# Management Center Innsbruck

## Department of Technology & Life Sciences

### Master's program Mechatronics & Smart Technologies

## Project report

**composed as part of the course**
**Digital Image Processing (MECH-M-1-SEA-DBV-ILV)**

**about**

# Ball Tracking Software

**from**

**Liam Nolan, Konstantin Wölfle and Leonhard Güntner**

| | |
|---|---|
| Study program | Master's program Mechatronics & Smart Technologies |
| Year | MA-MECH-23-VZ |
| Course | Digital Image Processing (MECH-M-1-SEA-DBV-ILV) |
| Name of lecturer | Daniel T. McGuinnes, Phd |
| Submission deadline | February 06, 2024 |

April 3, 2024

# Contents

# Chapter 1

# Introduction

In the modern age, digital image processing is a topic of particular interest, especially in sports such as soccer, cricket, etc. Analyzing the trajectory of a ball is necessary to improve training, game analysis, and understanding of physics. The appearance of the ball in a video can change unpredictably from frame to frame, recognizing and tracking it in front of an ever-changing background is a complicated task.[1]

This project aims to determine the position, velocity, acceleration, and trajectory of a thrown object in an imported Video. This report aims to document the processes involved in programming the ball-tracking software. The methods, results, and a brief conclusion are presented.

# Chapter 2

# Methods

In the following chapter, the methods used to accurately detect a flying ball are listed and explained. The whole process is a series of calculations performed on each frame of the video where a ball can be detected, repeating until the end of the video is reached. The software was developed exclusively in Python with the OpenCV, and Numpy libraries being used extensively.

## 2.1 Video Import/Display

Video processing in Python using OpenCV is similar to single-image processing but it is slightly more complex in implementation. First, a capture element is opened with $cap = cv2.VideoCapture()$. The program then enters a while loop where each iteration the $cap.read()$ method will return a frame and a boolean variable ($ret$) that indicates whether a frame is present. If the $ret$ element of the method is false, the video has ended and the loop will break.

Each iteration the loop will perform processing on the frame element and then display this element using the $cv2.imshow()$ method forming a video.

## 2.2 Color Masking

Color masking allows the color of the ball to be separated from the background image to allow for tracking and processing. For this project the OpenCV method $cv2.inRange()$ was used. This method takes a frame, an upper color value threshold, and a lower color value threshold. Every pixel of the frame that is found to be within these values is turned white and every other pixel is turned black. This allows for a clear distinction of the object from the background for processing.

These color threshold values were determined experimentally using a modified Python script from the OpenCV documentation. The script was modified to utilize a still image from the imported video. The user can then adjust sliders on a GUI to determine an optimal threshold value. These values can then be passed into the script.

For the inRange method to be applied a conversion from BGR to HSV is necessary. The script works in HSV as it was found to be a more manageable format to adjust the specific color masking values. HSV separates the color information into Hue, Saturation, and Value. Hue represents the color itself by a value between 0 and 255. The intensity and brightness are covered by Saturation and Value. BGR instead represents the colors by using three different channels: blue, green, and red. Each channel is an 8-bit value ranging from 0 to 255. By default, OpenCV works in BGR colorscape.

After the color mask is applied small perturbations are sometimes seen in the mask. This is likely due to small inconsistencies in the background of the video. To remove these small perturbations, the mask is eroded and then dilated. The erosion reduces the mask size and any small artifacts will be removed. The mask dilation will then restore any remaining masked sections to their original size. This process can be seen in figure 2.1 In theory this should only leave the mask on the ball. After masking, the contours are applied.

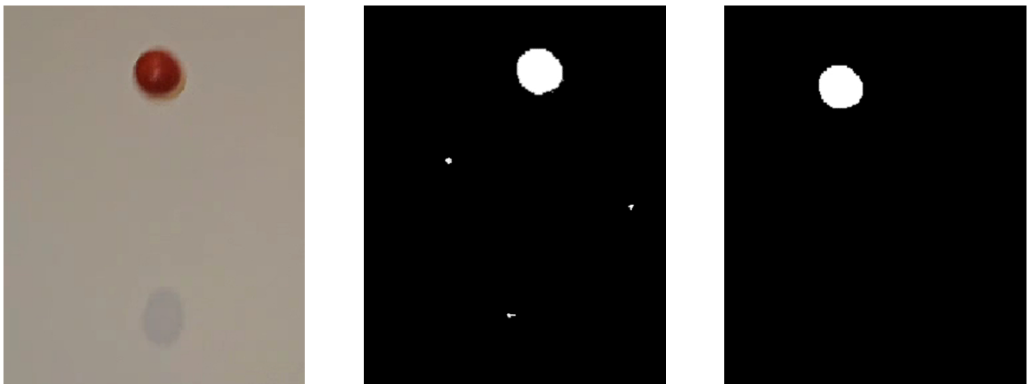
**(a)** Original image **(b)** Color mask **(c)** Erode and Dilate

**Figure 2.1.** Mask Erosion and Dilation

## 2.3 Contour Detection and Center Detection

This section covers the contour detection as well as the center detection of the thrown object. After the described color mask is applied it is necessary to determine the location of the object. The command $cv2.findcontours()$ is utilized to locate contours in the image. Because of the color masking previously done, there is a sharp contrast between the subject and the background, and the contours of the ball are consistently found. In some lighting conditions, a small spot can form on the ball leading to two contours being detected. To determine the actual outline of the ball, the area of the contours is examined, and the contour with the largest area is assumed to be the outline of the ball. For visualization, the ball contour is drawn onto the frame using the $cv2.drawcontours()$ method, which can be seen in figure 2.2.

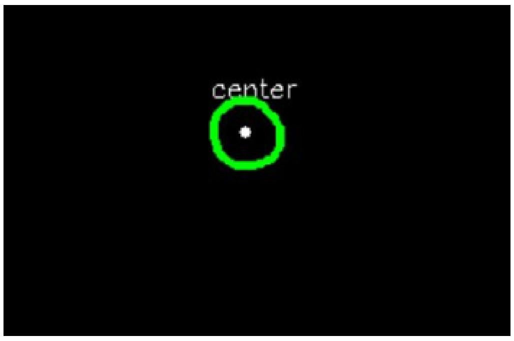

**Figure 2.2.** Ball Contour and Center

The center of the contour is crucial to track and predict the trajectory of the object. By using the $cv2.moments()$ command the moments of the image are found and the center can be calculated by the following equations 2.1 and 2.2.

$$C_x = \frac{M_{10}}{M_{00}} \tag{2.1}$$

$$C_y = \frac{M_{01}}{M_{00}} \tag{2.2}$$

For an accurate calculation, the OpenCV coordinate system is used. It is placed at the top left of the image. Therefore in figure 2.3 Y-Direction of the camera coordinates is facing downwards. Since the reached height of the thrown object is considered as the maximum value, the Y-Direction in the following calculation is the other way around. In parabolic motion, the positive Y-Direction is assumed to be upwards. In conclusion, upwards leads to positive velocity.

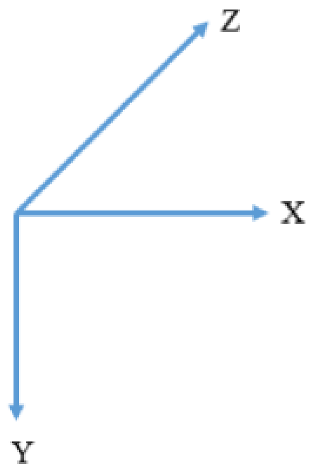

**Figure 2.3.** OpenCV Camera Coordinates

## 2.4 Motion Detection and Trajectory Prediction

After successfully detecting the ball's contour and determining its centroid within two successive frames, the ball's velocity can be calculated.
Using $fps = cap.get(cv.CAP\_PROP\_FPS)$, the video frame rate is detected. Afterwards, the time in between two frames is calculated via $\Delta t = \frac{1}{fps}$. As the $C_x$ and $C_y$ position of the contour changes in every frame, a velocity is determined by the equations 2.3.

$$v_x = \frac{C_{x1} - C_{x1}}{\Delta t} \qquad v_y = \frac{C_{y1} - C_{y0}}{\Delta t} \tag{2.3}$$

Since $C_x$ and $C_y$ are calculated in pixels, the velocities obtained are now given in $\frac{pixels}{s}$. For converting the velocities to $\frac{m}{s}$, a scaling factor $K$ has to be determined. Depending on the camera distance to the ball, $K$ takes on different values and can therefore not be hard-coded, if the algorithm is to work for different video clips. For determining the correct factor, a third frame is needed. Knowing that gravity is the only force acting on the ball, the difference in two subsequent $v_y$ velocities is used to calculate $K$. The formula 2.4 shows the relationship.

$$\frac{v_{y2} - v_{y1}}{\Delta t} = K \cdot g \quad \text{with} \quad g = 9.81 \tag{2.4}$$

Now the previously calculated velocities are converted to the correct units.

At this point, the ball's remaining trajectory is predicted. Utilizing the standard formulas for parabolic motion

$$x(t) = x_0 + v_x \cdot t$$

$$y(t) = y_0 + v_y \cdot t - \frac{1}{2 \cdot K} \cdot g \cdot t^2$$

the balls' estimated x- and y-position are calculated for several time steps until the ball leaves the video frame. The result is shown in figure 2.4. The actual parabolic movement is the blue line, while the calculation is represented by the green line.

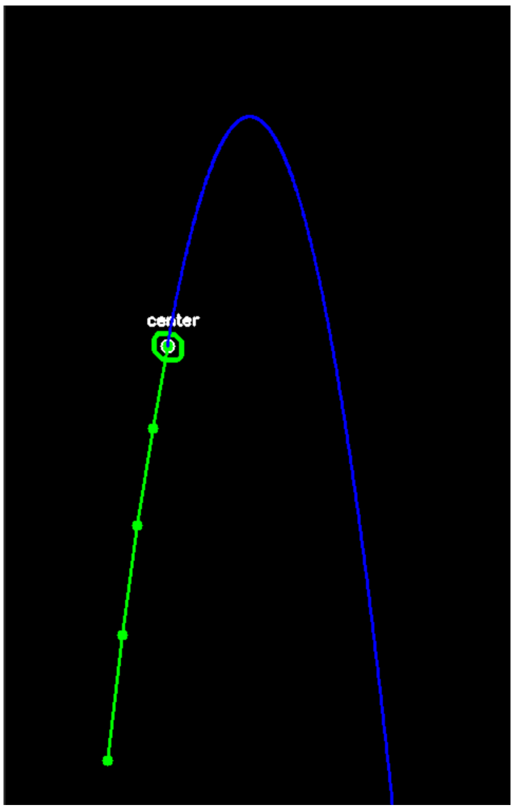

**Figure 2.4.** Predicted trajectory

Many tests have shown, that the accuracy of the prediction depends strongly on the scaling factor $K$. Calculating a value that is too low leads to an overestimation of the influence of gravity and thus to a maximum of the curve that is too low; a value that is too high results in a trajectory with an increased maximum. Also, the last value of $v_x$ is of great importance. A value that is too low results in a trajectory that is too narrow, and a value that is too high results in a trajectory that is too wide. In general, errors due to masking lead to errors in the detected contour, resulting in incorrect centroids, which ultimately leads to an error in the velocities, scaling factor, and predicted trajectory.

# Chapter 3

# Results

After processing the image and trajectory calculation, it is found that the tracking and prediction algorithms are both relatively accurate and robust. In figure 3.1 the predicted ball trajectory can be seen in blue and the actual trajectory is shown in green. As seen there is very little deviation between the predicted and actual path.

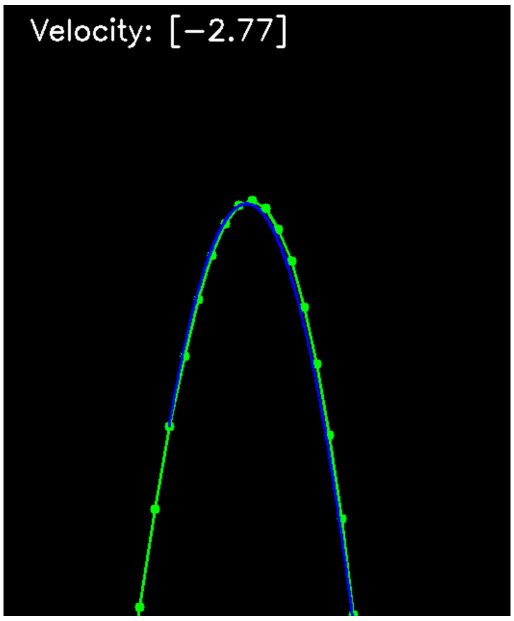

**Figure 3.1.** Predicted vs. Actual Trajectory

This small deviation could be a result of any number of factors including contour warping creating inconsistent physics, air resistance affecting the trajectory, or the camera not being fully planar. Figure 3.2 shows the warped shape of the ball during the video; detecting an accurate contour poses difficulties.

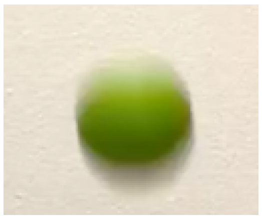

**Figure 3.2.** Deformed shape of the ball

In the following, possible improvements to the methods are shown. The described contour detection in chapter 2.3 can be improved. A preparation of the image to reduce the noise can be added. The Gaussian Blur Filter reduces the high-frequency noise and smoothness of the image. The Canny edge detector further calculates the Gradient and detects the edges in the image[2] in this way. Another challenge of the project is the constantly changing background. Therefore a possible improvement is to interchange the detection order. By detecting the shape of the object first the method is independent of the color. Moreover, multiple bodies can be identified. Due to the changing lighting conditions during the motion of the ball, both color and contour detection become difficult. The image is blurred, gray shades are visible and clear edges are impossible to recognize. By increasing the frame rate the images get sharper and the edges are visible more clearly. One option is to record with a high-speed camera for more accurate images. This ensures the recognition of equally shaped contours which reduces errors in velocity estimation and trajectory prediction. Another possibility is the usage of studio lighting to reduce the objects' shadows and improve the contrast to the background. Additionally, the motion of the ball is not always perfectly planar to the camera which can introduce errors.

# Chapter 4

# Conclusion

In summary, the ball detection as well as the motion calculation is successfully implemented. The method of color masking and using contour detection on the mask proved to be an effective way to track an object in motion and the trajectory prediction algorithm is working. While the predicted movement of the ball is slightly off, the total error is quite minimal and the results are satisfactory. In the future, the project could be improved upon by better masking and color detection. Improvements in this region would allow depth detection of the object and could accommodate varying lighting conditions and non-planar camera positioning.

# Bibliography

[1] P. R. Kamble, A. G. Keskar, and K. M. Bhurchandi, “Ball tracking in sports: a survey,” *The Artificial intelligence review*, vol. 52, no. 3, pp. 1655–1705, 2019.

[2] D. T. McGuinnes, “Digital Image Processing, version $\Psi$.2,” Reader, Management Center Innsbruck. 2023.

# List of Figures