CSE464 FINAL PROJECT REPORT

MULTIPLE BALL DETECTION AND TRACKING

SÜLEYMAN KORAMAZ
1901042615

ABSTRACT

Ball tracking is a vital aspect in various fields, including sports analysis, robotics, and interactive systems. The ability to accurately locate and track a ball in real-time opens possibilities for enhancing user experiences and enabling intelligent automation. This project focuses on real-time ball tracking using computer vision techniques. The system employs image processing algorithms to detect and track a ball within a video stream captured by a camera. The implemented solution utilizes color-based segmentation, contour analysis, and object tracking for robust and efficient ball localization.

CONTENTS AND FIGURES

1-INTRODUCTION

In this project, we present a computer vision-based solution for ball tracking, leveraging the OpenCV library and Python programming.

The primary goal of the project is to design an efficient and adaptable system capable of detecting and tracking a ball within a video stream. The system operates in two main modes: detection and tracking. During the detection mode, the system identifies the ball in the video frame using color-based segmentation in the HSV color space. Subsequently, noise reduction techniques, such as morphological operations, are applied to refine the segmentation.

The contour analysis is employed to filter out false positives and identify circular objects resembling a ball. The selected regions of interest (ROIs) are then used to initialize a multi-object tracking system based on the CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability) tracker. This system efficiently tracks the ball through subsequent frames, providing robust and consistent tracking.

2- PROJECT DESIGN

2.1- OBJECTIVE

Design and implement a real-time ball tracking system using computer vision techniques.

2.2- HARDWARE REQUIREMENTS

- Webcam or any camera device capable of capturing video feed.
- Computer with sufficient processing power for real-time image processing.

2.3- SOFTWARE REQUIREMENTS

- Python programming language.
- OpenCV library for image processing.
- NumPy for array manipulation

Figure 1: OpenCV

2.4- IMAGE PROCESSING TOOLS

GAUSSIAN BLUR

The Gaussian blur is a type of image-blurring filter that uses a Gaussian function (which is also used for the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. It is a widely used effect, typically to reduce image noise and reduce detail.

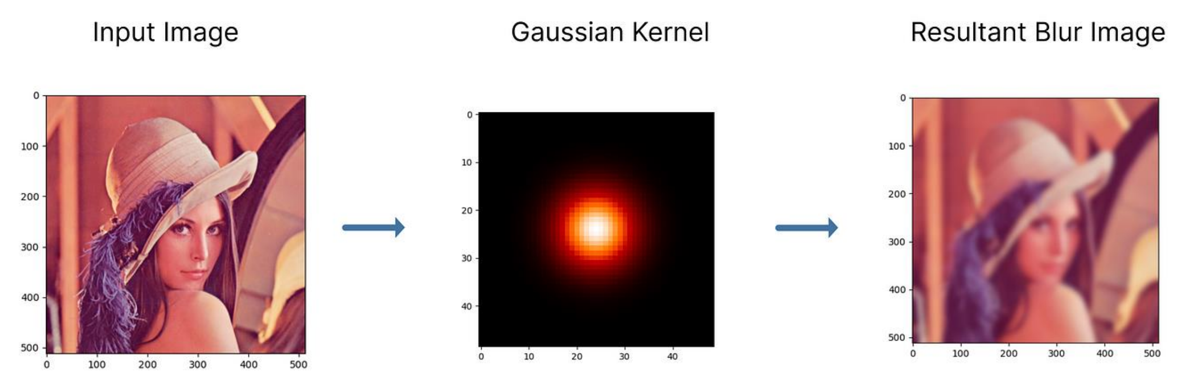

Figure 2: Gaussian Blur

HSV color space is the most suitable color space for color-based image segmentation. HSV is a cylindrical color model that remaps the RGB primary colors into dimensions that are easier for humans to understand. These dimensions are hue, saturation, and value.

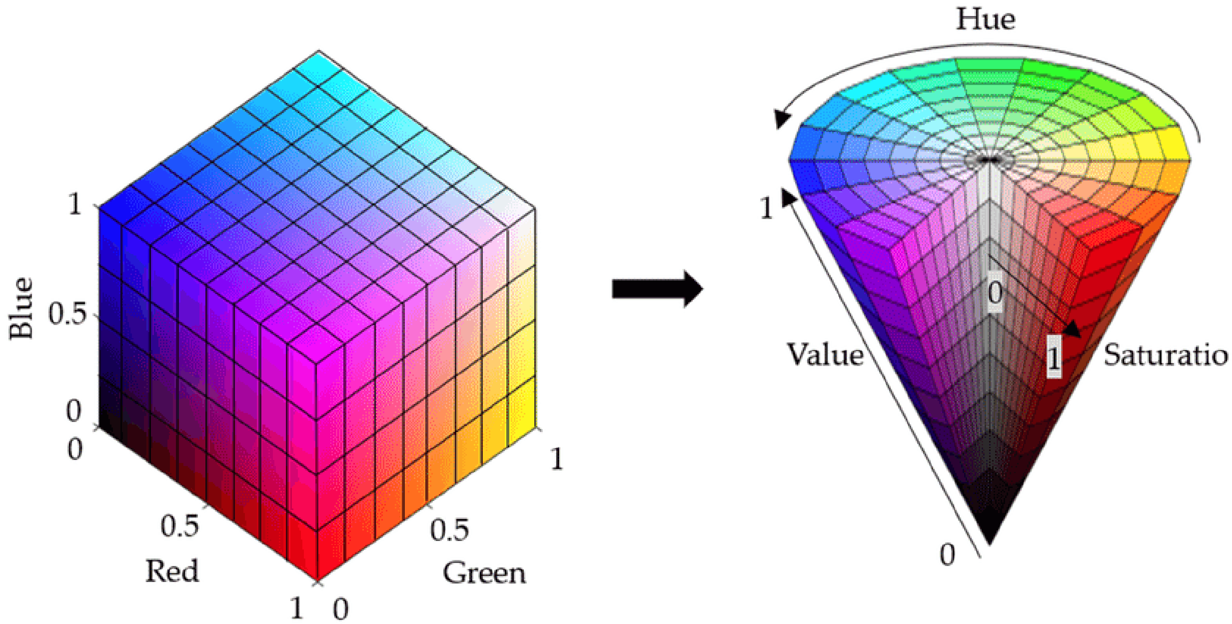

Figure 3: HSV Transformation

- Morphological opening is useful for removing small objects and noises.
- Morphological closing is useful for filling small holes.

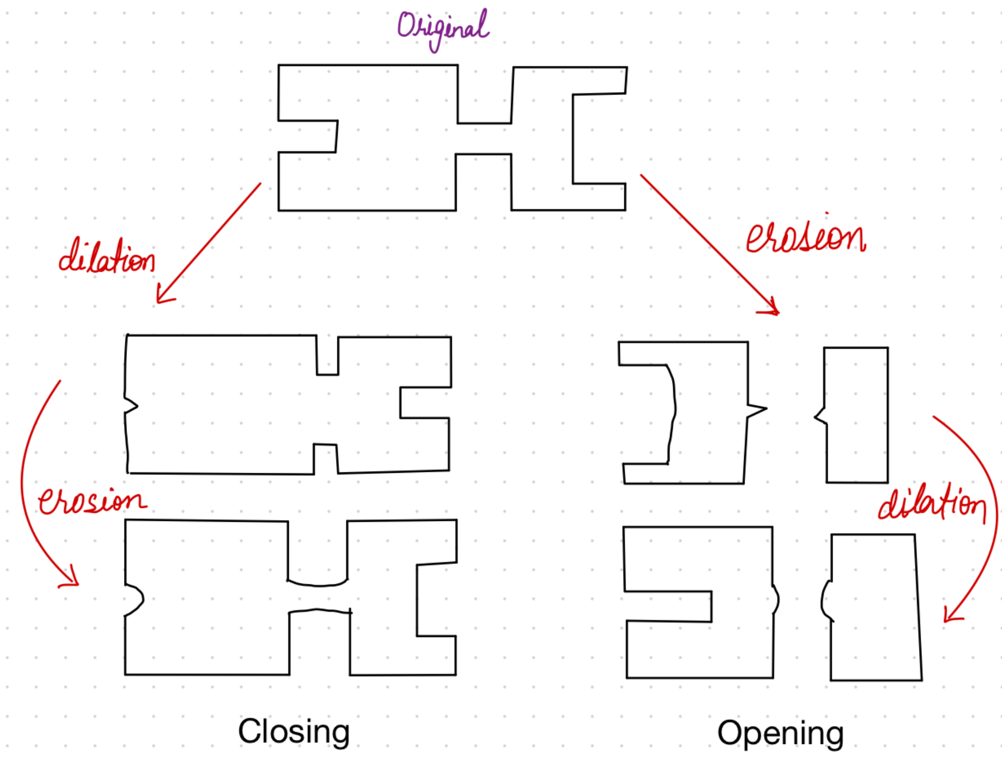

Figure 4: Morphological Opening and Closing

CONTOUR ANALYSIS

Image contouring is process of identifying structural outlines of objects in an image which in turn can help us identify shape of the object.

Consider an example below. On the left we have a hollow rectangle and solid circle. When we apply contouring with red color, what we get is image on the right. As you can see it has identified 3 objects, 2 rectangles and a circle and their borders have been drawn with red color.

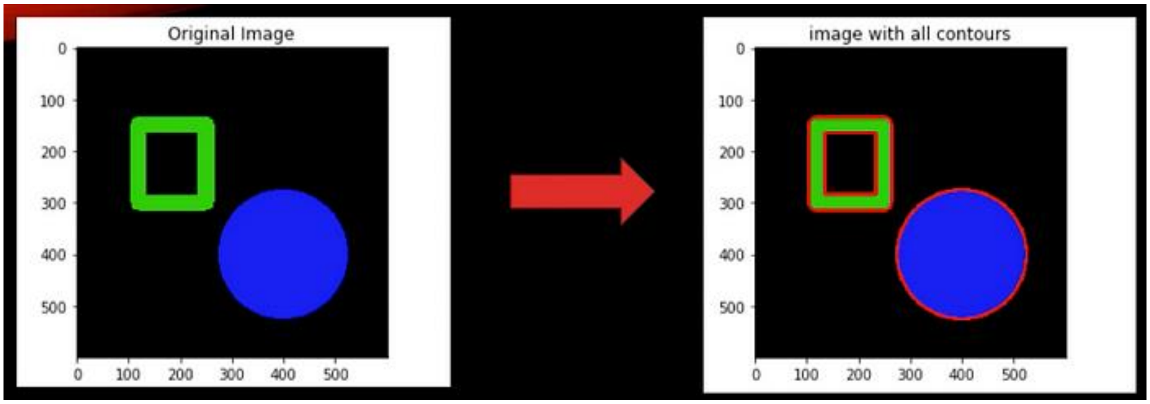

Figure 5: Finding Contours

CSRT TRACKER

In OpenCV, CSRT (Channel and Spatial Reliability Tracker) is one of the available object tracking algorithms. It is a robust tracker designed to handle challenging scenarios, including changes in scale, occlusion, and deformation of the tracked object.

- *Channel Reliability:* The CSRT algorithm utilizes color channel information to handle appearance changes caused by illumination variations. It models the target object using color features to enhance tracking accuracy.

- *Spatial Reliability:* Spatial reliability is employed to handle occlusions and non-rigid deformations. It considers both the spatial and temporal coherence of the object's motion, enabling robust tracking performance.

2.5- SYSTEM ARCHITECTURE

2.5.1- CAPTURE MODULE

- Utilize OpenCV to capture video frames from the camera.
- Ensure error handling for camera initialization.

```
cap = cv.VideoCapture(0)

if not cap.isOpened():
    print("Cannot open camera")
    exit()
```

2.5.2- COLOR SEGMENTATION MODULE

- Apply Gaussian blur to image.
- Convert frames to HSV color space.
- Apply a color threshold to identify the ball.

```
hsv_work = hsv_processing(frame, low_thres, high_thres)
```

2.5.3- NOISE REDUCTION MODULE

- Use morphological operations for noise reduction.
    - Morphological opening is used for removing small objects and noises.
    - Morphological closing is used for filling small holes.

```
noise_rmv = noise_processing(hsv_work)
```

2.5.4- OBJECT DETECTION MODULE

- Identify contours and filter out non-circular objects.
- Draw bounding boxes around detected balls.

```
ball_detection = findContours_processing(noise_rmv,ball_rois_list)
```

2.5.5- CIRCULARITY CONTROL MODULE

- Calculating the perimeter of the circumcircle of object.
- Calculating the area of the object.
- Calculating circularity based on this formula:

$$(\text{circularity}) = \frac{4\pi A}{l^2},$$

Figure 6: Circularity Formula

2.5.5- OBJECT TRACKING MODULE

- Utilize a multi-object tracker (CSRT tracker in this case) to track the detected balls.
- Update the tracker in subsequent frames.

```
multi_trackers = cv.legacy.MultiTracker_create()
    for ball_roi in ball_detection:
        multi_trackers.add(cv.legacy.TrackerCSRT_create(), frame, ball_roi)
```

2.5.6- SWITCHING MECHANISM

- Implement a mechanism to switch between detection and tracking modes.
- Adjust parameters for robustness, adaptability, and responsiveness.

```
if detection == 1:
    #... (detection mode)
else:
    #... (tracking mode)
```

2.5.7- USER INTERFACE MODULE

- Display the processed frames with bounding boxes and noise free mask.
- Provide an option to exit the application.

```
cv.imshow('Ball Tracking', frame)
cv.imshow('Mask', noise_rmv)
if cv.waitKey(1) == ord('q'):
    break
```

3- TEST

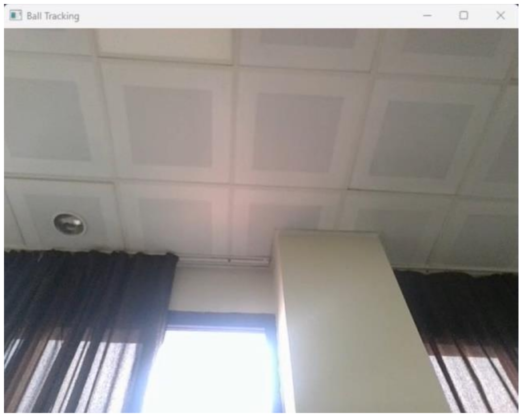

Figure 7: Testing – Background

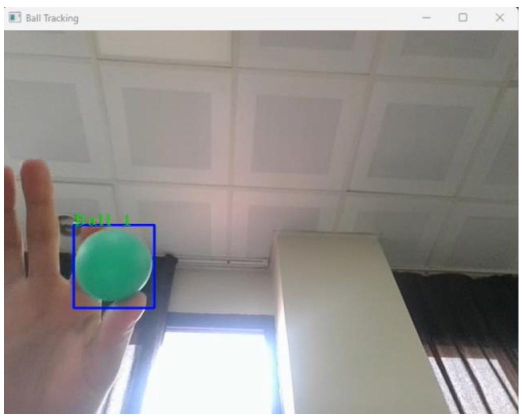

Figure 8: Testing - One Ball

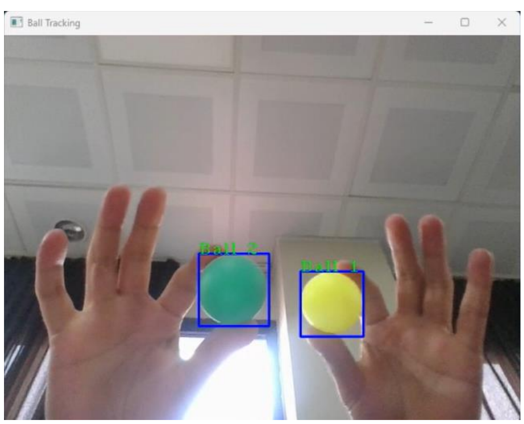

Figure 9: Testing - Two Balls

There is link to test video: (Reference 1)

CONCLUSION

This project design provides a structured approach to implementing a real-time ball tracking system using image processing. Adapt and extend the design based on specific project requirements and objectives.

REFERENCES

- Figure 1: https://datascientest.com/en/wp-content/uploads/sites/9/2023/11/opencv.webp
- Figure 2: https://miro.medium.com/v2/resize:fit:1400/1*G9Y4S7BLruiKiwGjjagINg.png
- Figure 3: https://www.researchgate.net/publication/359058781/figure/fig4/AS:1132594397749251@1647042955261/RGB-color-space-to-HSV-color-space-conversion.ppm
- Figure 4: https://miro.medium.com/v2/resize:fit:1400/1*YDeccRxRQsvSCx4eSmKqFw.png
- Figure 5: https://miro.medium.com/v2/resize:fit:1192/1*sNG4TP4R5A2AB0kKU_DbZw.png
- Figure 6: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRlRBnBLSCgWc-hnW6pGCoJQSvmsr394_ad8A&usqp=CAU
- Figure 7: Background image from testing video (Reference 1)
- Figure 8: One ball image from testing video (Reference 1)
- Figure 9: Two balls image from testing video (Reference 1)
- Reference 1: https://github.com/suleymankoramaz/464-Digital-Image-Processing/blob/main/final_project/ball_tracking_test.mp4
- Ball Tracking with OpenCV by Adrian Rosebrock on September 14, 2015 https://pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/

APPENDIX

```python
import numpy as np
import cv2 as cv
import time

# Open camera capture
cap = cv.VideoCapture(0)

# Check if the camera is opened successfully
if not cap.isOpened():
    print("Cannot open camera")
    exit()

# Define color codes
red = (0, 0, 255)
blue = (255, 0, 0)
green = (0, 255, 0)

# Flag for detection or tracking mode
detection = 1

# Parameters for tracking
max_tracking_frame = 20
count_tracking_frame = 0

# Frames per second and previous time initialization
fps = 60
prev = 0

# HSV color threshold values
low_thres = (29, 40, 40)
high_thres = (90, 255, 255)

# Function to draw bounding box and put text
def boundingBox_putText(input_frame, box_color, index, first_point, second_point):
    cv.rectangle(input_frame, first_point, second_point, box_color, 2)
    cv.putText(input_frame, 'Ball ' + str(index + 1), first_point, cv.FONT_HERSHEY_COMPLEX_SMALL, 1, green, 1)
```

```python
# Function for HSV color processing
def hsv_processing(input_frame, low_thres, high_thres):
    gauss_filter = cv.GaussianBlur(input_frame, (3, 3), 0)
    hsv = cv.cvtColor(gauss_filter, cv.COLOR_BGR2HSV)
    hsv_binary = cv.inRange(hsv, low_thres, high_thres)
    return hsv_binary

# Function for noise reduction
def noise_processing(input_frame):
    kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
    out = cv.morphologyEx(input_frame, cv.MORPH_OPEN, kernel, iterations=2)
    out = cv.morphologyEx(out, cv.MORPH_CLOSE, kernel, iterations=4)
    return out

# Function for contour processing and ball detection
def findContours_processing(input_frame, ball_rois_list):
    contour, _ = cv.findContours(input_frame, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    min_radius = 30
    max_radius = 90
    min_circularity = 0.75

    for index, cnt in enumerate(contour):
        _, radius = cv.minEnclosingCircle(cnt)
        radius = int(radius)
        if min_radius < radius < max_radius:
            circularity = 4 * np.pi * cv.contourArea(cnt) / (cv.arcLength(cnt, True) ** 2)
            if circularity > min_circularity:
                ball_rect = cv.boundingRect(cnt)
                first_point = (int(ball_rect[0]), int(ball_rect[1]))
                second_point = (int(ball_rect[0] + ball_rect[2]), int(ball_rect[1] + ball_rect[3]))

                ball_rois_list.append(ball_rect)
                boundingBox_putText(frame, red, index, first_point, second_point)
    return ball_rois_list

# Main Loop
while True:
    # Control frame rate
     timeElapsed = time.time() - prev
     if timeElapsed > 1. / fps:
        prev = time.time()
        # Read a frame from the camera
        ret, frame = cap.read()
        ball_rois_list = []
```

```python
        # Detection mode
        if detection == 1:
            hsv_work = hsv_processing(frame, low_thres, high_thres)
            noise_rmv = noise_processing(hsv_work)
            ball_detection = findContours_processing(noise_rmv, ball_rois_list)

            # Initialize multi-object tracker
            multi_trackers = cv.legacy.MultiTracker_create()
            for ball_roi in ball_detection:
                multi_trackers.add(cv.legacy.TrackerCSRT_create(), frame, ball_roi)
            detection = 0

        # Tracking mode
        else:
            if count_tracking_frame == max_tracking_frame:
                detection = 1
                count_tracking_frame = 0

            ret, objs = multi_trackers.update(frame)
            if ret:
                for index, obj in enumerate(objs):
                    # Check aspect ratio for potential false positives
                    if (float(obj[2]) / float(obj[3])) < 0.93 or (float(obj[2]) / float(obj[3])) > 1.36:
                        detection = 1
                    else:
                        first_point = (int(obj[0]), int(obj[1]))
                        second_point = (int(obj[0] + obj[2]), int(obj[1] + obj[3]))

                        boundingBox_putText(frame, blue, index, first_point, second_point)

            else:
                detection = 1
             count_tracking_frame += 1

        # Display frames
        cv.imshow('Ball Tracking', frame)
        cv.imshow('Mask', noise_rmv)

        # Exit condition
        if cv.waitKey(1) == ord('q'):
            break

# Release camera and close windows
cap.release()
cv.destroyAllWindows()
```