Badminton Court Detection Using Computer Vision

1: **Source Code**

This code displays frames of videos inside the boundary of the points I selected manually,using mouse control.

```
import cv2
import numpy as np

# Loading sample video
video_path = "sample_video.mp4"
cap = cv2.VideoCapture(video_path)

# Getting video properties
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Defining output video writer
fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Codec for output video
out = cv2.VideoWriter("masked_flipped_output.mp4", fourcc, fps,
(frame_width, frame_height))

# Defining polygon coordinates (court area) found manually
pts = np.array([[750, 575], [1200, 575], [1654, 963], [269, 961]],
np.int32)
pts = pts.reshape((-1, 1, 2))

while cap.isOpened():
   ret, frame = cap.read()
   if not ret:
       break

   flipped_frame = cv2.flip(frame, -1)
```

```
    # Creating a black mask
    mask = np.zeros(flipped_frame.shape[:2], dtype=np.uint8)

    cv2.fillPoly(mask, [pts], 255)
    result = cv2.bitwise_and(flipped_frame, flipped_frame, mask=mask)

    out.write(result)
    cv2.imshow("Masked & Flipped Video", result)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

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

Masked & Flipped Video Processing

This script processes a badminton court video by **flipping frames** and **applying a mask** to isolate the court area. The output is saved as a new video.

A. Loading the Video

The script reads a video (`sample_video.mp4`) and extracts its properties (frame width, height, and FPS).

B. Setting Up the Output Video

A video writer (`mp4v` codec) is initialized to save the processed frames.

C. Defining the Court Mask

A polygon representing the court area is defined using four coordinate points. A black mask is created, and the polygon is filled with white to highlight the court region.

D. Processing Each Frame

- The video is read **frame by frame**.
- Each frame is **flipped both vertically and horizontally**.
- The mask is applied using **bitwise operations**, keeping only the court area.
- The processed frame is written to the output video.

E. Display & Exit

The masked video is displayed in real-time. Pressing ‘q’ exits the loop.
Finally, all resources are released, and the output video is saved.

Code for mouse selection:

```
import cv2
import numpy as np


image = cv2.imread("court_frame.png")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Applying Canny edge detection
edges = cv2.Canny(gray, 50, 150)

# Finding contours
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Sorting by area and take the largest contour (assuming it's the court)
contours = sorted(contours, key=cv2.contourArea, reverse=True)

court_corners = []  # List to store detected court corners

if contours:
   largest_contour = contours[0]  # Biggest shape (court)(assumed)


   epsilon = 0.02 * cv2.arcLength(largest_contour, True)
   approx = cv2.approxPolyDP(largest_contour, epsilon, True)

   if len(approx) == 4:  # If it has four corners, store them
       court_corners = approx.reshape(4, 2)
       print("Automatically Detected Court Coordinates:", court_corners)


       for point in court_corners:
           cv2.circle(image, tuple(point), 10, (0, 0, 255), -1)
```

```
def mouse_callback(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        print(f"Clicked Coordinates: ({x}, {y})")
        court_corners.append([x, y])
        cv2.circle(image, (x, y), 10, (255, 0, 0), -1)
        cv2.imshow("Select Points", image)

# If detection is inaccurate, then manual selection
if len(court_corners) != 4:
    print("Click on the four corners of the court manually.")
    cv2.imshow("Select Points", image)
    cv2.setMouseCallback("Select Points", mouse_callback)
    cv2.waitKey(0)

# Displaying final detected court
cv2.imshow("Detected Court", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

A. Preprocessing & Edge Detection

- The image is converted to **grayscale**, and **Canny edge detection** is applied to highlight boundaries.

B. Contour Detection & Filtering

- Contours are extracted and sorted by area to identify the **largest shape** (assumed to be the court).
- The contour is approximated to a **quadrilateral** (four corners).

C. Automatic & Manual Corner Selection

- If **four corners** are detected, they are marked in red.
- If detection is inaccurate, the user can **click manually** to select court corners, marked in blue.

D. Displaying Output

- The detected or selected corners are displayed, and the final **court detection is shown**.

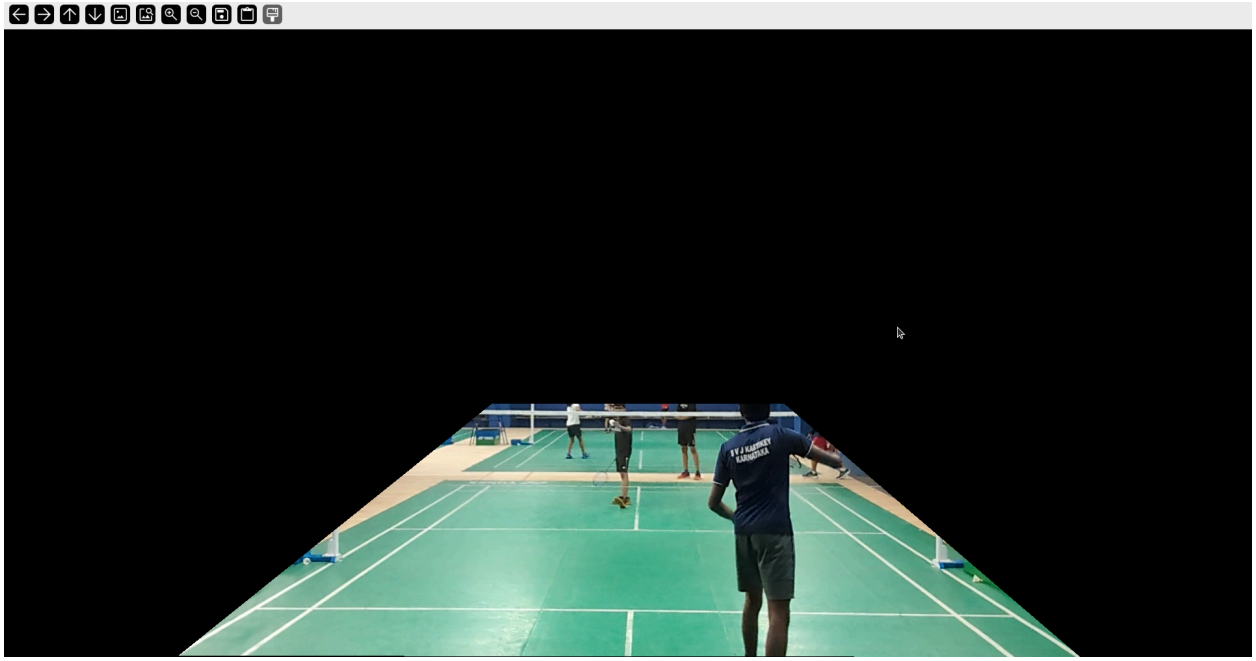
Here is the source code for Canny Edge Detection I tried to use for automatic edge detection using open cv library

```
import cv2
import numpy as np

# Load the video
cap = cv2.VideoCapture("sample_video.mp4")

if not cap.isOpened():
   print("Error: Could not open video file.")
   exit()

while cap.isOpened():
   ret, frame = cap.read()
   if not ret:
       print("End of video or error in reading frame.")
       break


   frame = cv2.flip(frame, -1)

   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
   blurred = cv2.GaussianBlur(gray, (5, 5), 0)

   edges = cv2.Canny(blurred, 50, 150)

   contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

   print(f"Contours found: {len(contours)}")

   if contours:

       court_contour = max(contours, key=cv2.contourArea)
```

```
        mask = np.zeros_like(frame)


        cv2.drawContours(mask, [court_contour], -1, (255, 255, 255),
thickness=cv2.FILLED)

        cropped_court = cv2.bitwise_and(frame, mask)

        cv2.imshow("Cropped Court", cropped_court)

    cv2.imshow("Original Frame", frame)
    cv2.imshow("Edges", edges)
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

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

2:Results

Here is the screenshot of the final video



3:Instructions to Run

Requirements

Ensure you have the following installed:

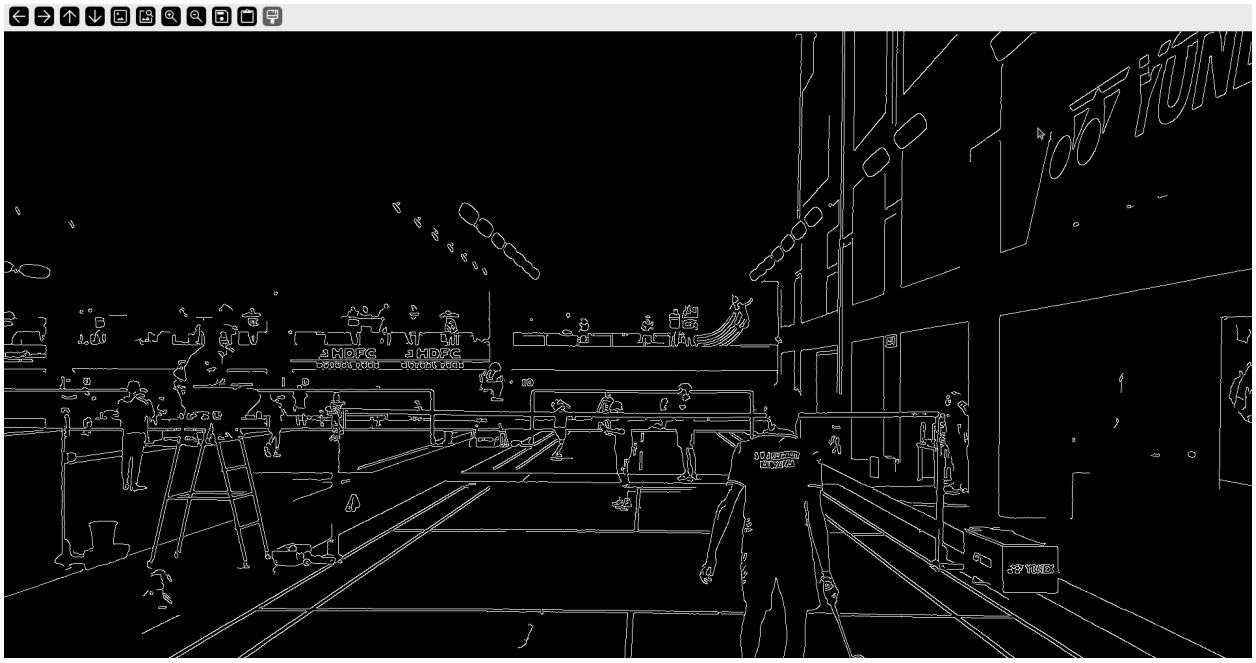
- Python 3.x
- OpenCV (`pip install opencv-python`)
- NumPy (`pip install numpy`)

Run the Code

1. Save the script as `court_detection.py`.
2. Place your **badminton court image** in the same folder.
3. Run the script using:
   python court_detection.py
4. The detected court will be displayed in a new window.

4:Limitations

- Automatic edge detection did not work properly.Here is a screenshot of the contours found using canny edge detection:



- I did not figure out yet how to filter the edges automatically,so I did the filtering manually.
- The assumption that the court is the biggest shape was not working.

5:Future Scope

- Figure out the algorithm for automatic boundary detection.
- Explore the possibility of deep learning models
- Implement a user interface or script parameters to adjust the
- cropping/masking dynamically.

6:Github Link

LINK