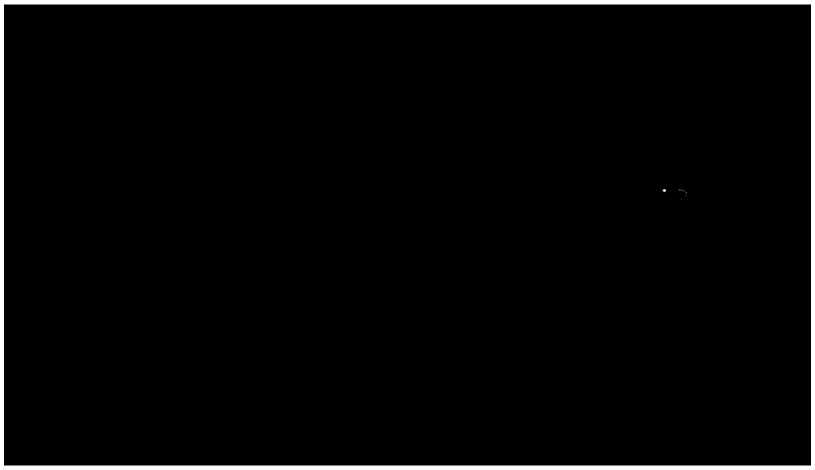
The figure 3.15 shows an example of a car moving in the background. This object will appear in the foreground after background subtraction. Using rule 1 above, the car, highlighted using the white rectangle, is identified as a slowly moving object and will not be considered for further processing.



(a)



(b)



(c)

Figure 3.15: Illustration of the proposed approach to identify and remove slowly moving objects. (a) Original video frame. (b) Resullts of simple background subtraction. A connected component of the car will be detected. (c) Result of forground subtraction. The connected component of the car is small and will not be detected

3.5 Ball tracking

Ball tracking has been improved gradually since the beginning of the project. The first issue that we encountered is that the ball may not be detected in some frames. The background subtraction algorithm may fail to detect the ball. In these cases, Kalman filter can be used to predict the position of the ball and maintain a smooth path. Figure 3.16 shows an example where the ball is missed but Kalman filter continues to give a good prediction of the next position ball.

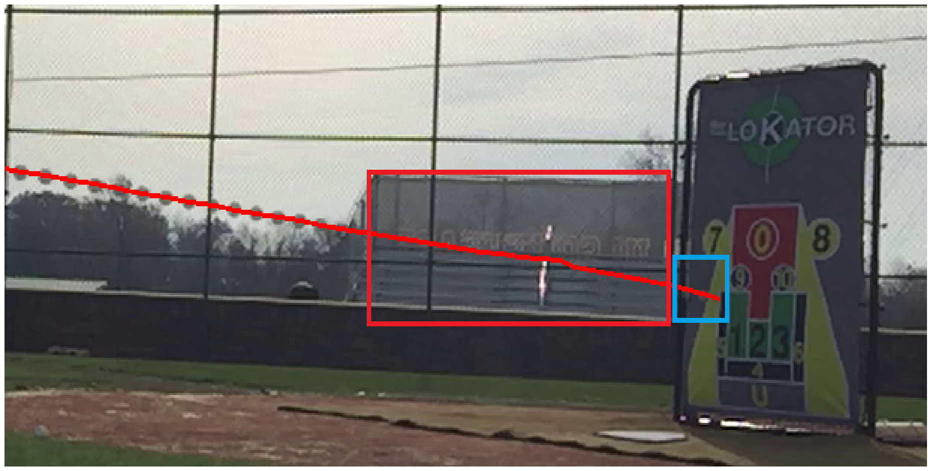


Figure 3.16: The ball is missed in the red box due to the white building (same color as the ball) but Kalman filter predicted smoothly the next locations of the ball. In the blue box, the software was again able to detect the ball twice.

Different scenarios could also appear while tracking the ball as is illustrated in the figure 3.1. The first scenario is when the ball bounces off the ground. Naturally, the ball's distance to the ground decreases gradually as it travels towards the target. As a result, if we detect that the ball position is getting higher than the previous positions before reaching the target, we will assume that it has hit the ground and bounced off. In this case, we will report this as an invalid pitch, go back to component 2 in figure 3.1 and wait for the next pitch. Figure 3.17 shows an example of a ball bounce detection.

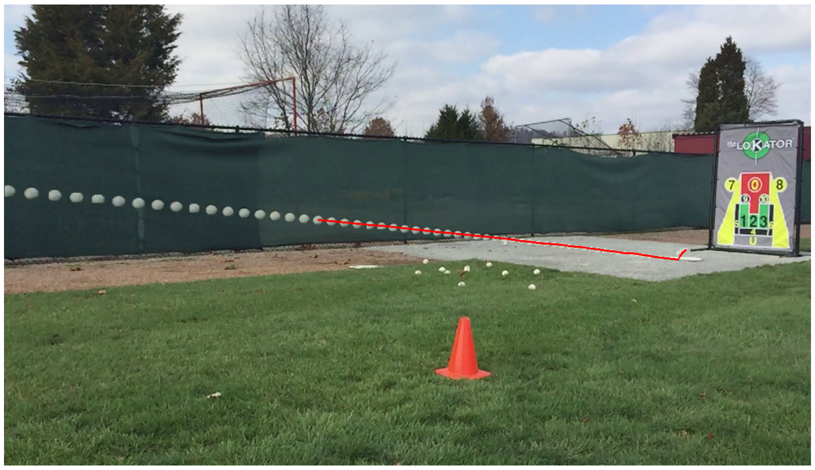


Figure 3.17: An example of a pitch where the Ball bounce off the ground. The algorithm stopped tracking the ball after detecting that it started getting higher.

Once the ball reaches the target area, it triggers component 4 as illustrated in figure 3.1. This component analyzes the ball motion and position in more details to detect if the ball (a) hits the target, (b) hits the target frame, (c) misses the target and passes in front of it, or (d) misses the target and disappears behind it. Our algorithm to detect these scenarios is outlined below:

a) Ball hits the target: when the ball hits the target, it will cause it to deform, i.e., many of its pixels will start moving in between frames. However, this motion should be distinguished from minor motions that can be caused by wind or by ball moving too close to it and at a high speed. First, we compute the sum of the differences of the pixels' values in the target window between the current frame and the frame at the end of component 3 (i.e., before we detect that ball is close to the target). Figure 3.18 displays the difference between 2 consecutive frames of all pixels within the target. In Figure 3.18 (a), the ball did not hit the target yet, as a result, only pixels that correspond to the moving ball have changed. On the other hand, in figure 3.18 (b) the ball has just hit the target and as a result most of its pixels have moved. Next, we detect the number and areas of connected components in the image of differences. The idea is that if the difference is caused by the moving ball only, we should get only one or a few connected components with small areas. On the other hand, major target deformation will results in a large number of connected components or few components with very large areas. Thus, our algorithm will decide that the ball has hit the target if the sum of differences exceeds a threshold, the number of connected components is more than 3, or the sum of the areas of the connected components exceeds a threshold. In this case, the target zone will be identified and reported.

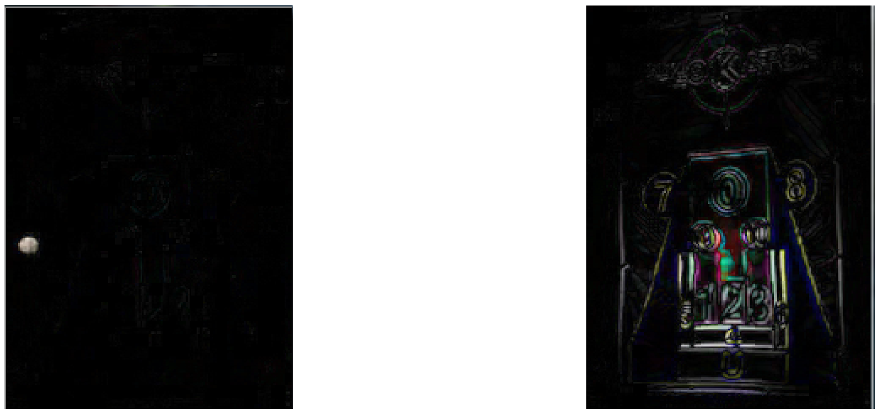

(a) (b)

Figure 3.18: Comparison of the pixels's difference in the target window (a) before and (b) after the ball hits the target.

b) Ball hits the target frame: If the ball's path reverses direction, we detect the instance the ball bounces back and report this as an invalid pitch due to ball hitting the frame. Figure 3.19 displays an example of this scenario.

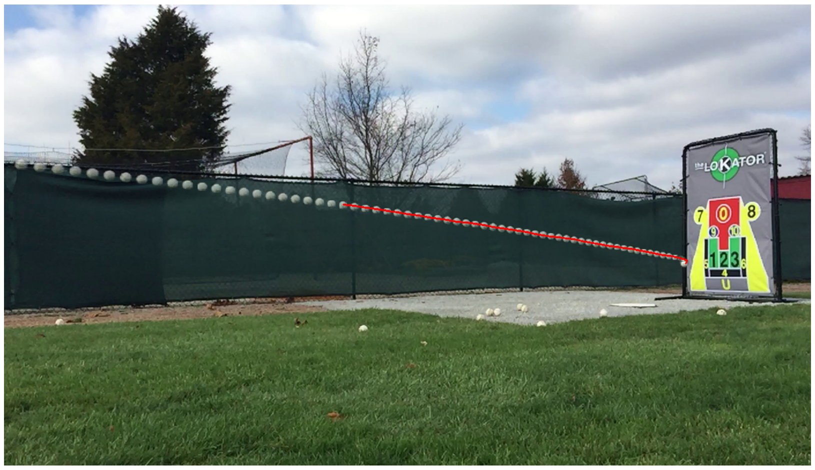


Figure 3.19: A sample pitch where the ball hits the frame.

c) Ball passes the target without hitting it: If we detect that the ball has moved away from the target region without hitting the target or its frame (cases a) and b) above), then we assume that the pitch has missed the target completely and report it as invalid.

d) Ball cannot be detected any longer:If the ball cannot be detected in several consecutive frames, then we assume that it has disappeared behind the target and report it as invalid. We should note here that, due to the non-uniform background (target frame with colors similar to the ball's color), we may miss detecting the ball in few frames. In order to avoid a premature decision that the ball has passed behind the target, we require that the ball goes undetected for several consecutive frames. Figure 3.20 shows an example where the ball disappeared behind the target.

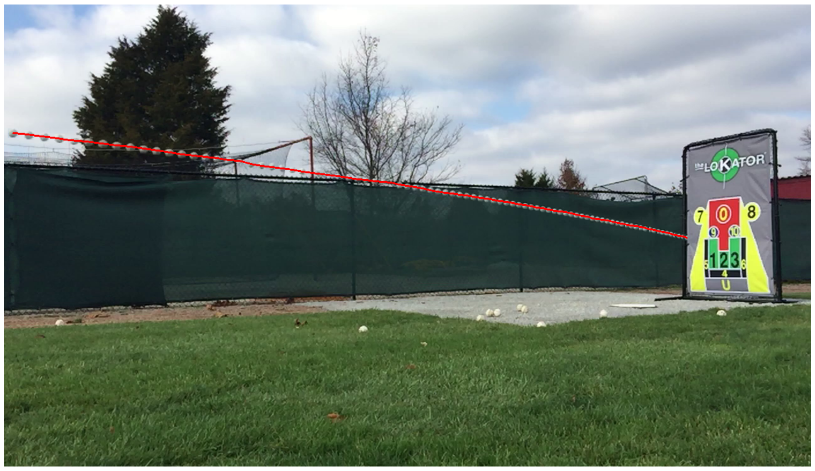


Figure 3.20: A sample pitch where the ball has disappeared behind the target

A flowchart of our algorithm to analyze the ball motion and position as it approaches the target is displayed in Figure 3.21. The 4 different scenarios discussed above are highlighted in orange.

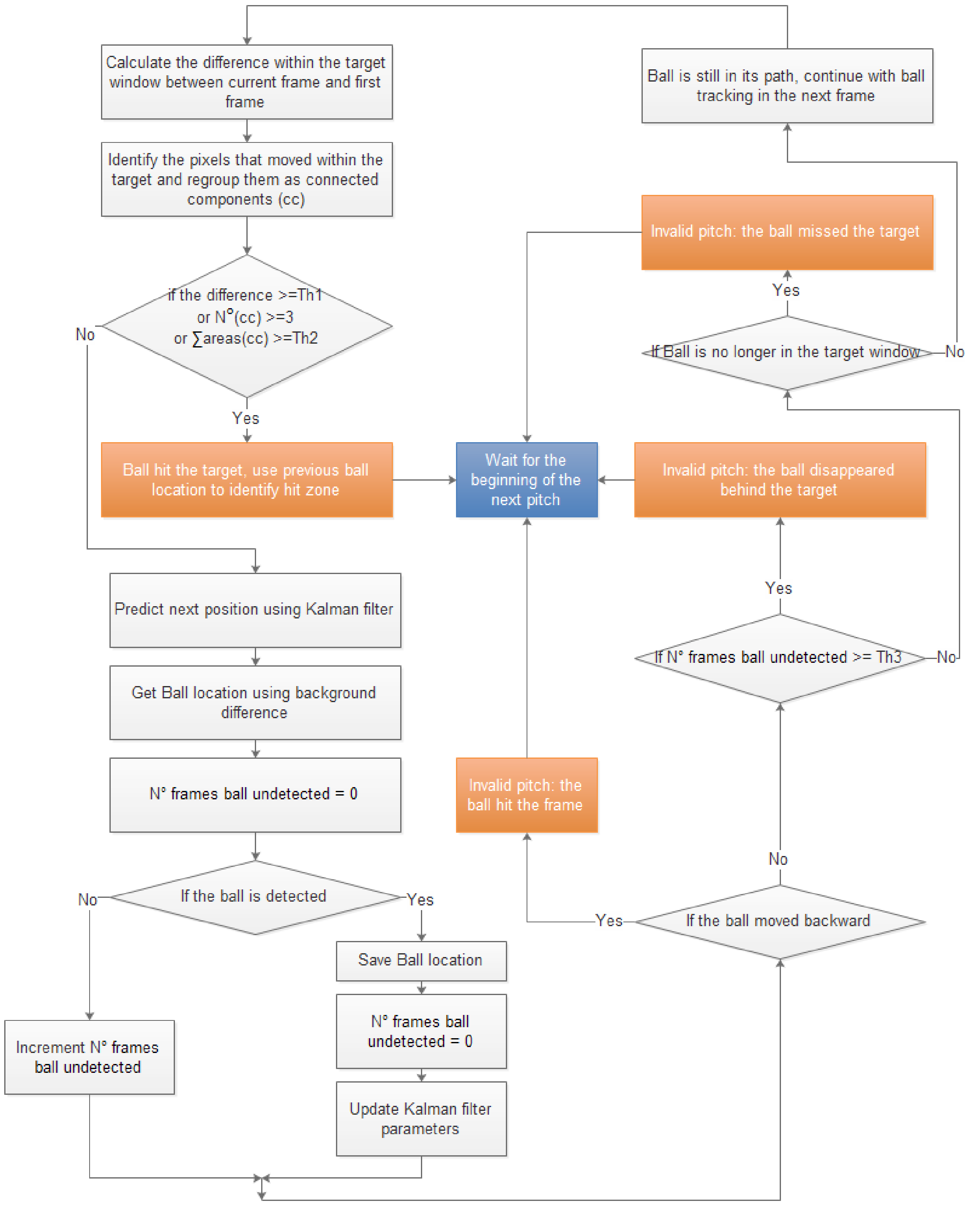

Calculate the difference within the target window between current frame and first frame
Identify the pixels that moved within the target and regroup them as connected components (cc)
if the difference >=Th1 or N°(cc) >=3 or ∑areas(cc) >=Th2
No
Yes
Ball hit the target, use previous ball location to identify hit zone
Wait for the beginning of the next pitch
Ball is still in its path, continue with ball tracking in the next frame
Invalid pitch: the ball missed the target
Yes
If Ball is no longer in the target window
No
Invalid pitch: the ball disappeared behind the target
Yes
If N° frames ball undetected >= Th3
No
Predict next position using Kalman filter
Get Ball location using background difference
N° frames ball undetected = 0
Invalid pitch: the ball hit the frame
No
If the ball is detected
Yes
Yes
If the ball moved backward
No
Save Ball location
N° frames ball undetected = 0
Update Kalman filter parameters
Increment N° frames ball undetected

Figure 3.21: A flowchart of our algorithm to analyze the ball motion and position as it approaches the target to decide if the ball hits the target, hits the target frame, misses the target and passes in front of it, or misses the target and disappears behind it.

3.6 Speed estimation

One of the tasks of this project is to estimate the speed of the ball as soon as it appears in the frame. To compute the speed of the ball we need to identify two parameters: (1) the distance traveled by the ball in a fixed number of frames, and (2) the temporal resolution of the video recording, i.e., the number of frames per second. The latter one can be extracted from the video header and is typically 240 frames per second. The number of pixels traveled by the ball can be easily computed using the position of the ball detected at each frame. The main challenge is converting the distance from pixels to physical distance. That is, identifying the pixel's spatial resolution. First, the conversion needs to be accurate as a small deviation can cause a large error in the estimated speed. Second, due the camera position, the resolution decreases as the ball travels. Figure 3.22 illustrates this fact by showing the trajectory of the ball for a sample pitch. As it can be seen, the size of the ball decreases gradually as it travels towards the target. Since the ball's size is fixed, this indicates a change in the spatial resolution.

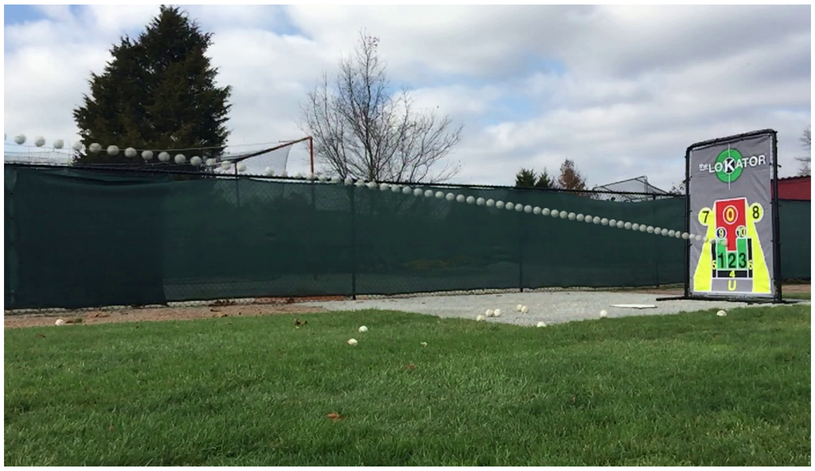


Figure 3.22: Trajectory of a sample pitch. The decrease in the size of the ball as it travels towards the target indicates a decrease in the spatial resolution (as the ball moves farther from the camera).

To address the non-constant pixel resolution issue, we assumed that the camera position and angle will be fixed and performed a calibration experiment. The sponsor provided us with recordings that included a stick, with a tick marker each 1 foot, in the background. Figure 3.23 displays one video frame with this setting.

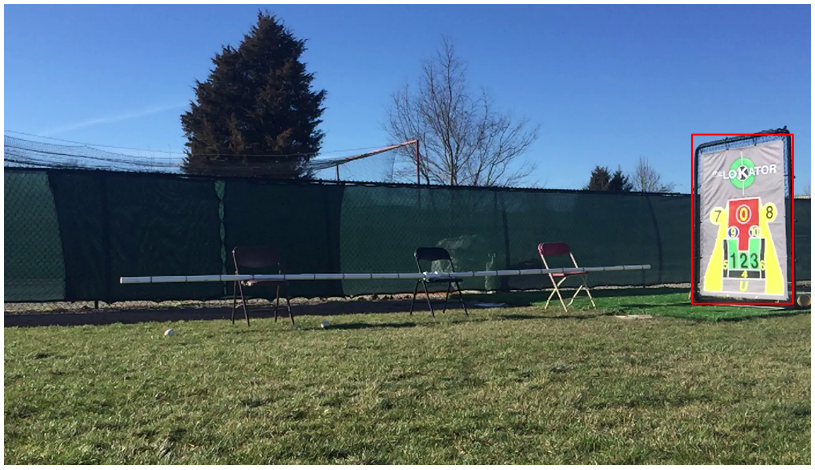


Figure 3.23: Setting used for calibration to learn the pixels spatial resolution at different locations.

Using multiple videos with this setting, we measure the length (in pixels) of each 1 foot section on the stick. Then, we plot this length versus the distance of the 1-foot segment's center from the left edge of the frame (we call this the x-coordinate of the section center). Figure 3.24(a) displays the collected measurements. As it can be seen, the resolution decreases (1-foot segment appears shorter) as we move to the right side (i.e., x-coordinate increases). More importantly, the decrease fits a linear pattern. To characterize this pattern, we fit a linear regression model and learn its parameters. The learned linear model is displayed in Figure 3.24 (b). This model can be used to estimate the dynamic pixel resolution at any location within the frame. We should note here that we have verified that the vertical distance is not significant enough to affect the resolution, and thus will be ignored.

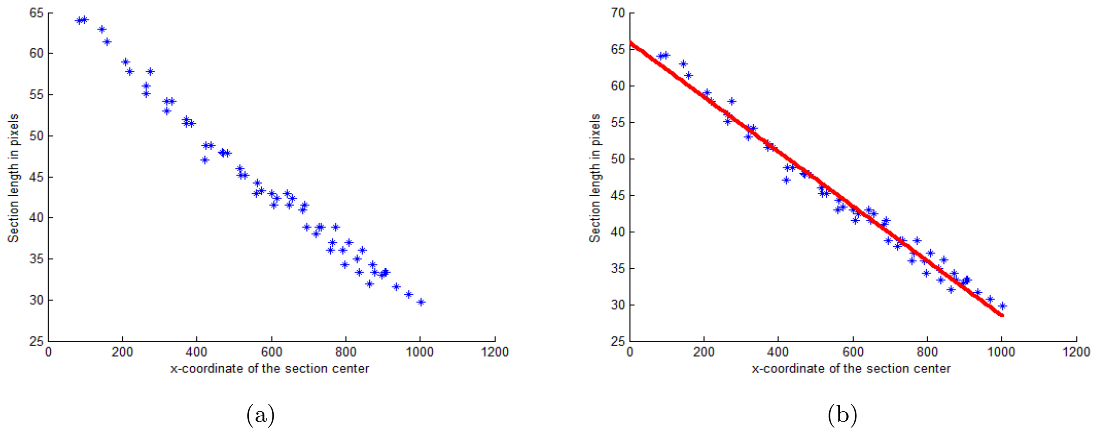


Figure 3.24: Learning the variation of the spatial resolution. (a) Length (in pixels) of 1 foot section versus its distance (in pixels) from the left edge of the image. (b) Linear model fit to the data in (a).

Once the system is calibrated, we use the first N frames (typically $N = 10$) after the ball appears on the view. For each frame, we compute the distance traveled in pixels, then convert this to actual distance using the learned regression model and the position of the first ball to map from foot length to number of pixels. The sum of all N distances is then converted to speed using the temporal resolution (frames/sec) of the video recording.

We evaluated our approach on a video collection where an upper and a lower speed limits are provided. We found that the estimated speed is usually similar to the upper speed limit. Figure 3.25 shows an example of comparison between the true speed limits and the estimated speed.

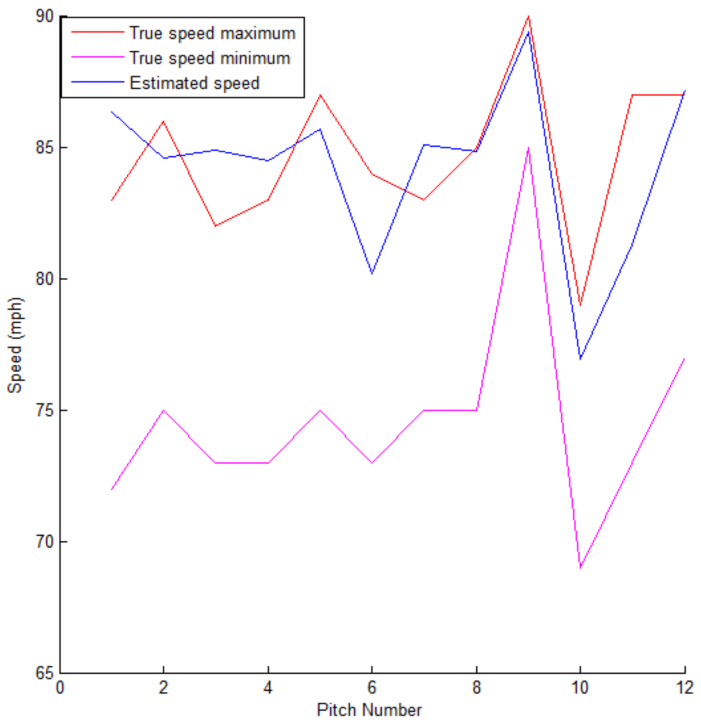


Figure 3.25: Comparison of the true speed limits and the estimated speed in one video example

3.7 Vertical displacement

This requested feature focuses on the difference of the vertical position of the ball between its first appearance in front of the camera and once it hits the target. We used the same approach in speed computation. We calculate the vertical displacement in number of pixels between the highest and the lowest position and we convert it to meters. The sponsor provided us videos with a vertical stick. Figure 3.26 shows a video frame with this setup.

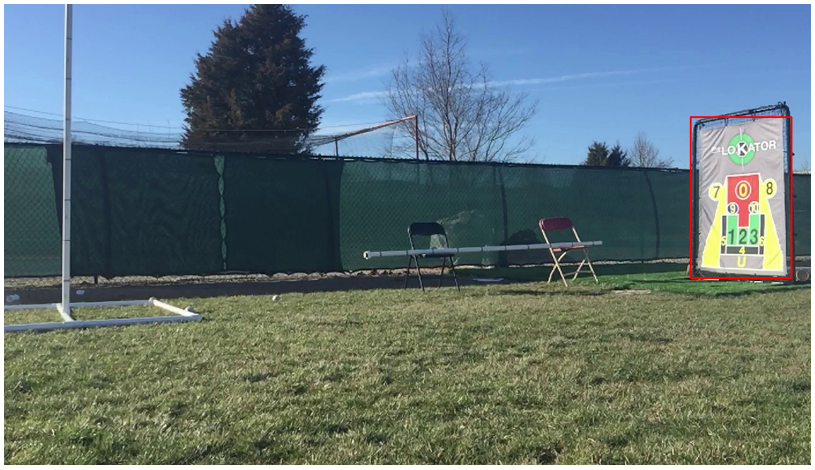


Figure 3.26: Vertical stick used to map a foot length to number of pixels.

We found that the vertical resolution (number of pixels per foot) is almost the same for all the sections of the stick. We extracted the mapping constant (usually 60 pixels per foot) from one section and we used it to calculate the vertical displacement. Figure 3.27 shows an example of calculated displacement.

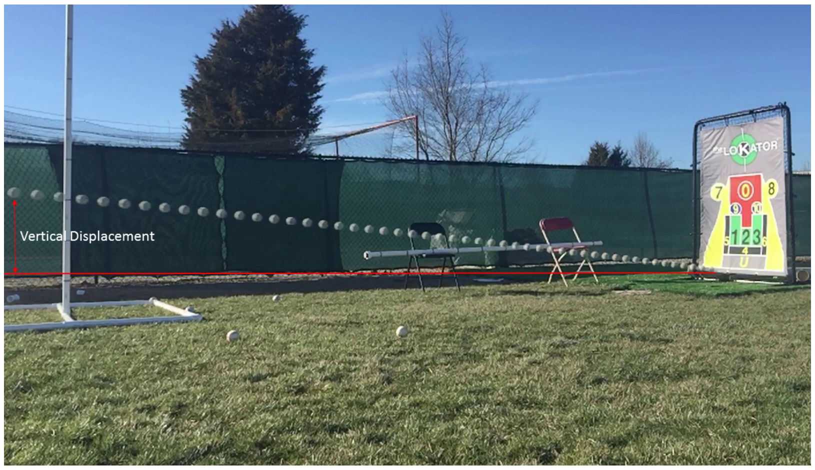
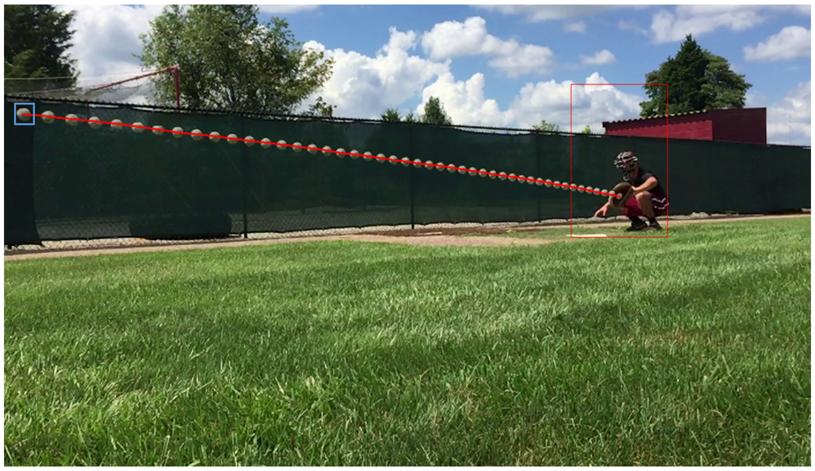


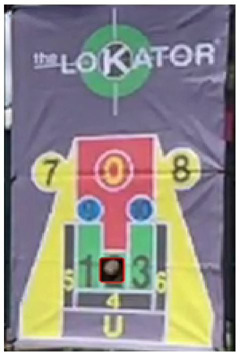
Figure 3.27: Example of calculated vertical displacement. The red lines show the highest and lowest positions of the ball. The obtained distance is equal 1.93 feet which is equal to 0.58 meter.

3.8 Using a catcher instead of a pitching target

Under this task, we investigated the possibility of replacing the target with a catcher and estimate the zones that would have been hit if a pitching target was there. The target is provided in the first few frames of the video then replaced. New problems arise with the catcher replacement. First, we can not use the perturbation of the target to detect the hit anymore. Moreover, the catcher hand can be moving in the same time as the ball is approaching which makes the distinction between the ball and the catcher's gloves hard. To solve this issue, we used a different method for tracking the ball. First, we save a copy of the ball sub-image once it appears in front of the camera. After that, once the ball is getting near to the catcher, we apply multi-scale template matching algorithm, using the saved ball patch as a template, to locate its next position. In every frame, we threshold the scores obtained from template matching to detect if the ball has disappeared and if it was caught by the catcher.



(a)



(b)

Figure 3.28: Example of ball catch. The location of the hand corresponds to the zone 2. The blue box corresponds to the saved ball patch. (a) trajectory of the ball until it is caught by the catcher. (b) Zone corresponding to the location of the caught ball.

CHAPTER 4

EXPERIMENTAL RESULTS

In this chapter, we will discuss the experimental results obtained from running the developed software on a video dataset.

4.1 Pitch accuracy results

4.1.1 Test datasets

The test datasets include 15 videos provided by the sponsor. Each video recording contains multiple pitches. The hit location on the target is provided and used to evaluate the results of our software. Table 4.1 summarizes the datasets.

TABLE 4.1

Test dataset summary

Video	settings	duration	N frames/sec	N pitches
1	$d_1 = 25$, $d_2 = 20$	16 min	240	10
2	$d_1 = 25$, $d_2 = 30$	23 min	240	10
3	$d_1 = 25$, $d_2 = 30$	22 min	240	11
4	$d_1 = 35$, $d_2 = 20$	20 min	240	10
5	$d_1 = 35$, $d_2 = 20$	21 min	240	14
6	$d_1 = 35$, $d_2 = 30$	18 min	240	12
7	$d_1 = 35$, $d_2 = 30$	21 min	240	14
8	$d_1 = 20$, $d_2 = 20$	12 min	240	19
9	$d_1 = 30$, $d_2 = 35$	8 min	240	5
10	$d_1 = 30$, $d_2 = 35$	8 min	120	11
11	$d_1 = 30$, $d_2 = 35$	16 min	240	4
12	$d_1 = 30$, $d_2 = 35$	14 min	240	8
13	$d_1 = 30$, $d_2 = 35$	8 min	120	10
14	$d_1 = 25$, $d_2 = 25$	11 min	240	7
15	$d_1 = 20$, $d_2 = 20$	16 min	240	9

In table 4.1, the settings column describes the location of the camera relative to the target as illustrated in figure 4.1. d_1 and d_2 are measured in feet.

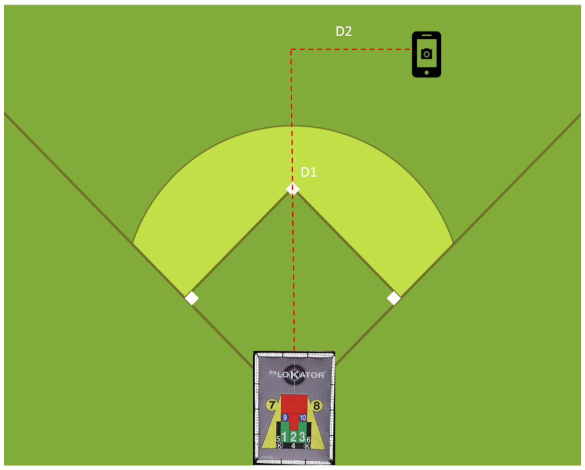


Figure 4.1: Illustration of the camera location settings.

In all our experiments, we use a 3.5 GHz 6-core processor. The processing of each video frame takes between 10 and 30 ms. The memory size is not issue since we process the video incrementally one frame at a time.

4.1.2 Pitch inter-class accuracy

To measure the pitch accuracy, we defined 4 classes that identify all possible outcomes:

- HZ: A zone is hit within the target.
- BB: The ball bounced off the ground.
- FH: The target frame is hit.
- NH: The ball went beyond the target without touching it.

Tables 4.2 through 4.16 illustrate the pitches inter-class confusion matrix for each video in the test dataset. For each conflict, we will explain the reason for the misdetection.

TABLE 4.2

Video 1

	Truth				
	HT	BB	FH	BT	
HT	8				100 %
BB					
FH			1		100 %
BT				1	100 %

TABLE 4.3

Video 2

	Truth				
	HT	BB	FH	BT	
HT	9				100 %
BB					
FH					
BT				1	100 %

TABLE 4.4

Video 3

	Truth				
	HT	BB	FH	BT	
HT	9		1		90 %
BB					
FH			1		100 %
BT					

TABLE 4.5

Video 4

	Truth				
	HT	BB	FH	BT	
HT	9				100 %
BB		1			100 %
FH					
BT					

TABLE 4.6

Video 5

	Truth				
	HT	BB	FH	BT	
HT	9	2			81 %
BB		1			100 %
FH		1	1		50 %
BT					

TABLE 4.7

Video 6

	Truth				
	HT	BB	FH	BT	
HT	11				100 %
BB					
FH					
BT				1	100 %

TABLE 4.8

Video 7

	Truth				
	HT	BB	FH	BT	
HT	10	1			90 %
BB		1			100 %
FH			2		100 %
BT					

TABLE 4.9

Video 8

	Truth				
	HT	BB	FH	BT	
HT	8				100 %
BB					
FH			1		100 %
BT					

TABLE 4.10

Video 9

	Truth				
	HT	BB	FH	BT	
HT	4				100 %
BB					
FH					
BT				1	100 %

TABLE 4.11

Video 10

	Truth				
	HT	BB	FH	BT	
HT	10				100 %
BB					
FH			1		100 %
BT					

TABLE 4.12

Video 11

	Truth				
	HT	BB	FH	BT	
HT	4				100 %
BB					
FH					
BT					

TABLE 4.13

Video 12

	Truth				
	HT	BB	FH	BT	
HT	4				100 %
BB		2			100 %
FH					
BT	1			1	50 %

TABLE 4.14

Video 13

	Truth				
	HT	BB	FH	BT	
HT	7				100 %
BB		2			100 %
FH					
BT				1	100 %

TABLE 4.15

Video 14

	Truth				
	HT	BB	FH	BT	
HT	5				100 %
BB		2			100 %
FH					
BT					

TABLE 4.16

Video 15

	Truth				
	HT	BB	FH	BT	
HT	6	1			85 %
BB		1			100 %
FH				1	0 %
BT					

Figure 4.2 illustrates one incorrectly detected pitch from the video 3. In this case, the ball hit the lower part of the frame and continued upward. It did not stop moving forward. As a result, the pitch was not detected as a frame hit. Instead, our algorithm detected it as hitting the purple zone containing the U letter.

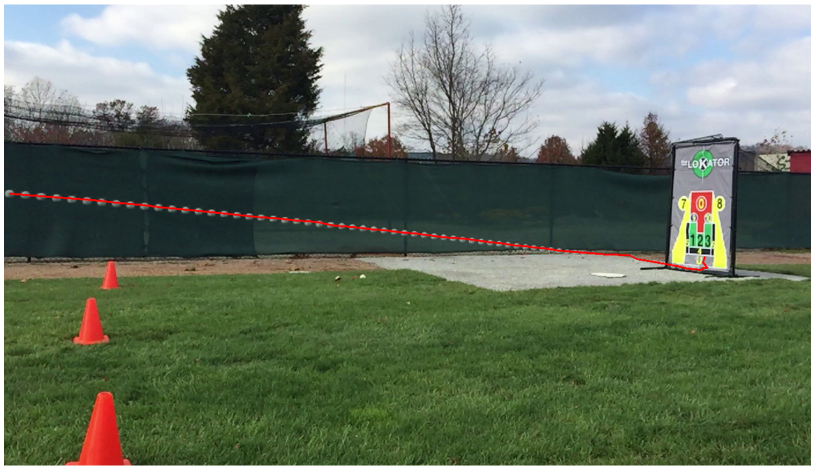


Figure 4.2: Incorrectly detected pitch. Ball hits the lower frame and proceeded to hit the target. Our software detected it as hitting zone U

In the video 5, all the incorrectly detected pitches are actually ball bounces. In all these cases, the bounce is very near to the target. In fact, using a 2-D view of a real 3-D scene, it is hard to distinguish between a ball within the the target frame and a ball close to the frame. As a result, most of the ball bounces close to the target are missed. Figure 4.3 shows an example of a misclassified pitch. In this case, the pitch is detected as a frame hit.

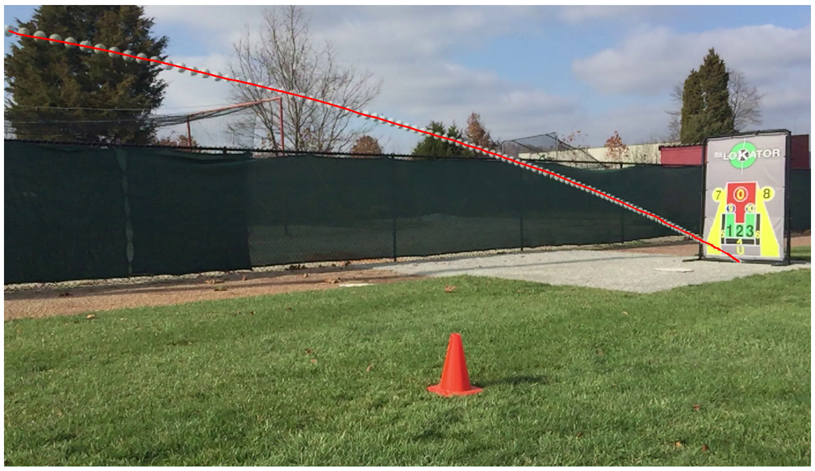
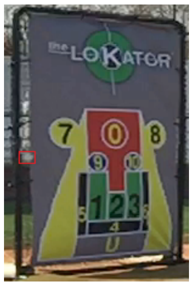


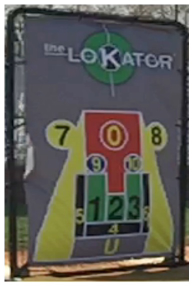
Figure 4.3: A sample pitch from video 5 where the ball was incorrectly classified as hitting the target frame. In this case, the ball bounces off the ground, very close to the target frame.

In this video, we have a pitch that hit the target. However, the software detected as it went beyond the target. Actually, if we see the video, we find that the ball hit the purple zone and went between the target and the frame. Figure 4.4 shows a set a images illustrating the movement of the ball.

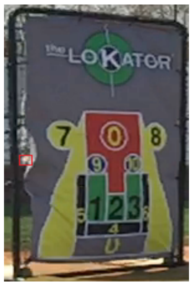
Figure 4.4 displays a pitch from video 12 as 'NH'. By examining the different steps that led the algorithm to misclassify this pitch, we observe the ball hits the purple zone and then went between the target and the frame.



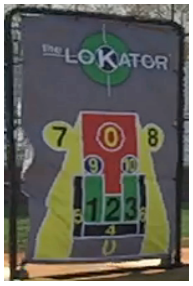
(a)



(b)



(c)



(d)

Figure 4.4: Ball hits the purple zone and went between the target and the frame. (a) the ball hits the purple zone. (b) the ball disappears behind the cage of the target. (c) the ball returns. (d) the ball disappears between the target and the frame

Figure 4.5 illustrates the software output for the same pitch. The result of the prediction phase of the Kalman filter was outside the target rectangle. As a result, the pitch was detected as beyond the target rectangle.

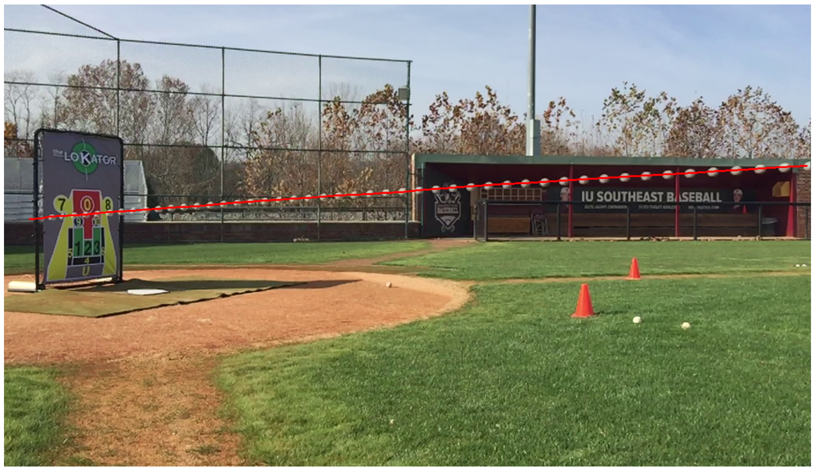


Figure 4.5: Path of the incorrectly detected pitch illustrated in figure 4.4. The ball disappeared between the target and the frame and the software Kalman filter predicted a position that went beyond the target.

4.1.3 Evaluation of the zone identification component

If the ball hits the target, our software detects the zones that are hit. In this section, we report the accuracy of the zone identification component of our software. The zones annotation provided by the sponsor usually include only one zone per pitch. However, zones can be very narrowed and many pitches hit multiple zones. To handle these cases, the developed software reports three zones that have the highest percentages of pixels from the ball. For evaluation purposes, we put an additional constraint that the reported percentages in one zone have to be higher than 30% to report that zone as being hit. Figure 4.6 shows an example where the ball hits the border between zone 7 and the purple zone. In this pitch, zone 7 was reported as hit with a percentage of 63% and the purple zone is reported with 36%.

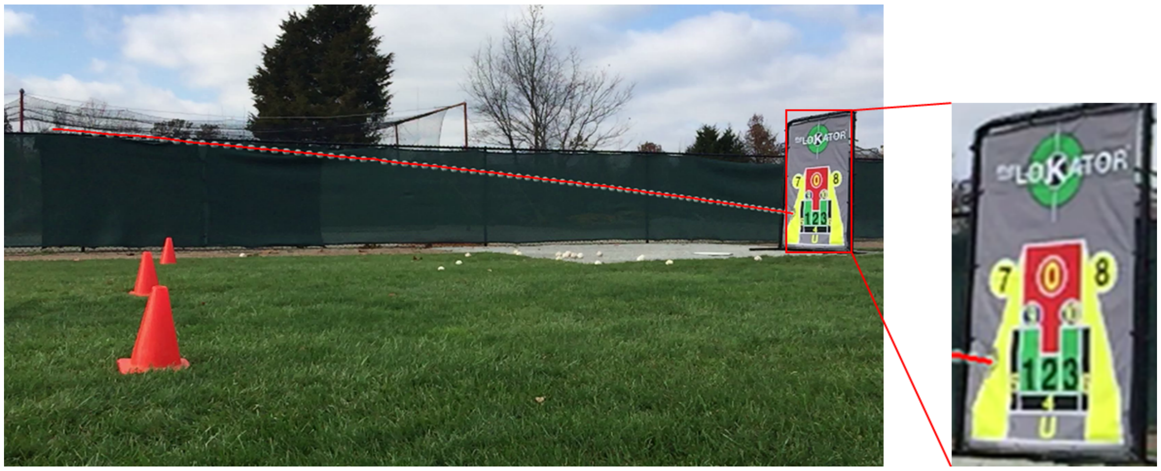


Figure 4.6: Ball hit the border between zone 7 and the purple zone

Table 4.17 reports the accuracy of our zone identification component. Figure

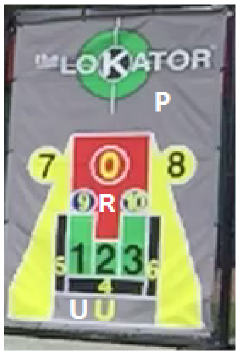


Figure 4.7: Illustration of the unnumbered zones R, U and P

We should note that most of the misclassifications reported in table 4.17 are the results of the ball hitting multiple zones. For example, zone 5 was detected in one case as zone 1 and zones 9 and 10 were detected few times as the red zone that surrounds them.

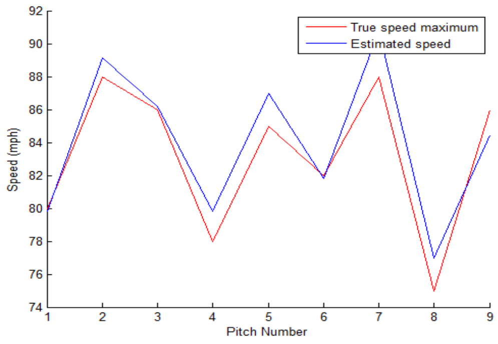
TABLE 4.17

Zones confusion matrix

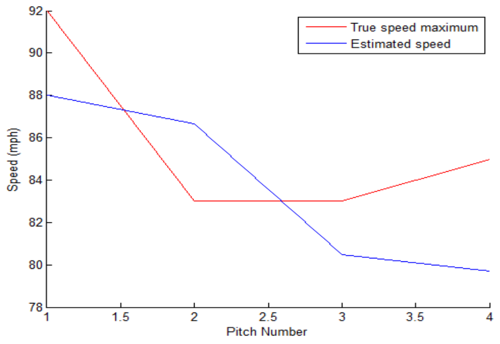
	Truth														
Measured	0	1	2	3	4	5	6	7	8	9	10	R	U	P	Accuracy
0	3														100 %
1		4				1									80 %
2		1	1												50 %
3				6											100 %
4					7										100 %
5						1									100 %
6							2								100 %
7								22							100 %
8									5						100 %
9										5					100 %
10											2				100 %
R										3	2	17			77 %
U													5		100 %
P									1			1	1	22	88 %

4.2 Evaluation of the speed estimation component

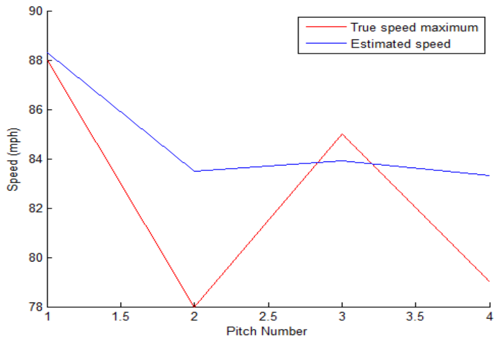
The speed computation was based on fixing the camera location and estimating the image pixel's real size. The sponsor provided us with 8 videos recordings where the ball speed was estimated using a radar. For each pitch, the ball's speed was estimated at two locations providing a minimum and a maximum speed. 3 out of the 8 video have only the maximum speed provided. These videos contained a measuring stick with markings on each 1-foot section. Section 3.6 described our proposed method for speed calculation. By evaluating the accuracy of the speed calculation, we found that the computed values are usually close to the true speed upper limit captured by the radar. Figure 4.8 compares the speed reported by our software to the true speed.



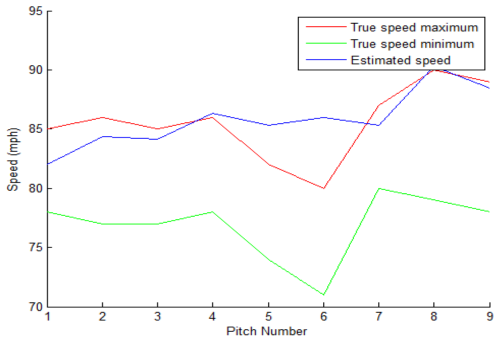
(a)



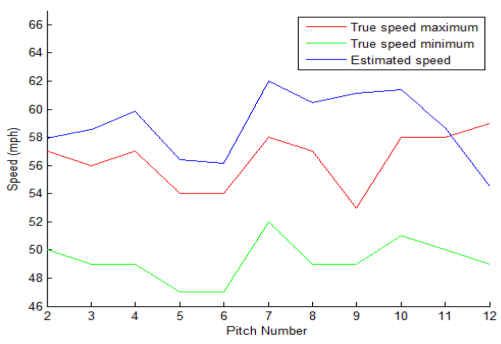
(b)



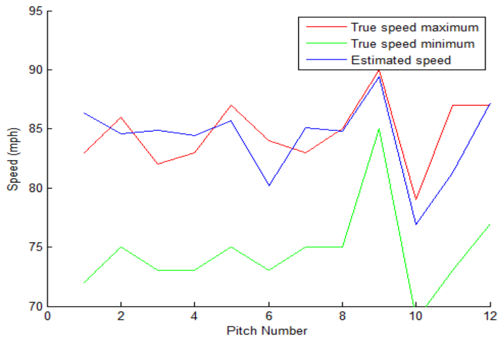
(c)



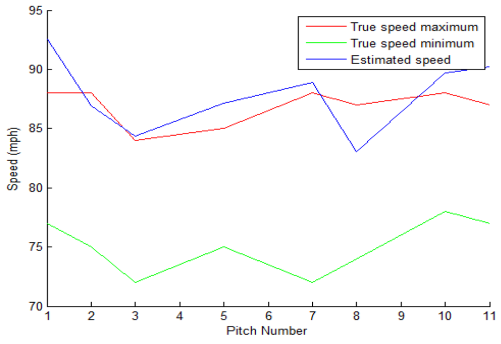
(d)



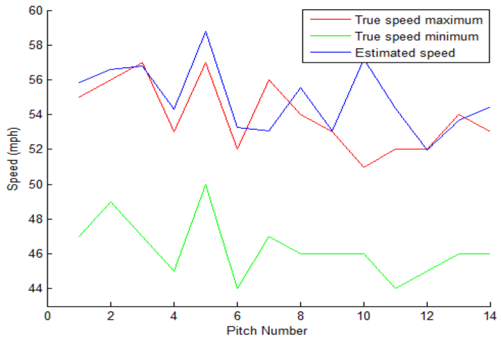
(e)



(f)



(g)



(h)

Figure 4.8: Comparison of the calculated speed for all pitches of the 8 videos with ground truth.

To quantify the accuracy of our method, we calculated the mean and standard deviation of the difference between computed values and the true upper limit values for all the pitches in the dataset. We found that the mean error is 0.7544 and the standard deviation of the error is 2.7054. As a result, we can report that the measured speed can have a maximum error of ±3.5 mph.

4.3 Target replacement results

The final task of this project was to replace the target with a catcher and check if we can adapt the developed algorithms to keep the same results quality as in the target case. By using the method described in section 3.8, the software was able to predict the target zone associated with the ball catch in different cases. Figures 4.9 and 4.10 show examples of ball catch.

(a) (b)

Figure 4.9: System performance when a catcher is used. (a) Path of the ball and its position when caught by catcher. The red square indicates the position of the target (before it was replaced by the catcher). (b) Replacing the catcher with a virtual target to read the hit zone

(a) (b)

Figure 4.10: Another example of system performance when a catcher is used. (a) Path of the ball and its position when caught by catcher. The red square indicates the position of the target (before it was replaced by the catcher). (b) Replacing the catcher with a virtual target to read the hit zone

CHAPTER 5

CONCLUSIONS AND POTENTIAL FUTURE WORK

5.1 Conclusions

In this thesis, we developed a computer vision based software system that automates the pitch analysis in baseball. The new software is designed to work with the Lokator pitching target to automate the following tasks:

1. Accurate reading of the pitch location on the target.
2. Accurate estimation of the velocity of the ball.
3. Provide contextual information about the pitch, such as vertical movement of the ball which can indicate late breaking.
4. Replace the target by a catcher and estimate the pitch location using a virtual target.

The input to the software are video files (recorded by an iPhone) that contain recordings of successive pitches. The target is located by applying multi-scale template matching. Identifying the different zones within it proved to be a challenging task due to several factors such as wind, illumination variation and target distortion. We solved this task by applying unsupervised learning methods on the pixels‘ colors and spatial locations.

To detect the ball while it is moving, we used background subtraction based on mixture of Gaussians. Motion noise is frequently present in the videos either due to dynamic background (e.g., moving camera, tree motion due to wind) or moving entities such as cars or birds. To filter this effect, we developed a technique based on subtracting consecutive foregrounds to eliminate slow motion. In some frames, depending on the background variation, the ball can be missed. In these cases, Kalman filter is used to predict the position of the ball and maintain a smooth path. As the ball approaches the target, several possible scenarios can arise including: ball bounces off the ground, Ball hits the frame and bounces back, ball passes the target without hitting it or ball hits one of the target zones. We have developed algorithms that can detect all of the above cases.

To compute the speed of the ball, the major challenge was the conversion from distance in pixels to a physical distance. Due to the camera location, the pixel resolution decreases as the ball travels towards the target. To address this issue, we fixed the camera position and angle and performed a calibration experiment by using a stick, with a tick marker each 1 foot. We found that a foot length in pixels decreases in a linear pattern. We used linear regression to fit its parameters. A similar approach was used to compute the vertical displacement. A vertical stick is used to calculate the vertical resolution but we found out that the vertical resolution does not vary significantly.

The final task was the investigation of the replacement of the target with a catcher and estimate the zones that would have been hit if a pitching target was there. New problems arose and different algorithm needed to be developed to identify the instant the ball caught.

We performed several testing experiments. To measure the pitch location accuracy, the sponsor provided us a set of recordings along with the true hit location for each pitch. The software was able to report correct results except some special cases. The sponsor also provided us a set of videos for measuring the accuracy of velocity computation. The software reports a speed that is within ± 3.5 mph from the maximum true speed.

5.2 Potential future work

- Currently, we use only one camera. A possible improvement is to use two cameras and perform 3-D modeling.
- we can use another camera to focus on the pitcher and analyze the motion of his arms and correlate this info with the pitch location on the virtual target.

REFERENCES

[1] Margaret Livingstone, *Vision and Art: The Biology of Seeing (New York: Harry N. Abrams)*, 2002.

[2] Richard Szeliski, *Computer vision: algorithms and applications*, Springer Science & Business Media, 2010.

[3] Sukhpreet Singh, “Optical character recognition techniques: A survey,” .

[4] Georgios Kordelas, J Perez-Moneo Agapito, J Vegas Hernandez, and P Daras, “State-of-the-art algorithms for complete 3d model reconstruction,” *Proceedings of the Engage Summer School, Zermatt, Switzerland*, vol. 1315, pp. 115, 2010.

[5] Divyarajsinh N. Parmar and Brijesh B. Mehta, “Face recognition methods & applications,” *CoRR*, vol. abs/1403.0485, 2014.

[6] Alex Teichman and Sebastian Thrun, “Practical object recognition in autonomous driving and beyond,” in *Advanced Robotics and its Social Impacts (ARSO), 2011 IEEE Workshop on.* IEEE, 2011, pp. 35–38.

[7] Graham Thomas, Thomas B. Moeslund, and Adrian Hilton, “Introduction to the use of computer vision in sports,” pp. 1–21, 2014.

[8] Rapsodo Pte Ltd, ,” http://rapsodo.com.

[9] Athla LLC, ,” http://www.athla.com/.

[10] Chris Stauffer and W Eric L Grimson, “Adaptive background mixture models for real-time tracking,” in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.* IEEE, 1999, vol. 2, pp. 246–252.

[11] Ahmed Elgammal, Ramani Duraiswami, David Harwood, and Larry S Davis, “Background and foreground modeling using nonparametric kernel density estimation for visual surveillance,” *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1151–1163, 2002.

[12] Greg Welch and Gary Bishop, “An introduction to the kalman filter,” 1995.

[13] Arthur P Dempster, Nan M Laird, and Donald B Rubin, “Maximum likelihood from incomplete data via the em algorithm,” *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[14] Richard A Redner and Homer F Walker, “Mixture densities, maximum likelihood and the em algorithm,” *SIAM review*, vol. 26, no. 2, pp. 195–239, 1984.

[15] JP Lewis, “Fast normalized cross-correlation,” in *Vision interface*, 1995, vol. 10, pp. 120–123.

[16] Ivan Culjak, David Abram, Tomislav Pribanic, Hrvoje Dzapo, and Mario Cifrek, “A brief introduction to opencv,” in *MIPRO, 2012 proceedings of the 35th international convention.* IEEE, 2012, pp. 1725–1730.

CURRICULUM VITAE

NAME: Mahdi Moalla

ADDRESS: Computer Engineering & Computer Science Department
Speed School of Engineering
University of Louisville
Louisville, KY 40292

EDUCATION:

M.Sc., Computer Science & Engineering
May 2017
University of Louisville, *Louisville, Kentucky*

B.Eng., Telecommunications Engineering
June 2015
Higher School of Communications of Tunis, *Tunis, Tunisia*

PROJECTS AND INTERNSHIPS:

1. Lokator project. Lokator is a baseball training system designed to document pitch location. It is composed of a pitching target and a smartphone app. The target is divided into a set of zones to identify the pitch location. The main limitation of the current system is its reliance on the user's feedback. We developed computer vision-based software to extend the Lokator system by adding the following contributions:
   (a) Automated and accurate reading of the pitch location on the target.
   (b) Automated and accurate estimation of the velocity of the ball.
   (c) Provide contextual information about the pitch, such as vertical movement of the ball.
   (d) Replace the target by a catcher and estimate the pitch location using a virtual target.
2. B.Eng graduation internship Project, Performed in Ooredoo Tunisia from Feb 2015 to May 2015. In this project, We developed a platform to supervise Ooredoo intelligent network. This

platform is used to keep track of the intelligent network servers' performance. It also reports statistics on the customers profiles.

HONORS AND AWARDS:

1. Higher School of Communications of Tunis Travel Award, June 2014
2. ACM TCPC Winning team, 2013 and 2014
3. ACM ACPC Maghreb champions trophy, 2013 and 2014