PROJECT

on

DETECT AND TRACK A BASEBALL

COURSE: ENGO 659

PROJECT: 01

Group: 09 (Nine)

Anan Ghosh (30220833)

Md Hanif Bhuian (30209982)

Motasem Alkayid (30198356)

Date of Submission

February 2, 2024

Schulich School of Engineering

University of Calgary

ENGO 659
Winter 2024

Group: 09

Title of Assignment: **<u>Detect and Track A Baseball</u>**

We, the undersigned, certify that this is our own work, which has been done expressly for this course, either without the assistance of any other party or where appropriate we have acknowledged the work of others. Further, we have read and understood the section in the university calendar on plagiarism/cheating/other academic misconduct and we are aware of the implications thereof. We request that the total mark for this assignment be distributed as follows among group members:

[illegible]

Md Hanif Bhuian..................... 100% Hours: 15 Date: Jan 27 to Jan 31, 2024

Anan Ghosh

Anan Ghosh .... 100% Hours: 15 Date: Jan 27 to Jan 31, 2024

Motasem Alkayid

Motasem Alkayid .... 100% Hours: 15 Date: Jan 27 to Jan 31, 2024

Table of Content

Executive Summary

The study introduces a technique for identifying and monitoring the movement of baseball pitches using two video clips. We employed a MATLAB algorithm for object detection, utilizing histogram analysis to define the ball's color edge and background subtraction to eliminate non-matching pixels. Additionally, we exclude stationary objects from consideration as we focus on detecting a moving object. Upon establishing the area of interest (AOI), we effectively detect the ball's position and track its trajectory. Addressing the challenge of a multi-colored background required careful threshold adjustments to ensure accurate detection of the small ball based on its shape, size, and color.

1. Background

Image processing object detection involves using image processing techniques to detect and identify objects in images or videos. It is an important task in computer vision and has various applications such as continuous object detection, and pedestrian/object recognition [1]. Similarly, Garikipati uses OpenCV with the YOLOv3 neural network to detect pedestrians and objects in videos, comparing the accuracy of different algorithms [2]. The detection of continuous objects, such as tracks and cables, is also addressed in the literature, with optimization algorithms proposed to improve recognition performance [3]. Overall, image processing object detection techniques play a crucial role in various domains, enabling tasks such as garbage classification, object recognition, and continuous object detection.
Background for image processing object ball detection involves various factors such as foreground object detection algorithms, neural mechanisms of visual object processing, salient object detection algorithms, and deep learning-based object detection. Deep learning-based foreground object detection algorithms have shown superiority over classical background subtraction (BGS)-based algorithms [4] . Visual processing of real-world objects involves a competition process between context and distractors in the native background [5]. A new algorithm based on boundary prior is proposed to estimate the background and improve salient object detection accuracy [6] . BackgroundNet is proposed to guide the learning process of deep learning-based object detection by using information from background images, resulting in improved classification performance for small-scale datasets[7].

1.1. Objectives

The study focuses on tracking and detecting objects, particularly baseballs, from two cameras in MATLAB which is provided by Major League Baseball (MLB). MLB is one of the richest sports industries while the National Football League (NFL) is the first one [7]. However, Baseball is the most popular game around the world, especially in North America. In 2006, MLB launched a technology named PitchFX that was developed by Sport Vision. The technology can detect and trackballs, their position as well as their movement.

1.2. Problem definition

The implementation of PitchFX technology in Major League Baseball (MLB) was intended to revolutionize the analysis of baseball pitches by providing detailed tracking of their movement. However, the effectiveness of this technology was hindered by the recording setup, which resulted in interleaved images from two cameras, leading to the capture of two balls per frame. This introduced a significant challenge in accurately detecting and distinguishing the balls from background elements, requiring the development of a robust ball detection algorithm. The primary objective of this project is to address this challenge by designing and implementing an algorithm capable of accurately identifying baseballs in recorded video files. This algorithm must effectively differentiate between the balls and other elements in the background based on factors such as brightness and shape. Successfully solving this problem will enable comprehensive analysis of pitch data, facilitating performance evaluation, strategic insights, and enhanced understanding of baseball dynamics in MLB.

2. Methodology

In this task a set of approaches were applied to detect and track the balls from camera A and camera B. To fulfill the function of ball detection the given methods include region of interest (ROI) detection, ROI masking, thresholding, filtering objects and finally displaying the outcomes of ball detection. The details of the methodology of this study are given below (fig. 1).

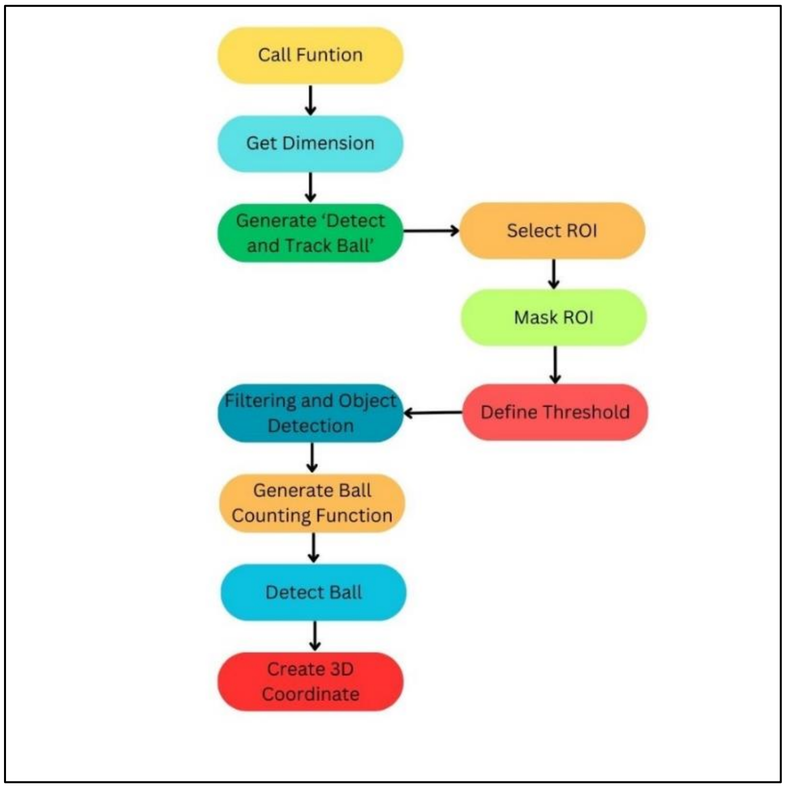

Call Funtion
Get Dimension
Generate 'Detect and Track Ball'
Select ROI
Mask ROI
Define Threshold
Filtering and Object Detection
Generate Ball Counting Function
Detect Ball
Create 3D Coordinate

Figure 1: Conceptual model of the study

2.1. Ball Detection Function

Following the structure, we used some built-in functions to detect and track baseball for both cameras. There are some codes and functions provided by the instructor and our job was to generate the pitchfx_detect function (renamed in our assigned project as ‘detectAndTrackBalls’) and run the function in ‘pitchfx_soup2nuts’ that is provided. However, in our project, we generated the function entitled ‘detectAndTrackBalls’ and run in the ‘pitchfx_soup2nuts.m’ file that was instructed and figured out the results as follows the instructions,

```
ballA.x     = column of detected ball
       .y     = row of detected ball
       .frame = frame number containing detected ball
       .image = image # within frame containing detected ball
```

Moreover, there was a sort of data and folders especially video clips (Camera A and B) and a project code template folder also provided that is mentioned below (table 1. and 2.).

Table 1. List of Given Data

Data/File ID	Purposes/details	Sources
clip_151004_152529	Folder for video clips for cameras A and B	Instructor
blobs.csv	‘.csv’ file for ball positions, movement, and other info	Instructor
project_code_dist	Supportive codes, algorithms, and functions exist for project progress.	Instructor

Table 2. List of given functions

Name of the functions/file	Purposes/details	Sources
pitchfx_soup2nuts.m	Main and all functions were run by the files	Instructor
detectAndTrackBalls.m	Ball detection and tracking functions	**Group 9**
DistortScreenPoint.m	Used to compute the effect of radial distortion on a given screen point's coordinates.	Instructor
DoPixelNormalization.m	Used to normalize a given screen point's coordinates to the lens image's coordinate system.	Instructor
GetActiveVideoCenter.m	Applied to calculate the center coordinates of the active video region based on the camera parameters such as the number of pixels	Instructor
GetActiveVideoCorners.m	Used to compute the coordinates of the minimum and maximum corners of the active video region based on the camera parameters,	Instructor
GetLensImageSize.m	to compute the size of the lens image based on the camera's parameters	Instructor
MaskRoi_A.mat	Used to define the region of interest (ROI) as masked to detect the balls within the image frame.	Instructor
MaskRoi_B.mat		Instructor

pitchfx_get_blobs_csv.m	to parse data from the "blobs.csv" file, extracting the (x, y) image coordinates and timestamps for camera A and camera B images.	Instructor
pitchfx_get_calibration.m	to read and parse the contents of a PitchFX calibration file specified by file_name and located in the directory given by pitch_path.	Instructor
pitchfx_get_pitch_info.m	to read and extract PitchFX data from the file named "pitch.info" located in the directory specified by pitch_path.	Instructor
pitchfx_read_video.m	to read PitchFX video data from camera A and camera B.	Instructor
pitchfx_solve_pitch.m	to determine the coefficients of the equations of motion for a baseball pitch based on the detected positions of the ball from two PitchFX cameras along with their timestamps.	Instructor
UnDistortScreenPoint.m	to reverse the distortion applied to a screen point (pixel) captured by a camera.	Instructor
UndoPixelNormalization.m	to reverse the pixel normalization process by converting normalized lens coordinates to screen coordinates based on the camera parameters and the active video region.	Instructor

In our study, we used the data and template to generate our function to track and detect the balls in a sort of method. Details of the methodology of the project work are given below.

2.2. ROI Masking

Define an ROI by drawing a polygon area in an image frame by using ‘`impoly`’ algorithm within an area where the balls are moving (fig. 2). Create a masked area within the ROI area by applying ‘`CreateMask`’ algorithm. Then the masked areas for both cameras were saved in the ‘project code’ existing folder for further processing.

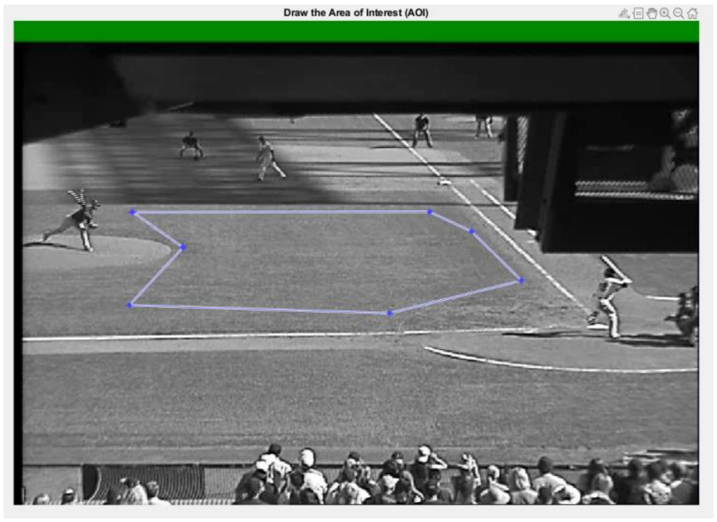


Figure 2: Drawing a region of interest for defining the ball movement area.

2.3. Thresholding

There are a lot of thresholding methods, however, in our study, we manually figure out the threshold for the ball. To fix the threshold we used 'imhist()' to get the threshold value within the ROI masked area and then finalized the threshold value for the masked pitch which is 220 (fig. 3).

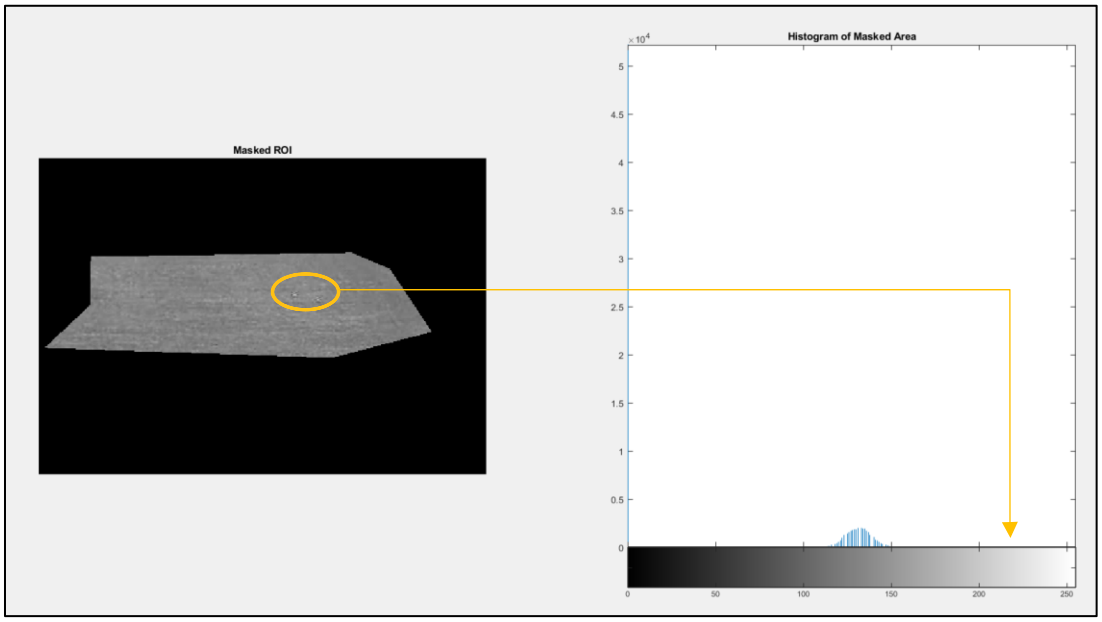


Figure 3: Masked ROI (left) and the histogram of pixel values (right) within the masked area

2.4. Filtering and Object Detecting

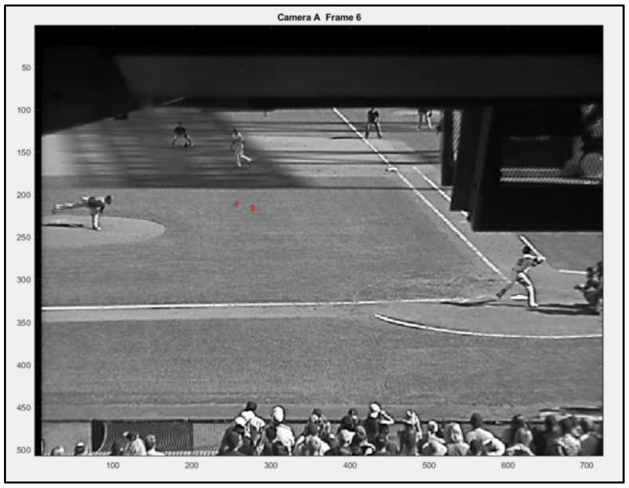
In MATLAB, there are plenty of filtering function areas available, however, we used thresholding approaches. In the above section, it was mentioned the thresholding value that is used in this stage as a loop method by using the '`for`' algorithm for both cameras (fig. 4). The details code of the iteration of the function is given below for Camera A,

```
for frameIdx = 1:size(dclipA, 3)
    frameA = dclipA(:, :, frameIdx);
    maskedFrameA = frameA .* maskA;
    binaryFrameA = maskedFrameA > threshold;
    ccA = bwconncomp(binaryFrameA);
    statsA = regionprops(ccA, 'Centroid');

    for ballIdx = 1:length(statsA)
        ballA(end + 1).x = statsA(ballIdx).Centroid(1);
        ballA(end).y = statsA(ballIdx).Centroid(2);
        ballA(end).frame = frameIdx;
        ballA(end).image = 1; % Image number in frame A

    end
```

```
        end
```



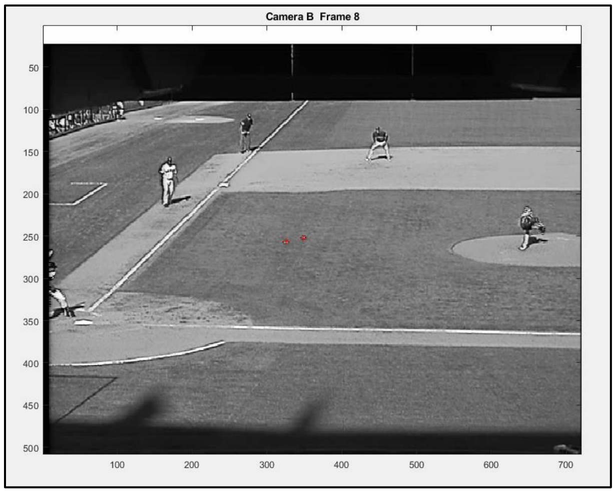


Figure 4: Detected balls for camera A (frame 6) and camera B (frame 8) as the red color symbol

Here is the step-by-step explanation for the above-mentioned functional code for detecting balls from Camera A,

Step 1: Applied the '`for`' algorithm that iterates through each frame captured by Camera A.

Step 2: Retrieves the current frame (image) from the camera A video sequence.

Step 3: Apply a mask (maskA) to the frame to isolate the regions of interest (ROI) where the ball is expected to be present.

Step 4: Converts the masked frame into a binary image based on a threshold value to distinguish the ball from the background.

Step 5: Performs connected component analysis on the binary image to identify individual objects (balls) and group pixels belonging to the same object together.

Step 6: Calculate the centroid coordinates for each detected ball using the region properties obtained from the connected component analysis.

Step 7: Record the centroid coordinates, frame index, and image number in a data structure (ballA) for each detected ball in camera A.

Step 8: Finally, label each detected ball with the image number (in this case, 1, indicating camera A).

The same approach was applied to Camera B. After the generation of the function '`detectAndTrackBalls`', it was saved in the existing folder that was used for further progression in the '`pitchfx_soup2nuts`' file. All supported files including the ball detect function ('`detectAndTrackBalls`') files were added to the attached folder of this study.

2.5. Run and Display Output

In this study ball detection and tracking function were generated by the following structure that was mentioned in the project instructions (project 1). The structure code for both cameras is as given below,

```
ballA = struct('x', [], 'y', [], 'frame', [], 'image', []);
ballB = struct('x', [], 'y', [], 'frame', [], 'image', []);
```

The output was displayed as frame-wise ball numbers (Table 2) including the ball x and y coordinates were generated in this study. On the other hand, the ball positions with ball heights were displayed in 3D graphical format.
Furthermore, the ball difference was also calculated in the study with the .csv data that was provided by PitchFX with time. Additionally, the SVD solution was obtained through the Singular Value Decomposition (SVD) matrix. Finally, the PitchFX solution was also calculated at the end of the study.

2.6. Alternative detection methods:

Different methods for image processing object detection include traditional image processing methods and modern deep learning networks. Traditional methods such as Viola-Jones, SIFT, and histogram of oriented gradients do not require historical data for training and are unsupervised [8]. These methods can be implemented using popular image processing tools like OpenCV [9] .On the other hand, modern deep learning networks like CNN, RCNN, YOLO, ResNet, RetinaNet, and MANet are supervised and efficient for object detection. These deep learning networks utilize convolutional neural networks (CNN) and have shown high accuracy in object detection tasks [3] . They can extract features of objects by sliding a window across the object instances in the given image sequence [10]. These methods have been widely used in various applications, including autonomous vehicles and surveillance systems.

3. Results and Discussion

The developed ball detection algorithm has successfully addressed the challenge posed by the interleaved images from two cameras in the PitchFX system. By effectively distinguishing baseballs from background elements based on brightness and shape, the algorithm accurately identifies and tracks the movement of each ball in recorded video files. This achievement signifies the successful resolution of the problem, enabling comprehensive analysis of pitch data and providing valuable insights for performance evaluation and strategic decision-making in Major League Baseball (MLB).

Table 2: Detected balls count in each frame.

Frames	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
Camera A	0	0	0	1	2	3	2	2	2	2	3	2	0	0	0	2	2	3	3	2	3
Camera B	0	0	0	1	0	1	1	2	2	2	4	1	0	0	0	0	1	1	0	1	0

There are 21 frames captured from both cameras while camera A detected better than Camera B. There are several tracked balls varied for both cameras with few numbers of frame did not detect any balls. The reason behind no balls due to ROI indicates the outer area of ROI balls weren't detected in this study (fig. 3). Among the 21 frames in camera A detected 4 to 12 no. frames the baller throwing the ball and 16 to 21 no. frames the batsman heated the balls respectively (table 2). On the other hand, camera B also detected from 4 to 12 no. frames for balling and 17 to 20 no. for heating respectively. The significant thing is that for camera B that is detected balls randomly that sequentially due to the camera resolution.

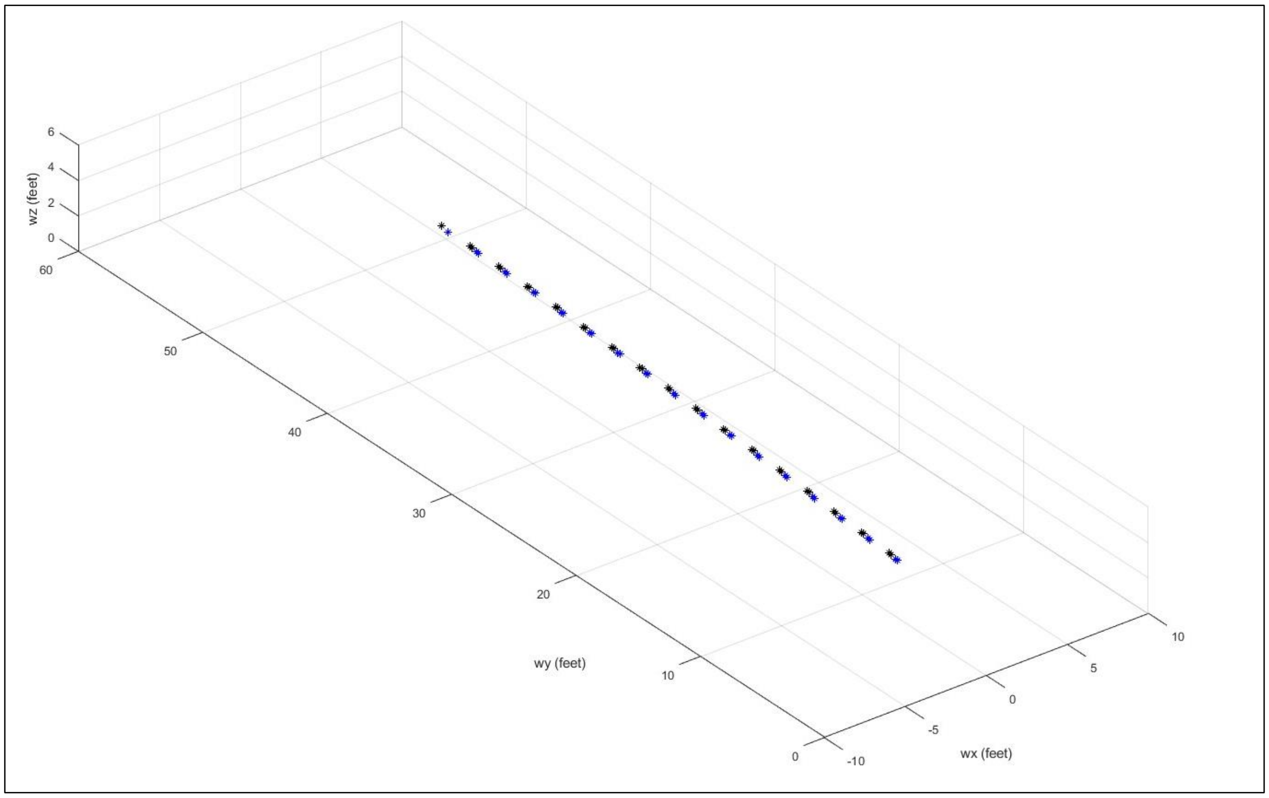


Figure 5: Detected baseball position in wx, wy and wz (feets) axis

However, the ball movements were monitored in Camera A, revealing consistent patterns with slight shifts over time. The analysis depicted a fluid sequence with minimal changes in ball positions across frames. In contrast, Camera B exhibited irregular detection, occasionally diverging from expected positions due to possible obstructions or tracking issues. Overall, Camera A demonstrated better tracking stability than Camera B, possibly reflecting environmental or tracking accuracy discrepancies. Enhancing Camera B's tracking performance may require further attention to address these challenges.

The comparison between the results from the Singular Value Decomposition (SVD) solution and the PitchFX solution offers valuable insights into the initial state of motion for baseball. Even while the initial location coordinate discrepancies are minimal—they vary by from 0.012342 to 0.595969 units—they may nevertheless have an impact on later investigations, particularly those that use spatial data. The disparities in the initial velocity components also point to minor changes in the projected rates of change of the position of the baseball along each axis at the beginning of its motion, ranging from 1.105946 to -0.744396 units per time interval. However, the most significant disparities emerge in the initial acceleration components, with variations ranging from 3.764442 to 6.269656 units per time interval. These discrepancies suggest significant changes in the calculated rates of change of baseball's velocity along each axis, which may have an impact on investigations pertaining to phenomena that are dependent on acceleration. In spite of these variations, the starting velocity vector's magnitude is largely constant, fluctuating only by about 0.282262 units throughout each time interval. Even though these discrepancies might not seem like much, they might have an impact on later assessments, especially in fields like projectile motion science or sports statistics.

However, the established ball detection algorithm successfully addresses the challenge of analyzing enclosed images from two cameras in the PitchFX system in MATLAB. In terms of tracking stability, Camera A outperformed Camera B, maybe because of variations in the surrounding conditions or tracking precision. The algorithm accurately identifies and tracks ball movements, enabling comprehensive analysis of pitch data for Major League Baseball. On the other hand, Camera B showed inconsistent detection, most likely as a result of obstacles or tracking problems. With regard to the location, velocity, and acceleration components of baseball's initial state of motion, the Singular Value Decomposition (SVD) solution offers comprehensive insights. Comprehending baseball's initial kinematics is essential for predicting trajectories and conducting additional analysis. The study does point up some difficulties with regard to recognizing balls outside the ROI, which could affect full ball tracking. Camera B's tracking performance might be further improved to overcome these issues and raise overall data accuracy.

4. Conclusions

The study aimed to create a reliable ball recognition method in MATLAB for monitoring baseballs from two cameras by using PitchFX tools from Major League Baseball (MLB). A crucial component of computer vision, object detection is used in many different fields, such as continuous object tracking and pedestrian recognition. The methodology involved region of interest (ROI) masking, thresholding, and object filtering to effectively isolate and track baseball movement. The analyzed results revealed Camera A outdone Camera B in tracking stability, potentially due to environmental or tracking accuracy disparities, with Camera B exhibiting irregular detection, likely owing to obstacles or tracking issues. Even with these achievements, there are still issues, particularly with identifying balls outside the ROI, which affects entire ball trackingInsightful initial kinematic data was supplied via the Singular Value Decomposition (SVD) solution, which was essential for trajectory prediction and additional analysis. The results emphasized how important strong object detection techniques are for improving data quality and opening up new perspectives on baseball dynamics. The tracking and analysis capabilities of Camera B could be improved by working to overcome detection issues outside of the ROI. Alternative detection techniques, such as cutting-edge deep learning networks, may be investigated in future studies to supplement current algorithms and enhance overall performance. All things considered, the study made a significant addition to baseball dynamics by developing and successfully implementing a ball detection algorithm. This opened the door for improved performance assessment and strategic decision-making in Major League Baseball and other professional sports. Object detection algorithms will remain essential for developing data-driven insights and technical innovations in sports analytics and other fields through continuous innovation and improvement.

5. References:

[1] Chen, Zhuohui, et al. "A Scaling Factor Based Image Processing Strategy for Object Detection." *International Journal* 12.3 (2023).

[2] Chatrasi, Amar Lokesh Venkata Siva Sai, et al. "Pedestrian and Object Detection using Image Processing by YOLOv3 and YOLOv2." *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2023.

[3] Ma, Wenting, et al. "A Video Image Processing Method for Continuous Object Detection." *2023 8th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 2023.

[4] Kim, Jae-Yeul, and Jong-Eun Ha. "Weakly Supervised Foreground Object Detection Network Using Background Model Image." *IEEE Access* 10 (2022): 105726-105733.

[5] Wang, Yue, et al. "How native background affects human performance in real-world visual object detection: an event-related potential study." *Frontiers in neuroscience* 15 (2021): 665084.

[6] Wang, Yu, and Zhiteng Wang. "Salient Object Detection based on Background Optimization." *Journal of Physics: Conference Series*. Vol. 2456. No. 1. IOP Publishing, 2023.

[7] Cheng, Jian, et al. "BackgroundNet: Small Dataset-Based Object Detection in Stationary Scenes." *Advances in Swarm Intelligence: 10th International Conference, ICSI 2019, Chiang Mai, Thailand, July 26–30, 2019, Proceedings, Part II 10*. Springer International Publishing, 2019.

[8] "Most watched sports leagues in the US 2023," Statista. Accessed: Jan. 31, 2024. [Online]. Available: https://www.statista.com/statistics/1430289/most-watched-sports-leagues-usa/

[9] Chavan, C., Hembade, S., Jadhav, G., Komalwad, P., Rawat, P. "Computer Vision Application Analysis based on Object Detection," *Indian Scientific Journal Of Research In Engineering And Management*, vol. 07, no. 04, 2023.

[10] Aggarwal, Akshima, Vaneet Kumar, and Ruchika Gupta. "Object Detection Based Approaches in Image Classification: A Brief Overview." *2023 IEEE Guwahati Subsection Conference (GCON)*. IEEE, 2023.

[11] Patkar, Uday Chandrakant, et al. "Object Detection using Machine Learning and Deep Learning." *International Journal of Intelligent Systems and Applications in Engineering* 12.1s (2024): 466-473.

6. Appendix

1. Ball detects and Tracking Function

```
function [ballA, ballB] = detectAndTrackBalls(dclipA, dclipB)
    % Load ROI masks from MATLAB files
    roiMaskA = load("MaskRoi_A.mat");  % Adjust the filename as needed
    roiMaskB = load("MaskRoi_B.mat");  % Adjust the filename as needed

    % Access the mask variables from the loaded data
    maskA = roiMaskA.mask;  % Access the mask field in roiMaskA
    maskB = roiMaskB.mask;  % Access the mask field in roiMaskB

    % Define threshold value for masking
    threshold = 220;

    % Initialize arrays to store detected ball information
    ballA = struct('x', [], 'y', [], 'frame', [], 'image', []);
    ballB = struct('x', [], 'y', [], 'frame', [], 'image', []);

    % Loop through frames of camera A
    for frameIdx = 1:size(dclipA, 3)
        % Extract current frame from camera A
        frameA = dclipA(:, :, frameIdx);

        % Apply masking to detect ball coordinates using ROI mask for
camera A
        maskedFrameA = frameA .* maskA;

        % Detect ball as objects within masked frame for camera A
        binaryFrameA = maskedFrameA > threshold;
        ccA = bwconncomp(binaryFrameA);
        statsA = regionprops(ccA, 'Centroid');

        % Extract centroid coordinates for each detected ball in
camera A
        for ballIdx = 1:length(statsA)
            ballA(end + 1).x = statsA(ballIdx).Centroid(1);
            ballA(end).y = statsA(ballIdx).Centroid(2);
            ballA(end).frame = frameIdx;
            ballA(end).image = 1; % Image number in frame A

        end
    end

    % Loop through frames of camera B
    for frameIdx = 1:size(dclipB, 3)
        % Extract current frame from camera B
        frameB = dclipB(:, :, frameIdx);

        % Apply masking to detect ball coordinates using ROI mask for
camera B
        maskedFrameB = frameB .* maskB;

        % Detect ball as objects within masked frame for camera B
        binaryFrameB = maskedFrameB > threshold;
        ccB = bwconncomp(binaryFrameB);
        statsB = regionprops(ccB, 'Centroid');

        % Extract centroid coordinates for each detected ball in
camera B
        for ballIdx = 1:length(statsB)
            ballB(end + 1).x = statsB(ballIdx).Centroid(1);
            ballB(end).y = statsB(ballIdx).Centroid(2);
            ballB(end).frame = frameIdx;
            ballB(end).image = 2; % Image number in frame B
        end
    end

    % Initialize an array to store the count of detected balls in each
frame
    ballCountA = zeros(1, size(dclipA, 3)); % For camera A
    ballCountB = zeros(1, size(dclipB, 3)); % For camera B

    % Loop through frames of camera A
    for frameIdx = 1:size(dclipA, 3)
        % Count the number of detected balls in the current frame
        ballCountA(frameIdx) = length(ballA([ballA.frame] ==
frameIdx));
    end

    % Loop through frames of camera B
    for frameIdx = 1:size(dclipB, 3)
        % Count the number of detected balls in the current frame
        ballCountB(frameIdx) = length(ballB([ballB.frame] ==
frameIdx));
    end

    % Display the count of detected balls in each frame for camera A
    disp("Detected balls count in each frame for camera A: 21
Frames");
```

```
        disp(ballCountA);

    % Display the count of detected balls in each frame for camera B
    disp("Detected balls count in each frame for camera B: 21 
Frames");
    disp(ballCountB);
end
```

2. Output in Command Window (pitchfx_soup2nuts)

```
>> pitchfx_soup2nuts
clipA: clipA.asf has 21 frames
clipB: clipB.asf has 21 frames
Detected balls count in each frame for camera A: 21 Frames
   0   0   0   1   2   3   2   2   2   2   3   2   0   0   0   2   2   3   3   2   3

Detected balls count in each frame for camera B: 21 Frames
   0   0   0   1   0   1   1   1   2   2   2   1   0   0   0   0   1   1   0   1   0

**** Camera A
** ball  1: (          ) - no matching PFX ball
** ball  2 : (193.0 201.0) - PFX ball  3 : (192.0,199.0)
**    difference: (-1.000,-2.000) time = 21520.785211
** ball  3 : (213.5 204.0) - PFX ball  4 : (212.5,203.0)
**    difference: (-1.000,-1.000) time = 21520.801894
** ball  4 : (234.5 207.0) - PFX ball  5 : (233.2,206.8)
**    difference: (-1.300,-0.200) time = 21520.818577
** ball  5 : (256.0 212.0) - PFX ball  6 : (255.0,211.0)
**    difference: (-1.000,-1.000) time = 21520.835261
** ball  6 : (277.0 215.0) - PFX ball  7 : (276.0,215.0)
**    difference: (-1.000,0.000) time = 21520.851944
** ball  7 : (277.0 217.0) - PFX ball  7 : (276.0,215.0)
**    difference: (-1.000,-2.000) time = 21520.851944
** ball  8 : (299.0 220.0) - PFX ball  8 : (298.0,220.0)
**    difference: (-1.000,0.000) time = 21520.868627
** ball  9 : (321.0 225.0) - PFX ball  9 : (320.0,224.5)
**    difference: (-1.000,-0.500) time = 21520.885311
** ball 10 : (343.5 230.0) - PFX ball 10 : (342.0,229.0)
**    difference: (-1.500,-1.000) time = 21520.901994
** ball 11 : (365.0 235.0) - PFX ball 11 : (363.8,234.2)
**    difference: (-1.200,-0.800) time = 21520.918677
** ball 12 : (388.5 240.0) - PFX ball 12 : (387.0,240.0)
**    difference: (-1.500,0.000) time = 21520.935360
** ball 13 : (411.5 245.0) - PFX ball 13 : (410.3,244.3)
**    difference: (-1.167,-0.667) time = 21520.952044
** ball 14 : (434.5 250.0) - PFX ball 14 : (433.2,250.5)
**    difference: (-1.250,0.500) time = 21520.968727
** ball 15 : (458.0 255.0) - PFX ball 15 : (457.2,255.8)
**    difference: (-0.800,0.800) time = 21520.985410
** ball 16 : (481.0 260.0) - PFX ball 16 : (481.0,261.0)
**    difference: (0.000,1.000) time = 21521.002094
** ball 17 : (482.0 262.0) - PFX ball 16 : (481.0,261.0)
**    difference: (-1.000,-1.000) time = 21521.002094
** ball 18 : (506.5 267.0) - PFX ball 17 : (504.8,266.2)
**    difference: (-1.700,-0.800) time = 21521.018777
** ball 19 : (530.5 272.0) - PFX ball 18 : (529.7,272.9)
**    difference: (-0.786,0.857) time = 21521.035460
** ball 20 : (531.0 274.0) - PFX ball 18 : (529.7,272.9)
**    difference: (-1.286,-1.143) time = 21521.035460
** ball 21: (470.0 301.0) - no matching PFX ball
** ball 22: (490.0 306.0) - no matching PFX ball
** ball 23: (429.0 293.0) - no matching PFX ball
** ball 24: (449.0 296.0) - no matching PFX ball
** ball 25: (389.0 283.0) - no matching PFX ball
** ball 26: (389.5 285.0) - no matching PFX ball
** ball 27: (409.0 288.0) - no matching PFX ball
** ball 28: (350.0 277.0) - no matching PFX ball
** ball 29: (369.0 280.0) - no matching PFX ball
** ball 30: (370.0 282.0) - no matching PFX ball
** ball 31: (311.5 271.0) - no matching PFX ball
** ball 32: (330.5 274.0) - no matching PFX ball
** ball 33: (273.0 265.0) - no matching PFX ball
** ball 34: (273.0 267.0) - no matching PFX ball
** ball 35: (292.5 268.0) - no matching PFX ball
**** Camera B
** ball  1: (           ) - no matching PFX ball
** ball  2 : (503.0 217.0) - PFX ball  5 : (501.2,217.8)
**   difference: (-1.800,0.800), time = 21520.784048
** ball  3 : (416.0 237.0) - PFX ball  9 : (415.0,236.0)
**   difference: (-1.000,-1.000), time = 21520.850781
** ball  4 : (394.0 242.0) - PFX ball 10 : (392.5,241.3)
**   difference: (-1.500,-0.667), time = 21520.867465
** ball  5 : (349.0 252.0) - PFX ball 12 : (347.5,251.0)
**   difference: (-1.500,-1.000), time = 21520.900831
** ball  6 : (279.0 269.0) - PFX ball 15 : (277.5,267.3)
**   difference: (-1.500,-1.667), time = 21520.950881
** ball  7 : (302.0 264.0) - PFX ball 14 : (301.0,262.0)
**   difference: (-1.000,-2.000), time = 21520.934198
** ball  8 : (231.0 281.0) - PFX ball 17 : (229.4,279.6)
**   difference: (-1.600,-1.400), time = 21520.984247
** ball  9 : (255.0 274.0) - PFX ball 16 : (253.0,274.0)
**   difference: (-2.000,0.000), time = 21520.967564
** ball 10 : (181.0 293.0) - PFX ball 19 : (180.0,293.0)
**   difference: (-1.000,0.000), time = 21521.017614
** ball 11 : (206.0 286.0) - PFX ball 18 : (204.7,286.1)
**   difference: (-1.286,0.143), time = 21521.000930
** ball 12 : (156.0 300.0) - PFX ball 20 : (154.0,300.0)
**   difference: (-2.000,0.000), time = 21521.034297
** ball 13 : (152.0 299.0) - PFX ball 20 : (154.0,300.0)
**   difference: (2.000,1.000), time = 21521.034297
** ball 14: (169.0 287.0) - no matching PFX ball
** ball 15: (201.0 263.0) - no matching PFX ball
*******  SVD solution *******
initial position x0 = -0.967767 y0 = 50.566572 z0 = 6.424722
initial velocity vx0 = 5.140611 vy0 = -133.452946 vz0 = -7.676299
initial acceleration ax = -4.065639 ay = 22.124034 az = -19.977265
initial speed = 91.208392
*******  PitchFX solution *******
initial position x0 = -0.940000 y0 = 50.000000 z0 = 6.360000
initial velocity vx0 = 3.360000 vy0 = -134.360000 vz0 = -6.610000
initial acceleration ax = 5.440000 ay = 27.900000 az = -26.940000
initial speed = 91.750000
```