Internship Assignment

Computer Vision Analysis on Basketball Video

S.NO	CONTENT	P.NO
1	Abstract	2
2	Absorbed Perspectives	2
3	In-depth Video Insights	2
4	Basketball Dribble Counter	2
5	Trajectory of the Ball	10
6	Optical Flow of Objects	13
7	Automatically Detecting Outlines And Borders of Objects	16
8	Further Enhancement	22
9	Challenges In The Task	22
10	Conclusion	22

Abstract

The objective is to enhance the accuracy of analyzing a basketball player's dribbling drills in a video. Various methods, including trajectory analysis of the ball, optical flow of objects, and automatic object outline/border detection, are employed. Additionally, a focus on basketball dribble count is integrated to improve the precision of the analysis. The goal is to leverage these computer vision techniques to gain insights into the player's movements and enhance the overall understanding of the dribbling exercises in the video.

Absorbed Perspectives

I understand that the video shows a person dribbling a basketball inside a room. In the background, there's significant light reflection, and the camera moves from side to side. During the recording, there's shaking, and at times, a finger obstructs part of the view.

Techniques for In-depth Video Insights

The approach to this task involves implementing techniques such as dribble counting, analyzing the trajectory of the ball, examining the optical flow of objects, and automatically detecting outlines and borders of objects in the video. Certainly! Let's break down each technique with detailed explanations.

Basketball Dribble Counter

Method 1

- **Code with Explanation**
  - **Importing Libraries**

```
import cv2
from ultralytics import YOLO
import numpy as np
```

- **DribbleCounter Class Initialization**

```python
class DribbleCounter:
    def __init__(self, video_path):
        # Load the YOLO model for ball detection
        self.model = YOLO("C:/Users/Admin/Downloads/basketballModel.pt")

        # Open the video file
        self.cap = cv2.VideoCapture(video_path)

        # Initialize variables to store the previous position of the basketball
        self.prev_x_center = None
        self.prev_y_center = None
        self.prev_delta_y = None

        # Initialize the dribble counter
        self.dribble_count = 0

        # Threshold for the y-coordinate change to be considered as a dribble
        self.dribble_threshold = 10
```

  - **DribbleCounter:** A class encapsulating the functionality of the dribble counting application.

Run Method

```python
    def run(self):
        # Process frames from the video until the end
        while self.cap.isOpened():
            success, frame = self.cap.read()
            if success:
                self.process_frame(frame)

                if cv2.waitKey(1) & 0xFF == ord("q"):
                    break
            else:
                break

        # Release the video capture and destroy the window
        self.cap.release()
        cv2.destroyAllWindows()
```

- run: A method responsible for processing video frames until the end of the video.
- Uses a loop to read frames from the video.
- Calls process_frame to handle each frame.
- Exits the loop if the 'q' key is pressed.
- Releases the video capture and closes the window after processing.

- **Process_Frame Method**

```python
def process_frame(self, frame):
    results_list = self.model(frame, verbose=False, conf=0.50)

    for results in results_list:
        for bbox in results.boxes.xyxy:
            x1, y1, x2, y2 = bbox[:4]

            x_center = (x1 + x2) / 2
            y_center = (y1 + y2) / 2

            print(f"Ball coordinates: (x={x_center:.2f}, y={y_center:.2f})")

            self.update_dribble_count(x_center, y_center)

            self.prev_x_center = x_center
            self.prev_y_center = y_center

        annotated_frame = results.plot()

        # Draw the dribble count on the frame
        cv2.putText(annotated_frame, f'Dribble Count: {self.dribble_count}', (50, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        cv2.imshow("YOLOv8 Inference", annotated_frame)
```

- process_frame: Method to handle individual video frames.
- Uses the YOLO model to detect objects in the frame.
- Iterates through detected results (bounding boxes).
- Prints ball coordinates and updates dribble count.
- Plots annotated results on the frame and displays it with dribble count.

- **Update_Dribble_Count Method**

```python
    def update_dribble_count(self, x_center, y_center):
        if self.prev_y_center is not None:
            delta_y = y_center - self.prev_y_center

            if (
                self.prev_delta_y is not None
                and self.prev_delta_y > self.dribble_threshold
                and delta_y < -self.dribble_threshold
            ):
                self.dribble_count += 1

            self.prev_delta_y = delta_y
```

  - update_dribble_count: Method to update the dribble count based on ball movement.
  - Calculates the change in y-coordinate.
  - If the conditions for a dribble are met, increment the dribble count.
  - Updates the previous change in y-coordinate.

- **Main Section**

```python
if __name__ == "__main__":
    video_path = "C:/Users/Admin/Downloads/WHATSAAP ASSIGNMENT.mp4"
    dribble_counter = DribbleCounter(video_path)
    dribble_counter.run()
#10-97
```

  - Specifies the path to the input video.
  - Creates an instance of DribbleCounter and calls the run method.

- **Results**
  - The provided code currently outputs a count of 97 for basketball dribbles, while the actual count is 106, indicating a discrepancy of 10 missed approximations.

- Output Image

- The method above is implemented by a third-party model, so I can't make more changes. Therefore, I attempted my own approach. Let's discuss it below.

Method 2

- **Code with Explanation**
  - **Importing Libraries**

```
import cv2
import numpy as np
import mediapipe as mp
```

- **Mediapipe Hands Initialization:**

```
# Initialize Mediapipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
```

  - Initializes the Mediapipe Hands module.

- **Video Reading**

```
cap = cv2.VideoCapture('C:/Users/Admin/Downloads/WHATSAAP ASSIGNMENT.mp4')
```

  - Reads the input video using OpenCV.

- **Touch Counting Logic**

```
touch_count = 0
is_touched = False
distance_sequence = []

while True:
    success, frame = cap.read()
    if not success:
        break
```

  - Initializes variables for touch count, touch status, and a sequence to store distances.
  - Uses a loop to process each frame from the video.

- **Color Space Conversion and Masking:**

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
lower_yellow = np.array([20, 100, 100])
upper_yellow = np.array([30, 255, 255])
mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
```

  - Converts each frame to HSV color space.
  - Defines a color range for yellow.
  - Creates a mask to isolate yellow regions.

- **Circle Detection:**

```
blurred = cv2.GaussianBlur(mask, (11, 11), 0)
circles = cv2.HoughCircles(
    blurred,
    cv2.HOUGH_GRADIENT,
    dp=1,
    minDist=40,
    param1=50,
    param2=20,
    minRadius=10,
    maxRadius=50
)
if circles is not None:
    # ...
```

  - Applies GaussianBlur to reduce noise.
  - Uses Hough Circle Transform to detect circles (representing the basketball).
  - Draws circles on the frame.

- **Hand Tracking with Mediapipe:**

```
rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
results = hands.process(rgb_frame)

# Check if hands are detected
if results.multi_hand_landmarks and len(results.multi_hand_landmarks) >= 2:
    # ...
```

  - Processes the frame with the Mediapipe Hands module.
  - Retrieves landmarks of the middle finger tips for both hands.

- **Distance Calculation and Visualization:**

```
distance_left = np.sqrt((left_middle_finger_x - x)**2 + (left_middle_finger_y - y)**2)
distance_right = np.sqrt((right_middle_finger_x - x)**2 + (right_middle_finger_y - y)**2)
```

  - Calculates the distance between the basketball center and each hand's middle finger tip.
  - Displays the distances on the frame.
  - Updates the minimum distance and appends it to a sequence.

- **Touch Detection and Counting:**

```
# ...
# Check if the hand and circle touch
if min_distance < r:
    if not is_touched:
        is_touched = True
else:
    is_touched = False

# Check if the sequence of minimum distances is less than 100
if len(distance_sequence) >= 3 and all(d < 100 for d in distance_sequence[-3:]):
    # Increase the touch count
    touch_count += 1
    # Reset the distance sequence
    distance_sequence = []
```

  - Check if the hand and the circle touch based on minimum distance.
  - Counts touch and resets the distance sequence if consecutive small distances are detected.

- **Display Results:**

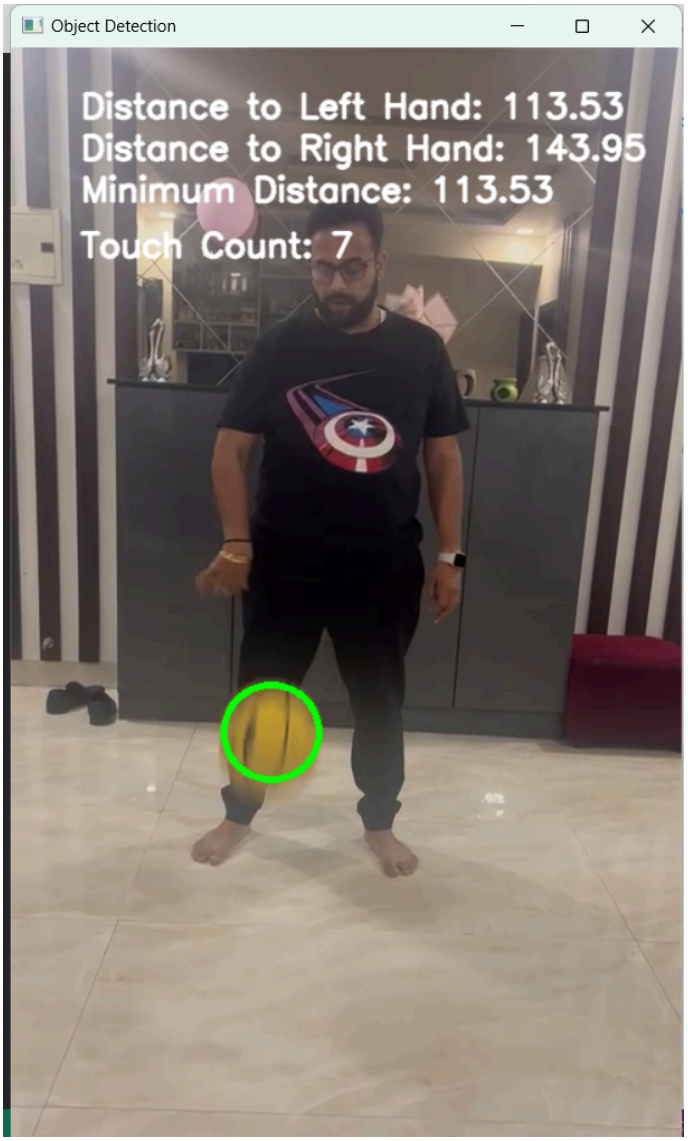
```
# Display the resulting frame
cv2.imshow('Object Detection', frame)

# Break the loop if the 'q' key is pressed
if cv2.waitKey(85) & 0xFF == ord('q'):
    break
```

  - Displays the frame with drawn circles, hand landmarks, and distance information.
  - Updates the touch count on the frame.
  - Breaks the loop if the 'q' key is pressed.

- **Result**
  - The provided code currently outputs a count of 78 for basketball dribbles, while the actual count is 106, indicating a discrepancy of 29 missed approximations.

- Output Image



- The above-discussed approach may not be the most optimized one, and there is a chance for miscalculation in distance.

Trajectory of the Ball

- **Code with Explanation**
  - **Initialization**

```
fourcc = cv2.VideoWriter_fourcc(*'XVID')
output_path = 'output_trajectory.avi'
out = cv2.VideoWriter(output_path, fourcc, 20.0, (visible_width, visible_height))
```

- **Color Range and Masking**

```
colourLow = (0, 100, 50)
colourHigh = (20, 255, 255)

blurred = cv2.GaussianBlur(frame, (11, 11), 0)
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

mask = cv2.inRange(hsv, colourLow, colourHigh)
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)
```

  - A color range for detecting the basketball is defined in HSV color space.
  - The video frames are processed to create a binary mask based on the defined color range.

- **Contour Detection and Trajectory**

```
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
center = None

if len(cnts) > 0:
    c = max(cnts, key=cv2.contourArea)
    ((x, y), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

    if radius > 15 and cv2.contourArea(c) > 150:
        cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
        cv2.circle(frame, center, 5, (0, 0, 255), -1)

        trajectory_points.append(center)

        for i in range(1, len(trajectory_points)):
            cv2.line(frame, trajectory_points[i - 1], trajectory_points[i], (0, 255, 0), 2)
```

  - Contours of the detected object are found, and the largest contour is selected as the basketball.
  - The center and radius of the minimum enclosing circle of the contour are determined.
  - If the radius and contour area conditions are met, the basketball is outlined, its center marked, and its trajectory traced.

- **Video Display and Writing**

```
out.write(frame)

cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF

if key == ord("q"):
    break
```

  - The processed frame is displayed in a window.
  - The frame is saved into the output video file.

- **Results**
  - Output Image

- Dribble trajectories are crucial for sudden changes in direction.
- Precise dribble trajectories contribute to better ball control

Optical Flow of Objects

- **Code with Explanation**
  - **Video Capture and Initialization**

```
video_path = 'C:/Users/Admin/Downloads/WHATSAAP ASSIGNMENT.mp4'
cap = cv2.VideoCapture(video_path)  # or cv2.VideoCapture(0) for webcam
```

    - The code initializes video capture from either a specified video file or the webcam if no video path is provided.
  - **Initial Frame and Feature Points**

```
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, maxCorners=100, qualityLevel=0.3, minDistance=7, blockSize=7)
```

    - The first frame is captured and converted to grayscale (old_gray).
    - A set of feature points (p0) is detected in the initial frame using the Shi-Tomasi corner detection method.
  - **Optical Flow Calculation**

```
p1, st, err = cv2.calcOpticalFlowPyrLK(
    old_gray, frame_gray, p0, None, winSize=(15, 15), maxLevel=2,
    criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03)
)
```

    - Inside the main loop, optical flow is calculated between consecutive frames using the Lucas-Kanade method (cv2.calcOpticalFlowPyrLK).
    - The function returns the new points (p1) based on optical flow, a status array (st), and an error array (err).

- **Selecting Good Points and Drawing Tracks**

```
good_new = p1[st == 1]
good_old = p0[st == 1]

for i, (new, old) in enumerate(zip(good_new, good_old)):
    a, b = new.ravel().astype(int)
    c, d = old.ravel().astype(int)
    mask = cv2.line(mask, (a, b), (c, d), (0, 255, 0), 2)
    frame = cv2.circle(frame, (a, b), 5, (0, 0, 255), -1)
```

  - Only the "good" points (where optical flow could be found) are selected.
  - The code iterates through the good points and draws tracks on the frame using lines and circles.

- **Display and Exit Condition**

```
cv2.imshow('Optical Flow', img)
k = cv2.waitKey(30) & 0xff
if k == 27:  # Press 'Esc' to exit
    break
```

  - The frame with drawn tracks is displayed, and the loop continues until the 'Esc' key is pressed.

- **Results**
  - Optical flow helps analyze the motion of objects within a video. This can be useful in applications such as surveillance, where detecting movement is crucial.

- Outputimage

Automatically Detecting Outlines And Borders of Objects

- **Code with Explanation**
  - **Tkinter GUI Setup**

```python
import random

class VideoProcessor:
    def __init__(self, root):
        self.root = root
        self.root.title("Video Processor")

        self.instructions = [
            "1. After clicking 'Run Video Processing,' press 's' to pause the video.",
            "2. After pressing 's,' use the mouse to click and crop the image, then press 'Enter.'",
            "3. After pressing 'Enter,' you can view the cropped image.",
            "4. After viewing the cropped image, press 'c' to clear and repeat the process of clicking and cropping the image to view the cropped image again.",
            "5. Only when you are viewing the cropped image can you quit the repeating process by pressing 'q.'",
            "6. After pressing 'q,' you will return to resume the video that was paused, and you can repeat the process."
        ]

        self.instructions_label = tk.Label(root, text="\n".join(self.instructions), justify='left')
        self.instructions_label.pack()

        self.run_button = tk.Button(root, text="Run Video Processing", command=self.run_video_processing)
        self.run_button.pack()

        self.exit_button = tk.Button(root, text="Exit", command=self.root.destroy)
        self.exit_button.pack()
```

    - This initializes the Tkinter GUI window, sets up labels for instructions, and creates buttons for running video processing and exiting the application.
  - **Video Processing Loop**

```python
def run_video_processing(self):
    messagebox.showinfo("Video Processor", "Video processing started. Press 's' to pause the video.")
    self.root.update_idletasks()  # Update the Tkinter window

    input_video_path = "C:/Users/Admin/Downloads/WHATSAAP ASSIGNMENT.mp4"
    cap = cv2.VideoCapture(input_video_path)

    if not cap.isOpened():
        messagebox.showerror("Video Processor", "Error opening video")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            messagebox.showerror("Video Processor", "Video frame completed ")
            break

        cv2.imshow('Video Viewer', frame)
        self.root.update_idletasks()  # Update the Tkinter window

        key = cv2.waitKey(1) & 0xFF

        if key == ord('s'):
            self.user_interactions(frame, cap)
        elif key == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

- This function runs the video processing loop. It displays each frame, allowing the user to interact by pressing 's' to pause the video or 'q' to exit. When 's' is pressed, it calls self.user_interactions for further interactions.

- **User Interactions**

```
def run_video_processing(self):
    def user_interactions(self, img, cap):
        original_img = img.copy()

        while True:
            roi = cv2.selectROI(img)
            cv2.destroyAllWindows()

            x, y, w, h = roi

            if x < 0 or y < 0 or x + w > img.shape[1] or y + h > img.shape[0]:
                print("Invalid ROI. Please try again.")
                img = original_img.copy()
                continue

            cropped_img = img[y:y+h, x:x+w]

            # Perform image processing techniques (resize, brighten, etc.)
            processed_img = self.process_cropped_image(cropped_img)

            # Display the processed image
            cv2.imshow('Processed Cropped Image', processed_img)

            key = cv2.waitKey(0) & 0xFF
            if key == ord('q'):
                break
            elif key == ord('c'):
                img = original_img.copy()

            # Save the processed image with a random number in the name
            save_dir = 'E:/computer_vision_intern/images/'
            if not os.path.exists(save_dir):
                os.makedirs(save_dir)
            try:
                img_name = f'processed_cropped_image_{random.randint(1, 100000)}.jp
                full_path = os.path.join(save_dir, img_name)
                cv2.imwrite(full_path, processed_img)
```

- This function handles user interactions for selecting a region of interest (ROI). It allows the user to draw a rectangle, and when 'Enter' is pressed, it processes the cropped image using self.process_cropped_image. The processed image is displayed and saved with a random filename.

- **Image Processing**

```
def process_cropped_image(self, img):
    # Resize the image to (416, 416)
    resized_img = cv2.resize(img, (416, 416))

    # Brighten the image
    brightness_factor = 1.5
    brightened_img = cv2.convertScaleAbs(resized_img, alpha=brightness_factor, beta=0

    return brightened_img
```

- This function processes the cropped image by resizing it to (416, 416) and applying brightness adjustment.

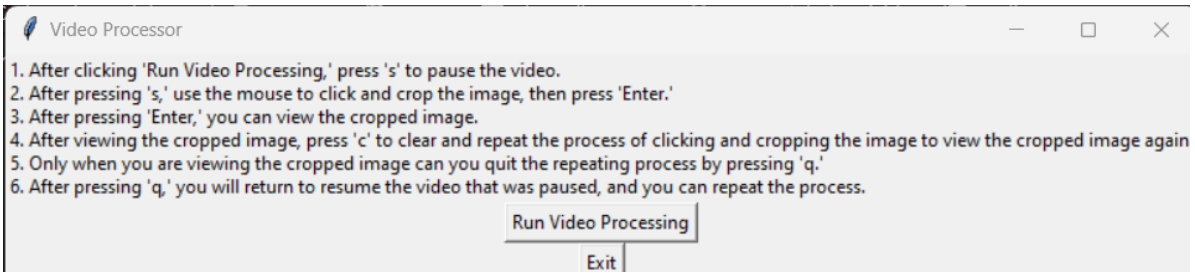
- **Main Execution**

```
if __name__ == "__main__":
    root = tk.Tk()
    video_processor = VideoProcessor(root)
    root.mainloop()
```

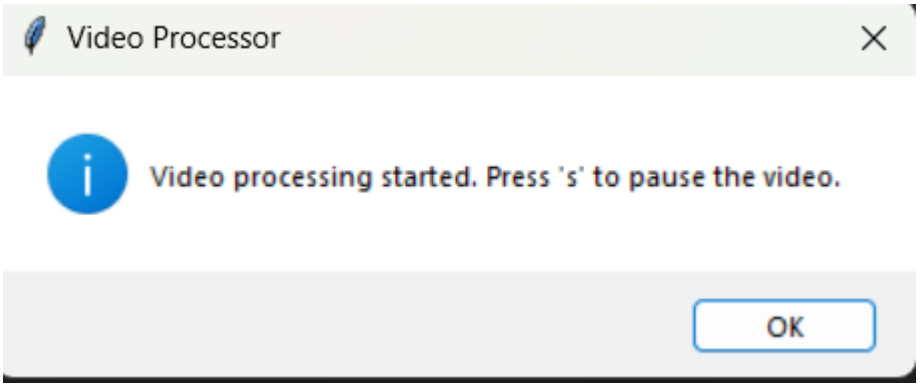
- This part of the code creates an instance of the VideoProcessor class and starts the Tkinter main loop.
- The code integrates Tkinter for GUI, OpenCV for video processing, and user interactions to select and process regions of interest in video frames. The processed images are saved with unique filenames in a specified directory.

- **Results**
  - a video processing application using Tkinter for the graphical user interface (GUI) and OpenCV for video manipulation. The GUI displays a set of instructions for the user to interact with the video processing functionalities. Upon clicking the "Run Video Processing" button, the code initiates the video processing loop, where frames from a specified video file are displayed. The user can pause the video ('s' key) to select a region of interest (ROI) by dragging the mouse. The selected ROI is then processed, resized to (416, 416) pixels, brightened, and displayed to the user. The user can clear the ROI ('c' key) or quit the process ('q' key). Additionally, each processed image is saved to a directory specified as 'E:/computer_vision_intern/images/' with a randomly generated filename. The code demonstrates the integration of Tkinter and OpenCV for creating an interactive video processing tool with user-friendly instructions and image saving capabilities.
  - **Output Images**

    Instructions



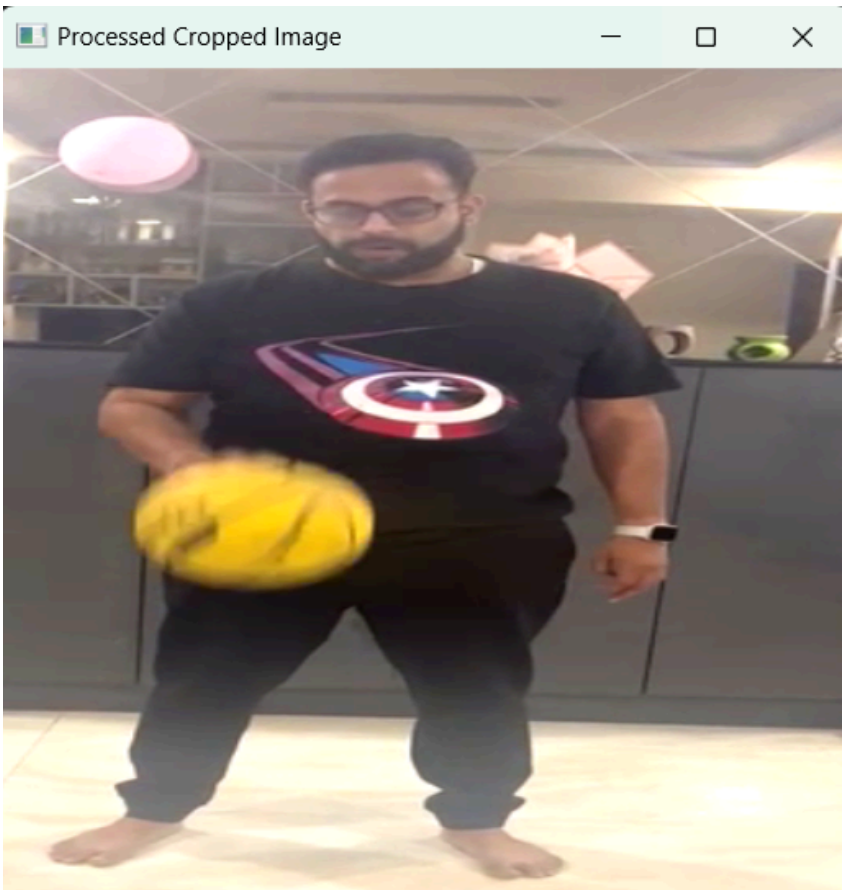
After Clicking Run Video Processing



After pressing 's'

Use The Mouse To Click And Crop The Image, then Press Enter

Viewing The Cropped Image



Further Enhancement

For further enhancement, users can analyze how many times the video has been paused and an image has been cropped. To improve image quality, various image processing techniques can be implemented. These techniques may include methods to enhance edges, adjust contrast, or reduce noise, thereby contributing to a more sophisticated video processing tool. Users can track their interaction frequency through a counter, which can be displayed in the GUI or printed in the console.

Challenges In The Task

Many challenges were encountered during the development process, including issues with library versions, implementing user interface logic, and correcting logical mistakes based on the initial idea. These challenges were overcome by addressing each issue step by step. For library versions, ensuring compatibility and updating as needed resolved the problem. Implementing the user interface logic involved careful design and integration to make the tool user-friendly. Logical mistakes were corrected through debugging and refining the underlying ideas. In simpler terms, overcoming these hurdles required a systematic approach and problem-solving skills to create a more robust and functional video processing tool.

Conclusion

I pursued my initial ideas step by step, encountering challenges along the way. Through trial and error, I made mistakes, but each one became a learning opportunity. Despite facing difficulties with library versions, implementing UI logic, and refining the overall logic, I persevered and successfully achieved my goals. These challenges not only enhanced my problem-solving skills but also provided valuable insights into the iterative development process. Moving forward, I am better equipped to tackle similar projects and continue learning from both successes and mistakes.