Part1:

1. Theory Explanation

1.1 Mean Shift Segmentation

The code applies 'Mean Shift Segmentation', a procedure that smoothens an image by clustering pixel values. This non-parametric feature-space analysis technique helps in reducing noise and improving the quality of the image for further processing.

1.2 Circle Detection using Hough Transform

After image segmentation and conversion to grayscale, the code employs the Hough Circle Transform for circle detection. This algorithm is adept at finding imperfect instances of circular shapes in an image, making it ideal for detecting objects like balls.

2. Code Analysis

2.1 Import Libraries: The code uses cv2 (OpenCV), numpy, and glob libraries. OpenCV is an open-source computer vision library, numpy supports large, multi-dimensional arrays, and glob helps with file path management.

3. Image Processing Loop:

3.1 Reading Images: Each image is read into an array.

3.2 Mean Shift Segmentation: cv.pyrMeanShiftFiltering is applied to each image for segmentation.

3.3 Grayscale Conversion and Blurring: The segmented image is converted to grayscale and blurred to reduce noise.

3.4 Circle Detection: cv.HoughCircles is used to detect circles in the processed image.

3.5 Drawing Circles: Detected circles are marked with red outlines.

3.6 Display and Break Option: Each processed image is displayed, and the loop can be broken by pressing 'q'.

3.7 Clean-up: cv.destroyAllWindows() is called at the end to close all OpenCV windows.

4. Processing Illustration

4.1 Segmentation: It enhances the clarity of the objects (balls) against varied backgrounds.

4.2 Grayscale and Blurring: Simplifies the image, focusing on structural details.

4.3 Circle Detection: Detects and marks circular shapes (balls) in the image.

5. Summary Statistics

5.1 Number of Images Processed: Count of img_files.

5.2 Number of Circles Detected per Image: Length of circles array for each image.

5.3 Average Radius of Detected Circles: Average of circle radii detected

across all images.

6. Weakness Analysis (Robustness of Approach)

6.1 Sensitivity to Parameters: The performance of mean shift segmentation and Hough Circle Transform heavily depends on the choice of parameters like sp, sr, param1, param2, etc. Incorrect parameter values can lead to poor segmentation or circle detection.

6.2 Error Handling: There is a lack of error handling, which could be problematic with corrupt or incompatible image files.

6.3 User Interaction: The need for manual intervention (pressing 'q') to proceed might not be ideal for automated processes.

Part2 -Watershed Segmentation

1. Theory Explanation

1.1 Watershed Algorithm

The Watershed Algorithm is a technique used in image processing to identify and segregate distinct regions within an image. It's particularly effective

for separating overlapping objects and complex structures in an image.

1.2 User Input for Foreground/Background Identification

The code allows the user to draw on the image to mark areas as either foreground or background. This manual input is critical for informing the Watershed Algorithm about different regions in the image.

2. Code Analysis

2.1 Global Variables and Mouse Callback:

The code uses global variables to track drawing states and store user input curves. A mouse callback function (draw_curve) updates these variables based on user interactions.

2.2 Image Loading and Window Initialization:

The image is loaded, resized, and displayed in a window.

The window is set to listen to mouse events, enabling user interaction.

2.3 Main Loop for User Interaction:

Users can switch between marking the foreground and background.

Drawing functions update the image based on user input.

2.4 Image Processing for Segmentation:

After user input, the image is converted to grayscale and thresholded.

Markers for the Watershed Algorithm are initialized and applied to the foreground and background curves.

The Watershed Algorithm is applied, identifying segment boundaries.

2.5 Visualization: The segmented image with highlighted boundaries is displayed.

2.6 Cleanup: The OpenCV windows are destroyed at the end of the session.

3. Processing Illustration

3.1 User Interaction: Drawing on the image to define foreground and background.

3.2 Watershed Segmentation: Applying the algorithm to segment the image based on user-defined regions.

3.3 Boundary Highlighting: Displaying the segmented regions with marked boundaries.

4. Summary Statistics

4.1 Number of Foreground Regions: Length of foreground_curves.

4.2 Number of Background Regions: Length of background_curves.

4.3 Total Number of User-Defined Points: Sum of points in foreground_curves and background_curves.

5. Weakness Analysis (Robustness of Approach)

4.1 Dependence on User Input: The algorithm's effectiveness heavily relies on accurate and comprehensive user input. Inaccurate or insufficient markings can lead to poor segmentation.

4.2 Subjectivity: Different users might define foreground and background

differently, leading to varying results.

4.3 Scalability Issues: For large or complex images, marking the regions manually can be time-consuming and impractical.

4.4 Lack of Automation: The approach lacks automation, making it unsuitable for applications requiring rapid or bulk processing.

4.5 No Undo Functionality: The program does not support undoing or editing previous markings, which could lead to errors during user interaction.

4.6 Lack of Error Handling: There is no explicit error handling for situations like file not found or incompatible file formats.

Part2 - Back Projection

1. Theory Explanation

1.1 Color Filtering and Histogram Back projection

Color Filtering:

It involves selecting a specific color range (blue) to identify relevant objects. This step is useful in scenarios where the object of interest has a distinct color.

Histogram Back projection:

This technique creates a model of the color distribution of an object and then uses this model to find the object in an image. It's particularly effective for finding objects with varying shapes but consistent color patterns.

Perspective Transformation

This is a geometric transformation that maps the points of a view onto a different view. It's used here to transform the identified table to a standard orientation and size.

2. Code Analysis

2.1 Utility Functions:

rescale_image: Resizes images.

calculate_histogram: Calculates color histograms in HSV space.

order_points: Orders the corner points of a quadrilateral.

2.2 Image Loading and Preprocessing:

Loads target and sample images and rescales them.

Applies color filtering to the target image to isolate blue regions.

2.3 Histogram Backprojection:

Calculates histograms for sample images and combines them.

Applies backprojection on the target image using the combined histogram.

2.4 Post-Processing and Contour Analysis:

Thresholds and dilates the combined backprojection result.

Finds contours and identifies the largest quadrilateral, presumed to be a table.

2.5 Perspective Transformation:

If a quadrilateral is found, it performs a perspective transformation to standardize its view.

2.6 Visualization and Cleanup:

Displays the warped image and the original image with the detected table outlined.

Destroys all OpenCV windows after keypress.

3. Processing Illustration

3.1 Color Filtering and Histogram Backprojection: These steps isolate the table based on color characteristics.

3.2 Finding the Largest Quadrilateral: Identifies the largest table-shaped contour.

3.3 Perspective Transformation: Transforms the identified table to a standardized view.

4. Summary Statistics

4.1 Number of Tables Detected: Based on the presence of largest_quad.

4.2 Area of the Largest Detected Table: largest_area.

5. Weakness Analysis (Robustness of Approach)

5.1 Color Dependency: The effectiveness depends on the distinct color of the tables, which may not be consistent across all images.

5.2 Histogram Model Generalization: The combined histogram model might not accurately represent all table types, leading to potential misidentification.

5.3 Manual Parameters: The method uses manually set parameters (like clipLimit, tileGridSize, kernel size), which may not be optimal for all images.

5.4 Quadrilateral Assumption: The assumption that the largest quadrilateral is the table may not always hold true, especially in cluttered or complex scenes.

5.5 Lack of Error Handling: The code doesn't include error handling for file loading or processing failures.

Part3

1. Theory Explanation

1.1 Color Filtering and Contour Analysis

Color Filtering:

The code uses HSV color space to filter the color of the table tennis ball. HSV is often preferred over RGB for color filtering due to its better separation of color information from lighting conditions.

Contour Analysis:

After color filtering, the code applies contour detection to identify the

ball's position in each frame.

Bounce and Hit Detection:

The code attempts to detect bounces and hits by analyzing the movement of the ball. It uses the change in the ball's trajectory and position relative to the table's middle line to infer these events.

2. Code Analysis

2.1 Video Capture and Initialization:

Uses cv2.VideoCapture to load a video file.

Initializes variables for tracking and detecting the ball's position, bounces, and hits.

2.2 Main Loop for Video Processing:

Reads frames from the video and processes them.

Converts each frame to HSV for color filtering and grayscale for frame differencing.

2.3 Ball Detection and Tracking:

Applies a mask to detect the ball and uses contour analysis to find its position.

Tracks the ball across frames to detect its absence, bounces, and player hits.

2.4 Bounce and Hit Logic:

Analyzes the ball's trajectory for bounce detection and hit detection based on ball movement across the table's middle.

2.5 Visualization and User Interaction:

Draws the ball's position on each frame and displays messages for bounces and hits.

Allows user to exit the video playback with the 'Esc' key.

3. Processing Illustration

3.1 Color Filtering and Ball Detection: Identifies the table tennis ball in each frame.

3.2 Trajectory Analysis: Tracks the ball's movement to detect bounces and hits.

3.3 Display: Visualizes the ball's position and annotates detected events.

4. Summary Statistics

4.1 Total Frames Processed: Equivalent to current_frame at the end of the video.

4.2 Number of Bounces Detected: Counted every time the bounce message is displayed.

4.3 Number of Hits Detected: Counted based on the change in ball direction relative to the middle of the table.

4.4 Ball Position Detected: Detected based on x and y axis related to video size displayed

5. Weakness Analysis (Robustness of Approach)

5.1 Color Sensitivity: The approach is sensitive to the specific color range defined for the table tennis ball. Variations in lighting or ball color can affect accuracy.

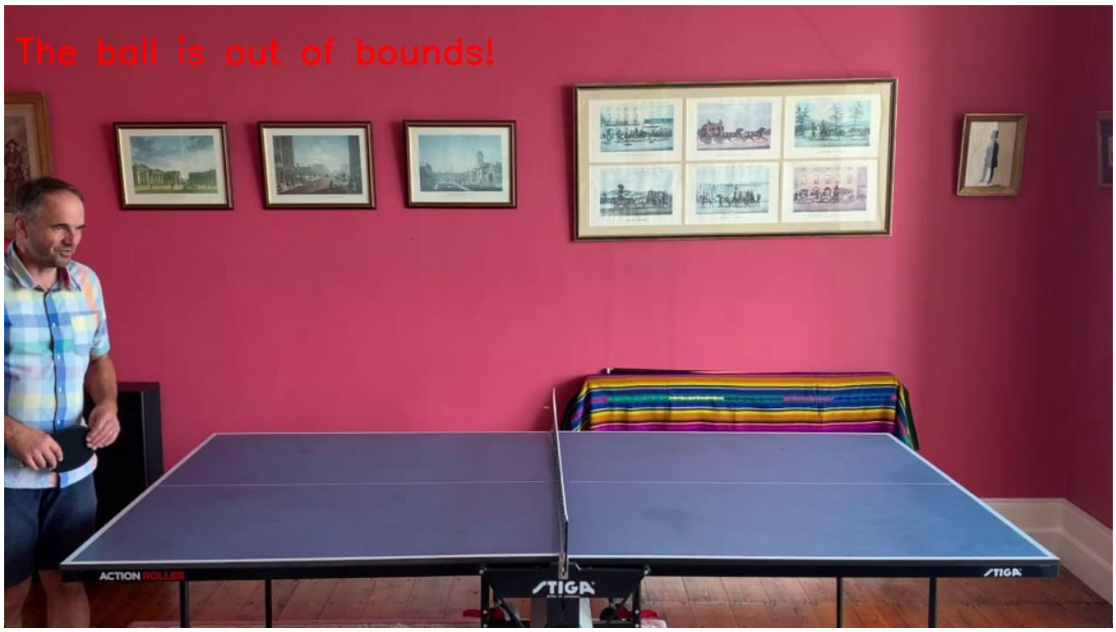
5.2 Frame Differencing Limitations: Relying on frame differencing for motion detection can be impacted by fast movements or camera shake.

5.3 Bounce Detection Reliability: Bounce detection is based on the change in y-coordinate trends, which might not be always reliable, especially in cases of fast or irregular bounces.

5.4 Hit Detection Assumptions: The logic for detecting hits assumes a straightforward back-and-forth pattern of play, which might not hold in more complex scenarios.

5.5 Manual Parameter Tuning: Parameters like contour area threshold and frame differencing threshold are hardcoded, which might not be optimal for all videos.


The ball bounced from the table!
ACTION ROLLER
STIGA
STIGA


The ball is out of bounds!
ACTION ROLLER
STIGA
STIGA